A Formal Requirements Modeling Approach: Application to Rail Communication

Steve J. Tueno Fotso[1,3], Régine Laleau[1], Hector Ruiz Barradas[2], Marc Frappier[3] and Amel Mammar[4]
[1]*Université Paris-Est Créteil, LACL, Créteil, France*
[2]*ClearSy System Engineering, Aix-en-Provence, France*
[3]*Université de Sherbrooke, GRIL, Québec, Canada*
[4]*Télécom SudParis, SAMOVAR-CNRS, Evry, France*

Keywords: *Saturn* Rail Communication Protocol, Requirements Engineering, Formal Models, Domain Modeling, Railway, *SysML/KAOS*, *B System*, *Event-B*.

Abstract: This paper is about the formal specification of requirements of a rail communication protocol called *Saturn*, proposed by ClearSy systems engineering, a French company specialised in safety critical systems. The protocol was developed and implemented within a rail product, widely used, without modeling, verifying and even documenting its requirements. This paper outlines the formal specification, verification and validation of Saturn's requirements in order to guarantee its correct behavior and to allow the definition of slightly different product lines. The specification is performed according to SysML/KAOS, a formal requirements engineering method developed in the *ANR FORMOSE* project for critical and complex systems. System requirements, captured with a goal modeling language, give rise to the behavioral part of a *B System* specification. In addition, an ontology modeling language allows the specification of domain entities and properties. The domain models thus obtained are used to derive the structural part of the *B System* specification obtained from system requirements. The *B System* model, once completed with the body of events, can then be verified and validated using the whole range of tools that support the *B* method. Five refinement levels of the rail communication protocol were constructed. The method has proven useful. However, several missing features were identified. This paper also provides a formally defined extension of the modeling languages to fill the shortcomings.

1 INTRODUCTION

Refinement-based methods that support proof-by-construction help to progressively construct the specification of a complex system while ensuring its correctness. In recent years, these methods have been widely used on large scale projects in critical areas such as railway or aeronautics, thanks in particular to the emergence of industrial strength formal methods such as *B* (Abrial, 2005) or *Event-B* (Abrial, 2010). A major difficulty identified in such projects lies in the construction of the initial formal model, based on needs identified by stakeholders (Lee and Suh, 2005; Abrial, 2010). When poorly conducted, this step is responsible for the vast majority of critical failures (Lee and Suh, 2005).

Our work focuses on the formal requirements modeling of critical and complex systems. *SysML/KAOS* (Fotso et al., 2018b; Laleau et al., 2010), as a requirements engineering method, is chosen because it includes an expressive and intuitive goal modeling language (Laleau et al., 2010) to represent system requirements extracted from artifacts that describe stakeholder needs. In addition, SysML/KAOS includes a domain modeling language (Fotso et al., 2018b; Tueno et al., 2017d) to represent application domain entities and their properties using ontologies. Furthermore, the rules required to generate a *B System* specification (ClearSy, 2014) from goal and domain models are defined and the most relevant ones have been formally verified (Fotso et al., 2018c). Goal models give the *behavioral part* (events) of the specification (Matoussi et al., 2011) while domain models provide its *structural part* (sets, constants and their properties, variables and their invariant) and the initialisation of state variables (Fotso et al., 2018c;

Fotso, S., Laleau, R., Barradas, H., Frappier, M. and Mammar, A.
A Formal Requirements Modeling Approach: Application to Rail Communication.
DOI: 10.5220/0007809701700177
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 170-177
ISBN: 978-989-758-379-7

Tueno et al., 2017a). Once the event bodies are specified, the *B System* model can be formally verified and validated to assess the requirements. This can be done using the full range of tools that support the *B* method (Abrial, 2005), largely and positively assessed on industrial projects for more than 25 years (Lecomte et al., 2017).

SysML/KAOS is supported by integrated development environments *Openflexo* (Openflexo, 2019) and *Atelier B* (ClearSy, 2014). Openflexo supports goal modeling while Atelier B supports the specification, verification and validation of *B System* models based on the semantics of SysML/KAOS modeling languages. Domain models, on the other hand, are constructed and translated to *B System* using a tool (Tueno et al., 2017b) implemented on top of *Jetbrains MPS* (Jetbrains, 2017) and PlantUML (Roques, 2015).

Saturn (Technology, 2019) is a rail communication protocol, proposed by *ClearSy*, which deals with exchanges of communication frames between rail agents through a bus. The protocol was developed and implemented within a rail product widely used, without modeling, verifying and even documenting its requirements. This paper outlines the formal specification, verification and validation of Saturn's requirements in order to guarantee the correctness of its behavior and to allow the definition of slightly different product lines. SysML/KAOS has provided a methodical and structured way for the formal specification of requirements. Furthermore, it ensures a better reusability and readability of models and a strong traceability between models. The specification is decomposed into five refinements. The use of SysML/KAOS on this case study led us to extend the domain modeling language and make it more suitable for use in system modeling: the language, described in (Tueno et al., 2017d; Tueno et al., 2017c; Fotso et al., 2018b) has been adjusted to allow the definition of associations between associations and to support variable data items. This completes the definition of the domain modeling language: associations have been generalised into concepts and variability has been extended to individuals. In addition, the translation rules (Fotso et al., 2018c; Tueno et al., 2017a) have been updated to match the adjusted language and formally verified.

The remainder of this paper is structured as follows: Section 2 briefly describes the *B System* formal method, the SysML/KAOS requirements engineering method and its goal and domain modeling languages, and the translation of SysML/KAOS models. Follows a presentation, in Section 3, of the work done on the case study and of updates performed on the SysML/KAOS domain modeling language. Finally, Section 4 discuss the work done and Section 5 reports our conclusion and future work.

2 BACKGROUND

2.1 Event-B and B System

Event-B (Abrial, 2010) is an industrial-strength formal method for *system modeling*. It allows the incremental construction of system specifications, using stepwise refinement, and the proof of useful properties. *B System* is an *Event-B* syntactic variant proposed by *ClearSy*, an industrial partner in the *FORMOSE* project (ANR-14-CE28-0009, 2017), and supported by *Atelier B* (ClearSy, 2014). It shares the same semantics with *Event-B*.

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 refinement is a component which refines another one in order to concretise the system construction: addition of functionalities or specification of the achievement of some purposes. Constants, abstract and enumerated sets (user-defined types), and their properties, constitute the static part. The dynamic part includes the representation of system state using variables constrained through invariants (first-order predicates that constrain the possible values that the variables may hold) and updated through events.

2.2 SysML/KAOS Goal Modeling

SysML/KAOS (Laleau et al., 2010) is a requirements engineering method which combines the traceability provided by *SysML* (Hause et al., 2006) with goal expressiveness provided by KAOS (van Lamsweerde, 2009). It allows the representation of requirements to be satisfied by a system and of expectations with regards to the environment through a hierarchy of goals. The goal 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. An ***OR refinement*** decomposes a goal into subgoals such that the achievement of only one of them is sufficient for the achievement of the parent goal.

2.3 SysML/KAOS Domain Modeling

Domain models in SysML/KAOS are represented using ontologies. These ontologies are expressed using

the SysML/KAOS domain modeling language (Tueno et al., 2017d; Tueno et al., 2017c), which is built based on *OWL* (Sengupta and Hitzler, 2014) and *PLIB* (Pierra, 2004).

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. *Concepts* (instances of Concept) denote collections of *individuals* (instances of Individual) with common properties. A concept can be declared *variable* when the set of its individuals can be updated by adding or deleting individuals. Otherwise, it is considered to be *constant*.

Relations (instances of Relation) are used to capture links between concepts, and *attributes* (instances of Attribute) capture links between concepts and *data sets* (instances of DataSet). *Predicates* (instances of Predicate) are used to represent constraints between different elements of the domain model in the form of *Horn clauses*.

2.4 Translation of SysML/KAOS Models

The formalisation of SysML/KAOS goal models is detailed in (Matoussi et al., 2011). 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 level of the goal hierarchy; this component defines one event for each goal. As the semantics of the refinement between goals is different from that of the refinement between *B System* components, new proof obligations for goal refinement are defined in (Matoussi et al., 2011). They depend on the goal refinement operator used and complete the *B System* proof obligations for invariant preservation and for event feasibility. For instance, the following proof obligations formalise the AND refinement of an abstract goal G into two concrete goals G_1 and G_2 (for an event `G`, `G_Guard` represents the guards of G and `G_Post` represents the post condition of its actions):

- $G_1\_Guard \Rightarrow G\_Guard$
- $G_2\_Guard \Rightarrow G\_Guard$
- $(G_1\_Post \wedge G_2\_Post) \Rightarrow G\_Post$

It should be noted that variables updated by subgoals must be distinct.

However, the *B System* specification generated from goal diagrams does not contain the static part and the state variables. These elements are provided by the translation of SysML/KAOS domain models. The corresponding rules are fully described in (Tueno et al., 2017a) and their formal verification is described in (Fotso et al., 2018c). In short, domain models identify *B System* components. A concept without a parent gives a *B System* abstract set. Each concept *C*, with parent *PC*, gives a formal constant, subset of the correspondent of *PC*. Relations and attributes give formal relations.

3 SPECIFICATION OF THE SATURN COMMUNICATION PROTOCOL

3.1 Main Characteristics of the Protocol

SATURN describes exchanges of communication frames between different agents connected via a bus so as to ensure high robustness and availability (Technology, 2019). It deals with one active gateway (and possibly seven passive ones), several input/output agents (1-128) and an innovative ring network. Input/output agents can be secure or unsecure. Input agents provide boolean data. The data are periodically collected by the gateway, transformed and the result is made available to output agents through the ring network.

3.2 Goal Modeling

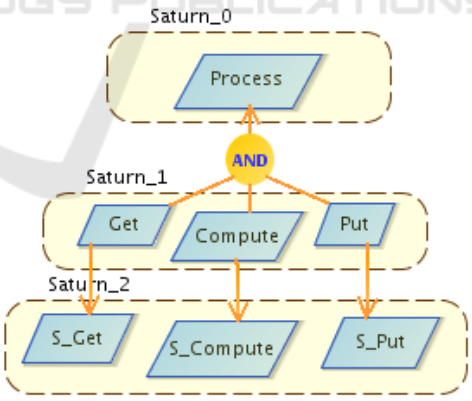
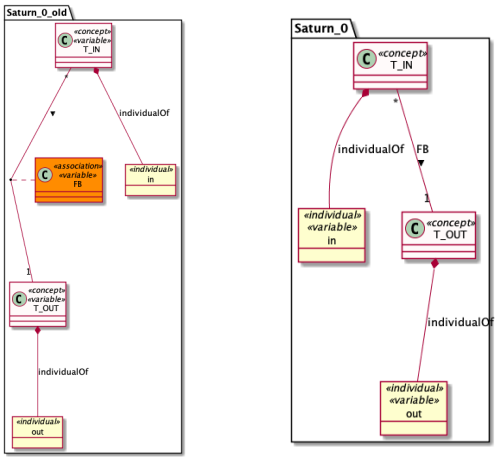


Figure 1: Excerpt from the SATURN protocol goal diagram.

Figure 1 is an excerpt from the SysML/KAOS goal diagram representing the functional goals of SATURN. The main purpose of the system is to transform data provided by input agents (`in`) and make the result (`out=FB(in)`) available to output agents: `FB` is a boolean function that transforms inputs into outputs. The purpose gives the most abstract goal *Process* of the goal diagram. Goal *Process* is refined us-



(a) (b)

Figure 2: ***Saturn_0***: root level ontology.

ing the AND operator to introduce a goal *Get* for input data acquisition from input agents and a goal *Put* to make the result available to output agents. The second refinement level is a *data refinement* which refines goals defined within the first refinement level to take into account multiplicities of input and output agents. Thus, input data acquisition (goal *S_Get*) generates a boolean array, the computation (goal *S_Compute*) becomes a transformation between arrays and result delivery (goal *S_Put*) transfers the resulting array to output agents.

3.3 Domain Modeling

3.3.1 First Attempt

Figure 2 (a) is an attempt to represent the domain model associated with the root level of the goal diagram of Fig. 1 using the SysML/KAOS domain modeling language previously described. It is illustrated using the syntax proposed by the *SysML/KAOS domain modeling tool* (Tueno et al., 2017b). For readability purposes, we have decided to hide the representation of optional characteristics. The type of input data is modeled as a concept `T_IN` defining an individual `in` which represents the input data. Similarly, the type of output data is modeled as a concept `T_OUT` defining an individual `out` which represents the output data. The computation function `FB` is modeled as a functional relation from `T_IN` to `T_OUT`.

The first difficulty we encountered is related to the changeability of domain entities. In fact, inputs and outputs change dynamically. In domain model of Fig. 2 (a), a workaround consisted in considering that concepts `T_IN` and `T_OUT` and relation `FB` are variables. In the *B System* specification, the variability is reflected through the definition of a variable subset (Fotso et al., 2018c; Tueno et al., 2017a): for instance, the variable concept `T_IN` gives a *B System* abstract set `T_IN` and a variable $x\_T\_IN \subseteq T\_IN$ which contains, at any system state, the current value of the input. Thus, going from a system state where `out1 = FB(in1)` to a system state where `out2 = FB(in2)` is feasible and goes through: (1) withdrawal of maplet $in1 \mapsto out1$ from `x_FB`; (2) withdrawal of individual `in1` from `x_T_IN`; (3) withdrawal of individual `out1` from `x_T_OUT`; (4) addition of individual `in2` in `x_T_IN`; (5) addition of individual `out2` in `x_T_OUT`; and (6) addition of maplet $in2 \mapsto out2$ in `x_FB`. Too many actions are thus required and the modeling does not conform to SATURN's specification. Indeed, from a conceptual point of view: (1) the input data type must be constant (corresponds to the set of n-tuples of booleans, when considering *n* input agents); (2) the output data type must be constant; (3) the computation function `FB` is hard-coded and is therefore constant. What should change are individuals representing input and output states. It is thus necessary to be able to model variable individuals: individual which can dynamically take any value in a given concept. A similar need appears for relations with relation maplets, attributes with attribute maplets and data sets with data values.

Another difficulty has been encountered related to multiplicities of input and output agents (domain model associated with the third refinement level of the goal diagram of Fig. 1). Indeed, the array that represents input data needs to be modeled by a relation, ditto for the array that represents output data. Thus, the computation function needs to be modeled by a relation for which the domain and the range are relations, which is impossible with the current definition of the SysML/KAOS domain modeling language.

The *SATURN* case study also revealed the need to be able to:

- define domain and range cardinalities for attributes;
- define a named maplet (instance of RelationMaplet or AttributeMaplet) with or without antecedent and image;
- define an initial value for a variable individual, maplet or data value;
- define associations between data sets and maplets between data values;
- refine a concept with an association or a data set;
- refine an individual with a maplet or a data value.

We have therefore identified the need to build a generalisation of the domain modeling language to

enrich its expressiveness while preserving the fundamental constraints identified in (Tueno et al., 2017c; Tueno et al., 2017d). A major contribution of this new metamodel is that it merges notions of concept, data set, attribute and relation as well as notions of individual, maplet and data value that have always been considered distinct by ontology modeling languages such as *OWL* (Sengupta and Hitzler, 2014). Additional constraints are defined to preserve the formal semantics of the language and to ensure unambiguous transformation of any domain model to a *B System* specification (Tueno et al., 2018).

3.3.2 The Revised Domain Modeling Language

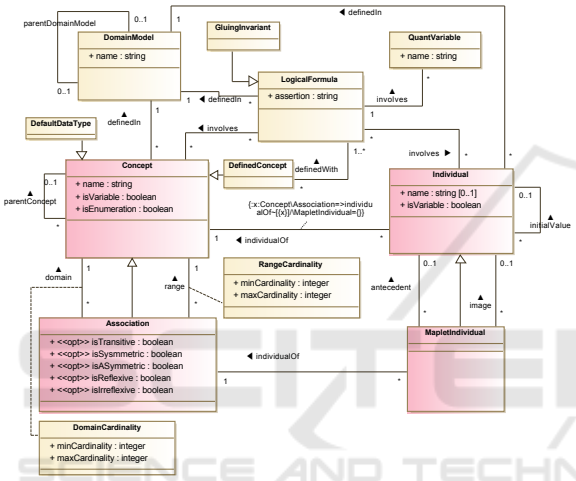


Figure 3: Excerpt from the updated SysML/KAOS domain metamodel.

Figure 3 is an excerpt from the revised SysML/KAOS domain metamodel. Major updates were made within the elements in pink. Classes Concept and DataSet have been merged into class Concept. In addition, classes Individual and DataValue have been merged into class Individual. A concept can now be an enumeration (*isEnumeration=TRUE*) if all its individuals are defined within the domain model. An individual can be *variable* (*isVariable=TRUE*) if it is introduced to represent a system state variable: it can represent different individuals at different system states. Otherwise, it is *constant* (*isVariable=FALSE*).

Associations (instances of Association) are concepts used to capture links between concepts. Class Association is used to merge classes Relation and Attribute. *Maplet individuals* (instances of MapletIndividual) capture associations between individuals. Each named maplet individual can reference an antecedent and an image. When the maplet individual is unnamed, the antecedent and the image must be specified. Class LogicalFormula replaces class Predicate to represent constraints between domain model elements. A defined concept (instance of class DefinedConcept) is a concept for which the type is provided by a logical formula.

Additional constraints are required to preserve the formal semantics of the domain modeling language and to ensure an unambiguous transformation of any domain model to a *B System* specification. The constraints are fully defined in (Tueno et al., 2018) using the *B* syntax (Abrial, 2010). For instance:

- $x \in \text{Concept} \setminus \text{Association} \Rightarrow individualOf^{-1}[\{x\}] \cap MapletIndividual = \emptyset$: if a concept `x` is not an association, then no individual of `x` can be a maplet individual.
- $x \in MapletIndividual \cap dom(antecedent) \Rightarrow antecedent(x) \in domain(individualOf(x))$: antecedents of a maplet individual must be individuals of the domain of its association.
- $x \in \text{Concept} \setminus (\text{Association} \cup dom(parentConcept)) \Rightarrow Concept\_isVariable(x) = FALSE$: every abstract concept (that has no parent concept) that is not an association must be constant. Abstract concepts that are associations can be variable.

3.3.3 The Saturn Protocol Domain Model

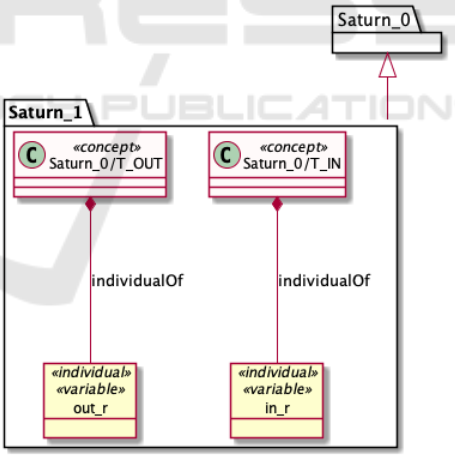


Figure 4: ***Saturn_1***: ontology associated with the first refinement level.

Figures 2 (b), 4 and 5 represent domain models associated with the refinement levels of the goal diagram of Fig. 1 using the updated SysML/KAOS domain modeling language.

In domain model `Saturn_0` (Fig. 2 (b)), the type of input data is modeled as a constant concept `T_IN` (instance of class Concept of Fig. 3) defining a variable individual `in` (instance of class Individual of Fig. 3) which represents the input data. Similarly, the type of output data is modeled as a constant concept `T_OUT`

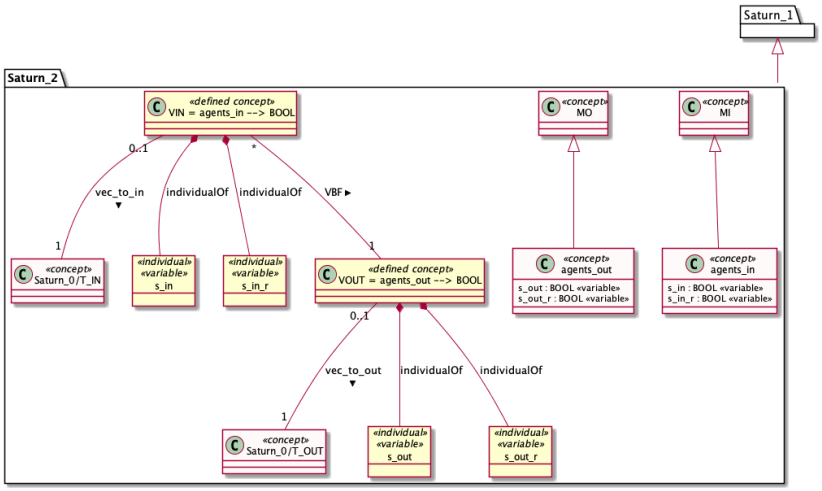


Figure 5: ***Saturn_2***: ontology associated with the second refinement level.

defining a variable individual `out`. Finally, the computation function `FB` is modeled as a functional association (instance of class Association of Fig. 3) from `T_IN` to `T_OUT`. In domain model `Saturn_1` (Fig. 4) which refines `Saturn_0`, a new variable individual named `in_r` is defined to represent the acquired input data and another one, `out_r`, is defined to represent the computed result.

In domain model `Saturn_2` (Fig. 5) which refines `Saturn_2`, two concepts are defined: `MI` which represents the set of input agents and `MO` which represents the set of output agents. Concept `agents_in` (respectively `agents_out`) is a subconcept of `MI` (respectively `MO`) which represents the set of input (respectively output) agents that are active. Concept `VIN`, defined as the set of total functions from `agents_in` to `BOOL` ($VIN = agents\_in \longrightarrow BOOL$ where $BOOL = \{TRUE, FALSE\}$), represents the type of input data which are now arrays. Similarly, concept `VOUT` ($VOUT = agents\_out \longrightarrow BOOL$) represents the type of output data. Individuals `in`, `in_r`, `out` and `out_r` are refined respectively by individuals `s_in`, `s_in_r`, `s_out` and `s_out_r` using total injective associations `vec_to_in` from `VIN` to `T_IN` and `vec_to_out` from `VOUT` to `T_OUT`: $in = vec\_to\_in(s\_in), in\_r = vec\_to\_in(s\_in\_r), out = vec\_to\_out(s\_out), out\_r = vec\_to\_out(s\_out\_r)$. Finally, the computation function is modeled as a functional association named `VBF` from `VIN` to `VOUT`: $VBF = vec\_to\_in; FB; vec\_to\_out^{-1}$ (operator ; is the association composition operator used in logical formula assertions).

3.4 The B System Specification

3.4.1 B System Specification Constructed from Domain Models

Updates performed on the SysML/KAOS domain modeling language have resulted in adjustments on translation rules defined in (Fotso et al., 2018c). The adjusted rules are fully described in (Tueno et al., 2018). They have been formally verified with Event-B. The corresponding Event-B specification can be found in (Tueno et al., 2019). The rules are implemented within the *SysML/KAOS Domain Modeling tool* (Tueno et al., 2017b).

The specification below represents the *B System* specification obtained from the domain model of Fig. 2 (b).

SYSTEM Saturn_0
SETS b_T_IN b_T_OUT
CONSTANTS b_FB
PROPERTIES
axm1: $b\_FB \in b\_T\_IN \rightarrow b\_T\_OUT$
VARIABLES b_in b_out
INVARIANT
inv1: $b\_in \in b\_T\_IN$
inv2: $b\_out \in b\_T\_OUT$
Event *INITIALISATION* $\widehat{=}$ **then**
act1: $b\_in :: b\_T\_IN$
act2: $b\_out :: b\_T\_OUT$

Concepts `T_IN` and `T_OUT` give abstract sets `b_T_IN` and `b_T_OUT`. Variable individuals `in` and `out` give *B System* variables `b_in` and `b_out` typed (`inv1` and `inv2`) and initialised (`act1` and `act2`) respectively as items of `b_T_IN` and `b_T_OUT` (rule 11 of Table 1 of (Tueno et al., 2018)). Finally, association `FB` gives a *B System* constant named `b_FB`, typed (`axm1`) as a total function between `b_T_IN` and `b_T_OUT` (rule 10 of Table 1 of (Tueno et al., 2018)).

The *B System* specification obtained from the domain model of Fig. 4 defines a refinement

named Saturn_1 which refines Saturn_0 and introduces variables b_out_r and b_in_r with invariant $b\_out\_r \in b\_T\_OUT \wedge b\_in\_r \in b\_T\_IN$. The specification obtained from the domain model of Fig. 5 defines a *B System* refinement named Saturn_2 which refines Saturn_1. In Saturn_2, concepts MI and MO are defined as abstract sets. In addition, Saturn_2 defines agents_in, agents_out, VIN, VOUT, vec_to_in, vec_to_out and VBF as constants. Properties

axm1: $b\_agents\_in \subseteq b\_MI$
axm2: $b\_agents\_out \subseteq b\_MO$

define b_agents_in and b_agents_out as subsets of b_MI and b_MO (rule 5 of Table 1 of (Tueno et al., 2018)).

axm3: $b\_VIN = b\_agents\_in \rightarrow BOOL$
axm4: $b\_VOUT \in b\_agents\_out \rightarrow BOOL$
axm5: $b\_vec\_to\_in \in b\_VIN \rightarrowtail b\_T\_IN$
axm6: $b\_vec\_to\_out \in b\_VOUT \rightarrowtail b\_T\_OUT$
axm7: $b\_VBF \in b\_VIN \rightarrow b\_VOUT$
axm8: $b\_VBF = (b\_vec\_to\_in; b\_FB; b\_vec\_to\_out^{-1})$

Following rule 10 of Table 1 of (Tueno et al., 2018), property axm5 defines b_vec_to_in as a total injection from b_VIN to b_T_IN. Constant b_vec_to_out is typed in a similar manner by property axm6. Finally, the total function b_VBF (axm7) is defined by property axm8 as the composition of functions b_vec_to_in, b_FB and $b\_vec\_to\_out^{-1}$ (property axm8 results from the translation of a logical formula defined in domain model ***Saturn_2***). The *B System* refinement Saturn_2 also defines variables such as b_s_in and b_s_out, along with their invariants and initialisations.

3.4.2 B System Specification Constructed from Goal Model

The root level of the goal diagram of Fig. 1 gives the *B System* event *Process* specified as:

Event *Process* $\widehat{=}$ **then** $b\_out := b\_FB(b\_in)$ **END**

Furthermore, the first refinement level of the goal diagram gives the following *B System* specification:

Event *Get* ***ref_and*** *Process* $\widehat{=}$ **then** $b\_in\_r := b\_in$ **END**;
Event *Compute* ***ref_and*** *Process* $\widehat{=}$ **then** $b\_out\_r := b\_FB(b\_in\_r)$ **END**;
Event *Put* ***ref_and*** *Process* $\widehat{=}$ **then** $b\_out := b\_out\_r$ **END**

Each refinement level goal is translated into an event for which the body has been manually specified: event *Get* reads the input data, event *Compute* computes the output data and event *Put* outputs the result. The keyword ***ref_and*** is used to specify that concrete events *Get*, *Compute* and *Put* refine abstract event *Process*, in accordance with SysML/KAOS goal refinements, through the *AND* operator. This allows the automatic generation of proof obligations related to usage of the *AND* operator between abstract and concrete refinement levels (Matoussi et al., 2011) by the *Atelier B* tool:

(po1) $Get\_Guard \Rightarrow Process\_Guard$
(po2) $Compute\_Guard \Rightarrow Process\_Guard$
(po3) $Put\_Guard \Rightarrow Process\_Guard$
(po4) $(Get\_Post \wedge Compute\_Post \wedge Put\_Post) \Rightarrow Process\_Post$

For instance, the full specification of proof obligation (po4) is:
$(b\_in\_r = b\_in \wedge b\_out\_r = b\_FB(b\_in\_r) \wedge b\_out = b\_out\_r) \Rightarrow b\_out = b\_FB(b\_in)$.

It expresses that when the input data is read and the output data is computed, the output data is the result of applying b_FB to the input data.

The keyword ***ref_or*** is used when the *OR* operator appears between an abstract and a concrete refinement levels.

4 DISCUSSION

The specification of the Saturn protocol includes five refinement levels. It has been built, in a methodical and structured way, thanks to SysML/KAOS. Table 1 summarises the key characteristics related to proof obligations. Discharged using *Atelier B*, they allow the detection of omissions, ambiguities, redundancies and contradictions within requirements, while considering domain constraints. Furthermore, the domain modeling language has been extended in order to be more suitable for use in system modeling.

Table 1: Key characteristics related to the formal specification.

Refinement level	L0	L1	L2	L3	L4	*Summary*
Invariants	**2**	**6**	**8**	**1**	**1**	***18***
Events	**1**	**4**	**3**	**5**	**9**	***22***
Proof Obligations (PO)	**1**	**3**	**10**	**9**	**25**	***48***

5 CONCLUSION AND FUTURE WORK

This paper focusses on assessment of the SysML/KAOS method through the formal modeling of requirements related to *Saturn*, a rail communication protocol proposed by *ClearSy* (Technology, 2019). SysML/KAOS goal and domain modeling languages have been used to specify Saturn's requirements and application domain entities and properties. Translation rules, supported by tools (Matoussi et al., 2011; Tueno et al., 2017b), have then been used to obtain a *B System* specification. The SysML/KAOS method has

proven its usefulness for the specification of the protocol and has been enhanced, especially the domain modeling language, to make it more suitable for use in system modeling.

Work in progress is aimed at (i) integrating the updates within the open-source platform *Openflexo* (Openflexo, 2019), which federates the various contributions of *FORMOSE* project partners (ANR-14-CE28-0009, 2017), (ii) at studying the impact of proposed revisions on the back propagation rules (Fotso et al., 2018a), and (iii) at assessing the updated SysML/KAOS method on further case studies.

REFERENCES

Abrial, J. (2010). *Modeling in Event-B - System and Software Engineering*. Cambridge University Press.

Abrial, J.-R. (2005). *The B-book: assigning programs to meanings*. Cambridge University Press.

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

ClearSy (2014). Atelier B: B System.

Fotso, S. J. T., Frappier, M., Laleau, R., and Mammar, A. (2018a). Back propagating B system updates on SysML/KAOS domain models. In *ICECCS*, pages 160–169. IEEE.

Fotso, S. J. T., Frappier, M., Laleau, R., and Mammar, A. (2018b). Modeling the hybrid ERTMS/ETCS level 3 standard using a formal requirements engineering approach. In *ABZ*, volume 10817 of *LNCS*, pages 262–276. Springer.

Fotso, S. J. T., Mammar, A., Laleau, R., and Frappier, M. (2018c). Event-B expression and verification of translation rules between SysML/KAOS domain models and B System specifications. In *ABZ*, volume 10817 of *LNCS*, pages 55–70. Springer.

Hause, M. et al. (2006). The SysML modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9. Citeseer.

Jetbrains (2017). Jetbrains mps.

Laleau, R., Semmak, F., Matoussi, A., Petit, D., Hammad, A., and Tatibouet, B. (2010). A first attempt to combine SysML requirements diagrams and B. *Innovations in Systems and Software Engineering*, 6(1-2):47–54.

Lecomte, T., Déharbe, D., Prun, É., and Mottin, E. (2017). Applying a formal method in industry: A 25-year trajectory. In *SBMF*, volume 10623 of *LNCS*, pages 70–87. Springer.

Lee, D. G. and Suh, N. P. (2005). Axiomatic design and fabrication of composite structures-applications in robots, machine tools, and automobiles. *Oxford University Press*, page 732.

Matoussi, A., Gervais, F., and Laleau, R. (2011). A goal-based approach to guide the design of an abstract Event-B specification. In *ICECCS*, pages 139–148.

Openflexo (2019). Openflexo project.

Pierra, G. (2004). The PLIB ontology-based approach to data integration. In *IFIP 18th World Computer Congress*, volume 156 of *IFIP*, pages 13–18. Kluwer/Springer.

Roques, A. (2015). Plantuml: Open-source tool that uses simple textual descriptions to draw uml diagrams.

Sengupta, K. and Hitzler, P. (2014). Web ontology language (OWL). In *Encyclopedia of Social Network Analysis and Mining*, pages 2374–2378.

Technology, R. (2019). SATURN: SIL2-Certified Fail-safe Remote I/O System Architecture for Trains, https://www.railway-technology.com/contractors/computer/leroy-automation/pressreleases/presssaturn-certified-fail-safe/ .

Tueno, S., Frappier, M., Laleau, R., Mammar, A., and Barradas, H. R. (2018). The Generic SysML/KAOS Domain Metamodel. *ArXiv e-prints, cs.SE, 1811.04732*.

Tueno, S., Laleau, R., Frappier, M., and Mammar, A. (2019). Event-B specification of the updated translation rules from ontology-based domain models to B System specifications, https://github.com/anonym21/SysML_KAOS_Domain_Model_Parser/blob/master/SysMLKAOSNewDomainModelRulesFormalisation_20190301.zip.

Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017a). Formal Representation of SysML/KAOS Domain Models . *ArXiv e-prints, cs.SE, 1712.07406*.

Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017b). SysML/KAOS Domain Modeling Tool, https://github.com/anonym21/SysML_KAOS_Domain_Model_Parser.

Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017c). The SysML/KAOS Domain Modeling Approach. *ArXiv e-prints, cs.SE, 1710.00903*.

Tueno, S., Laleau, R., Mammar, A., and Frappier, M. (2017d). Towards using ontologies for domain modeling within the SysML/KAOS approach. In *RE Workshops*, pages 1–5. IEEE Computer Society.

van Lamsweerde, A. (2009). *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley.