Aligning SysML with the B Method to Provide V&V for Systems Engineering

Erwan Bousse, David Mentré, Benoit Combemale, Benoit Baudry, Katsuragi Takaya

Aligning SysML with the B Method to Provide V&V for Systems Engineering

Erwan Bousse
Mitsubishi Electric R&D
Rennes, France
erwan.bousse@gmail.com

David Mentré
Mitsubishi Electric R&D
Rennes, France
d.mentre@fr.merce.mee.com

Benoît Combemale
University of Rennes 1, IRISA
Rennes, France
bcombema@irisa.fr

Benoît Baudry
Inria
Rennes, France
benoit.baudry@inria.fr

Takaya Katsuragi
Advanced Technology R&D
Center, Mitsubishi Electric
Amagasaki, Japan
katsuragi.takaya@ap.mitsubishielectric.co.jp

ABSTRACT

Systems engineering, and especially the modeling of safety critical systems, needs proper means for early Validation and Verification (V&V) to detect critical issues as soon as possible. The objective of our work is to identify a verifiable subset of SysML that is usable by system engineers, while still amenable to automatic transformation towards formal verification tools. As we are interested in proving safety properties expressed using invariants on states, we consider the B method for this purpose. Our approach consists in an alignment of SysML concepts with an identified subset of the B method, using semantic similarities between both languages. We define a restricted SysML extended by a lightweight profile and a transformation towards the B method for V&V purposes. The obtained process is applied to a simplified concrete case study from the railway industry: a SysML model is designed with safety properties, then automatically transformed into B, and finally imported into Atelier-B for automated proof of the properties.

1. INTRODUCTION

Systems engineering is a field that involves many different stakeholders, each of them with a different role in the development and with a specific set of skills. This implies handling different views of the system, each of them adapted to the specialist it is dedicated to. For that matter, Domain Specific Modeling Languages (DSML) offer a satisfying solution: designing languages whose syntax and purposes exactly fit stakeholder roles, while being adapted to their domains. For instance, the Systems Modeling Language (SysML) is a graphical language adapted to systems engineering that can be used throughout a system development. A particular kind of systems are *safety critical systems*, for which failures can lead to casualties or the loss of equipment. Validation and Verification (V&V) of safety critical systems is required, as safety properties and compliance with standards such as EN 50128 (railway applications), IEC 61508 (electronic safety related systems), or DO 178C (software-based aerospace systems) must be ensured. This must be done as early as possible during the design phase to limit rollbacks and underlying costs. In this regard, formal methods can be used to check specified properties of the system in all possible configurations. However, existing approaches require the use of formal notations that can be difficult to read and understand, especially for experts of a particular field. A way to provide V&V for existing DSMLs is to provide a translation towards formal notations, so that existing theorem provers or model checkers can be used.

Our work focuses on what we call the *alignment* of a DSML with a formal language. The idea is to identify good constructs and good practices of the formal language, then to search semantic similarities between such constructs and the considered DSML, and finally to define a transformation from the DSML to the formal language using Model Driven Engineering (MDE) techniques. We apply these principles to SysML – using the Object Constraint Language (OCL) for safety properties and the Action Language for Foundational UML[1] (Alf) for behaviors – and the B method. The resulting process relies on SysML block definition and state machine diagrams on the one hand, and on B developed modules linked by *imports* and *sees* links on the other hand. This process leads to concrete proved software using the Atelier B tool. To complete this work, we implemented the first version of a prototype in Kermeta that realizes this transformation, and we provide a concrete application of this tool-supported process to a simplified example of railway specification provided by Mitsubishi Electric.

Remaining sections are organized as follows. Section 2 lays down the industrial context of this work at Mitsubishi Electric. Section 3 introduces languages we consider in our approach – namely SysML and the B method. Section 4 presents our contribution, which consists in an alignment of SysML with the B method. Sections 5 and 6 conclude with related work and final words.

[1]Unified Modeling Language

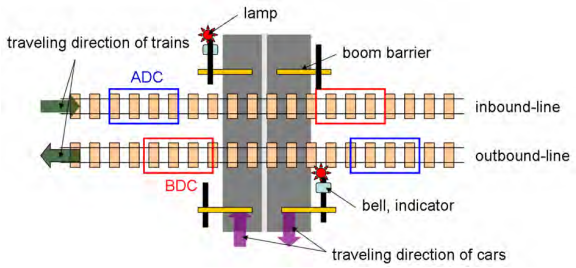


Figure 1: Case study: a railway crossing

2. INDUSTRIAL CONTEXT

In this second section, we present the context of this work at Mitsubishi Electric and the underlying objectives.

2.1 Goals and technical choices

The scope of this work at Mitsubishi Electric is the design of safety-critical railway systems; in particular, we have the following goals. (1) Modeling languages must be adapted to systems engineering and its practitioners, which may not be formal methods specialists. (2) We want to be able to design systems at least at the implementation level, i.e. models translatable into concrete deployable embedded software. (3) It is necessary to manage good traceability between informal requirements and the modeled system, especially for safety properties. (4) Ultimately, we must be able to prove such properties using formal methods with tool support, keeping in mind scalability issues. (5) Finally, we need to consider at least safety properties expressed as invariants on states of the system.

The SysML language [19] is a rising modeling language for systems with dedicated traceability specific features: a requirements diagram and links between requirements and elements of the model. Additionally, this language is complete enough to provide ways to design systems at any level of abstraction, including the implementation level. Following goals 1, 2 and 3, we choose to use SysML for the modeling part of our process.

As the B method [1] has been used in important safety-critical system realizations in Europe, such as the Paris Métro Line 14 [4], we looked at this formal method with interest. The B method relies on theorem proving, which means that scalability is rather well handled, unlike state explosion problems that may appear with model checking techniques [9, 7]. In terms of granularity, this method goes down to the implementation of the system. B notations and concepts can be difficult to apprehend, but this is not a problem since we only want to use this formal method for V&V purposes. Overall, it meets satisfactorily goals 2, 4 and 5

2.2 Case study: a railway crossing controller

For our work, we use a proprietary case study provided by Mitsubishi Electric. It is a technical specification and requirements document describing a railway crossing controller. Figure 1 shows the configuration of the crossing: there are two lines (inbound and outbound), both with a critical section between two train sensors, and cars may pass on the crossing road. Each track has two sensors: one called ADC that detects trains arriving in the warning section, another called BDC for trains leaving the warning section. The main requirement is to detect when trains are in the critical sections in order to activate the barriers, the bells, and the lamps accordingly. Being a life-critical system, properties such as "when there is a train in a critical section, lamps are lit" or "barriers are closed or closing when there are trains in the critical sections" have to be proved.

3. BACKGROUND

3.1 SysML, OCL and Alf

The Systems Modeling Language (SysML) [19] is a graphical modeling language managed by the Object Management Group (OMG). It is oriented towards general purpose systems engineering applications. It is well suited for systems mixing hardware and software with constraints of many kinds, as in rail systems. Over the 9 SysML diagrams, we only use in this paper the block definition diagram (structure modeling) and the state machine diagram (reactive behavior modeling), a *block* being the basic building block of the language to define system elements. It is also possible to design extensions for SysML using its profile mechanism.

Additionally, we use the possibility provided by the SysML language to enrich models using other languages. The Object Constraint Language (OCL) is a specification language that allows to enrich models with side-effect-free annotation; we use it to write invariant constraints in our SysML models. The Action Language for Foundational UML (Alf) [18] is an action language with a Java-like syntax for describing behaviors, such as values assignments or operation calls; we use it to define low level behaviors in our SysML models.

3.2 The B Method

The B method [1] is a field-proven formal method that was successful in various safety-critical rail system applications like Paris Métro Line 14 or Roissy VAL [4, 3, 2]. Tools are available for operational use like Atelier B [8], a proprietary integrated development environment developed by ClearSy. For our work, we consider Atelier B as a part of the suggested process: we generate B code with Atelier B specific syntax.

A B *project* is a collection of modules that models a system. A *module* is a part of the system, and consists in a stack of *components* which can be abstract machines, refinements or implementations. An *abstract machine* is the visible specification of a module. A *refinement* refines either an abstract machine or another refinement, and there can be several refinement steps. An *implementation* is the final refinement step of a module. Most of the B language is a specification language, which includes set theory, first order logic, indeterminism and abstract data. This specification language is used to define abstract machines and refinements. Another part of the language, called B0, is a formally defined programming language used in implementations that can be compiled to C or Ada. Each module is made of at least one abstract machine, zero or more refinements, and zero or one implementation. A module only made of an abstract machine is said to be *abstract*, while a module with an implementation is *developed*. It is possible to *instantiate* modules, which allows the use of instances of lower level modules within the implementation of a module.

This is done by using the *imports* link. On the other hand, the *includes* link is a way to decompose an abstract machine into a set of abstract machines, but without instantiations: this is equivalent to splitting the machine into multiple files instead of a single one. Finally, the *sees* link is a read-only access from a module to another; it is required when data has to be shared between modules, e.g. constants definition.

In this paper, even though B has abstraction capabilities, we consider B models that describe implementations of systems. Specifications of such systems are translated in two kind of B elements: invariant properties to be proved on the implementations, and B abstractions corresponding to assumptions on the environment.

4. ALIGNING SYSML WITH B

In this section we present our main contribution: based on a B subset we first delimit, we identify a subset of SysML aligned with this subset of B, and we define of a mapping between the two languages we eventually obtain.

4.1 Industrial use of the B method

There always are multiple ways to model a system with a specific language; however, all languages have what we can call "good practices", which are rules to apply in order to obtain "good" models or programs. For instance, in object-oriented programming, design patterns give solutions considered effective for some precise problems, like *strategy* for algorithm modularity. In the case of formal languages, another factor than effectiveness comes into account: the ability to prove required properties of the model. The more constrained, simple and small a model is, the easier it is to analyze it. Applying these criteria can limit the state-space size of the model or give clearer relationships between model elements. Formal models must also be well decomposed, so that properties that are specific to a part of a system are easier to prove and do not depend on unrelated parts.

Regarding the B method, there mainly are two aspects to consider: what kinds of B components and what kinds of links between these components to use. We deduct from past industrial uses [3] of the B method that the most important is to have well decomposed B projects: for scalability concerns, a property to prove must be as much as possible contained into a subpart of the model. The *imports* link is a way to do such a modular design, by defining a module using lower level instances of other modules. This implies that we work with developed modules instead of purely abstract modules. Refinement components are rarely used[2]: each abstract machine is refined only once into an implementation. Additionally, *sees* links are required when data has to be shared between modules, like global constants. To sum up, we restrict our usage of the B method to developed modules with one abstract machine and one implementation linked by *imports* and *sees* links. Concerning data, we only use primitive types (boolean, integer and enumerations) in this first approach.

4.2 Semantic similarities

[2]Refinements are only used when an algorithm or a data structure are too complex to be proved directly at implementation level.

Table 2: List of semantic similarities we consider

B concept	SysML concept
Project	Set of blocks
Module	Block
Imports link	Part property
Sees link	Reference property
Basic type	Value type
Basic data	Value properties
Constants	`isReadOnly` meta-attribute
Enumerated set	Enumeration
Component parameters	Default values
Initialization and valuing	Default values
Operation	Operation (with behavior)
Invariant	Constraint property

After studying and restricting the B method, the idea is now to align SysML with this delimited subset. We call *semantic similarities* between languages features of both languages that are close semantically. In a nutshell, we want to be able to construct models in SysML that can be translated in this subset of the B method while preserving semantics.

4.2.1 Parallel read of the languages specifications

Searching for semantic similarities consists in reading official specifications of both languages and highlighting potentially related parts. It is important to note that the role of each concept is defined using natural language, which means that the parallels depend on our reading and understanding. This must not be confused with the fact that concepts may be *defined* using formalisms, like the B language with the B-Book [1], while we look at the purpose of the concepts.

To illustrate our approach, here is how we proceed for the first three similarities. Table 1 shows quotes of SysML and B specifications, each line putting side by side similar concepts of SysML and the B method. We find the following parallels. A B *project* has the same purpose as a set of SysML *blocks*, which is the definition of a system. Such a *block* is very similar to a B *module* to model an element of the system. Finally, an *imports* link defines a subsystem of a module the same way a SysML *part property* does for a block.

Altogether, we identify 12 semantic similarities between SysML and the B method for this first approach – see Table 2 for the complete list. The complete analysis is available in [5]. Using these similarities, we define a subset of SysML which allows only a few constructs among the many possibilities of SysML. In short, each model is a collection of blocks with primitive data in value properties, linked by *part property* links for the system decomposition, with textual operations to define their behavior and required constraints over variables. Such models can be entirely designed with block definition diagrams.

Additionally, we use *opaque behavior* and *opaque expression* features of SysML to enrich models with textual behaviors and expressions written in Alf and OCL respectively. This part of the translation relies on straightforward parallels between B operators or instructions and Alf, and between B predicates and OCL. We do not present these semantic similarities in this paper (refer to [5] for more details).

Table 1: Semantic similarities between B method projects and SysML blocks to model structure

Quote from ClearSy B Language Manual [8]	Quote from OMG SysML specification [19]
"*A complete development in B corresponds to a* ***B project****. A project enables formally a system of any type. [. . .] A B project refers to a complete set of B module instances. The components of these module instances are connected by links.*"	"***Blocks*** *provide a general-purpose capability to model systems as trees of modular components. The specific kinds of components, the kinds of connections between them, and the way these elements combine to define the total system can all be selected according to the goals of a particular system model*"
"*A* ***B module*** *models a sub-system; it forms a part of a B project.*"	"*A* ***Block*** *is a modular unit that describes the structure of a system or element.* "
"***Import*** *is used to structure a B project into layers, since the implementation of a module is implemented by importing other modules.*"	"*SysML blocks [. . .] provide the ability to represent a system hierarchy, in which a system at one level is composed of systems at a more basic level. [. . .] A* ***part property*** *holds instances that belong to a larger whole.*"

4.2.2 Some additions to our subset of SysML

Even with a subset of the B method and a subset of SysML that are semantically similar, we are not completely satisfied yet for two reasons. First, we did not find SysML counterparts for some essential concepts of the B method: the notion of *main module* of a project, being able to define abstract elements, and how to consider invariants defined in the implementation of a module. Second, we want to be able to design reactive systems in a practical way.

To overcome the first reason, we use UML profile mechanism to define an very simple extension to the SysML language. Firstly, a `main` stereotype to tag the main block of the SysML model. Secondly, an `abstract` stereotype to tag value properties that are abstract and should not appear in the final implementation – this is necessary to model hypotheses on the environment. Thirdly, a `gluing` stereotype to tag constraints that both affect properties of a block and properties of its parts – this corresponds to B invariants expressed in the implementations of modules.

For the second reason, we rely on the SysML state machine feature to define reactive behaviors of blocks. Since there is no such concept within the B method, this part does not rely on any semantic similarities. However, preserving semantics in a translation is nonetheless possible, as shown by Sekerenski [20] who defined rules to transform UML state machines into B abstract machines. Using and adapting this work, we add a subpart of SysML state machine concepts to our restricted SysML – putting aside most pseudostates and `entry/do/exit` behaviors amongst other things.

We defined our customized SysML with14 rules (presented in [5]) written in natural language that restrict SysML usage. Further work may lead us to write them using the OCL language over the SysML metamodel.

4.2.3 Modeling the case study

To experiment with the restricted SysML we eventually define, we model a simplified version of the case study introduced Section 2.2. We consider that only two lamps have to be lit when trains are in the warning section between the sensors and that a failure lamp must be activated when a failure state is reached.

Figure 2 is a block definition diagram representing all blocks of the system and their links. The main block is the Controller, whose properties are a counter `train_counter` for trains in the warning section and a constant `MAX_TRAINS` for the maximum number of trains that can exist in the warning section at a given time. Two operations are defined: `ADC()` called when a train arrives in a warning section and `BDC()` when a train leaves one. The Lamp block models a physical lamp: it has a boolean `is_on` set to true (respectively false) when a `turn_on()` (respectively `turn_off()`) operation is called. This variable is *abstract* because we do not need at the implementation level to keep in memory the state of the lamp: it is an hypothesis we consider on the behavior of a real lamp. Operations `turn_on/turn_off` concretely act on these lamps using a hardware API, which is not shown here. The Controller block possesses three part properties `lamp1`, `lamp2` and `failure_lamp`, which are instances of the Lamp block. Initial conditions are the following: no trains on the tracks, lamps are not lit. Two safety properties are defined within the Controller block using *gluing* constraints: the first one guarantees that lamps are lit when the train counter is strictly positive; the second one ensures that the failure lamp is lit when the train counter becomes inconsistent.

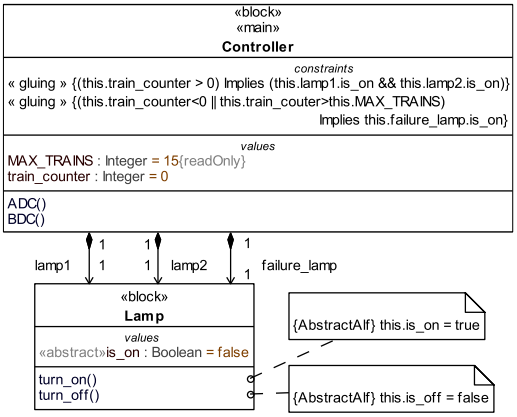


Figure 2: Block definition diagram of the case study SysML model

A state machine diagram representing the reactive behavior of the Controller block is also part of this SysML model, but due to the page limit, we chose in this paper to focus on the part of our SysML subset based on semantic similarities. Refer to [5] for the complete case study.

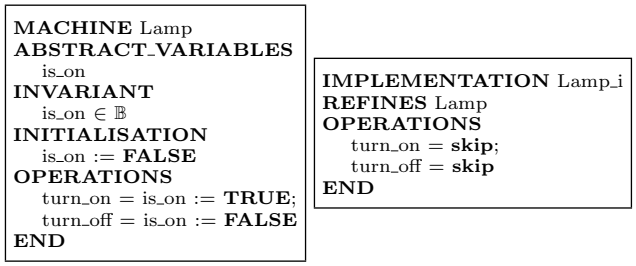


Figure 3: B module obtained from the Lamp block of the SysML model of Figure 2.

4.3 Translation of SysML into B

Now that we have a SysML subset corresponding to the targeted B subset, the last step of our work is the definition of a translation toward the B method. Semantic similarities we identified between both languages allowed us to delimit the SysML we use, but also directly gives us a mapping between SysML and B concepts, which directly leads to transformation rules. Figures 3 and 4 are parts of a B model that results from the application of such rules of the SysML model introduced Section 4.2.3.

Each block of the SysML model is translated into a B module, which is made of an abstract machine with the same name as the block (ex. Lamp) and an implementation with a _i suffix (ex. Lamp_i). Part properties of the Controller block (ex. `lamp1`) are translated into *imports* links with the same names. Regular value properties (ex. `train_counter`) become B concrete variables, whereas those with their `isReadOnly` attribute set (ex. `MAX_TRAINS`) become B concrete constants. Abstract elements of the SysML model (`is_on` property, behaviors of the `turn_on` and `turn_off` operations) are translated into purely abstract B data and behaviors. Constraints of the SysML model are translated into their B predicate counterpart as a clause of a B *invariant* of the corresponding module. If the constraint is tagged with the *gluing* stereotype, we put it in the invariant of the implementation – this is necessary since B *imports* are only visible from there. Note that we do not present the part of the transformation that concerns state machines, as we chose to focus on SysML blocks in this paper. When importing the whole model in Atelier B, 49 Proof Obligations (PO) are generated[3]. They are all solved automatically.

Our translation is defined by 19 rules established using minimal examples (presented in [5]). Using Model Driven Engineering (MDE) techniques, we implemented a proprietary model transformation written in Kermeta [17]. The tool takes as input a SysML model encoded using the XML Metadata Interchange (XMI) format, and applies a model-to-model transformation defined between the SysML metamodel and the B metamodel[4]. Then, a model-to-text transformation is applied to the B model encoded in XMI to obtain a usable Atelier B project. More information about the tool can be found in a dedicated web page [6].

[3]PO distribution: 2 for initial values, 12 for underflows/overflows of the counter, and 35 for safety invariants.
[4]We wanted to use the B metamodel of Idani [10], but as it focuses on dependencies between B constructs, we had to design one that considers the complete B abstract syntax.

5. RELATED WORK

Snook and Butler [21] worked on a translation from UML towards the B method. While their motivation is similar to ours – extending UML to provide an easier way to use the B method– their solution is different. They transform a whole UML model into a single B module, while we prefer to have several machines within a well decomposed B project. To enrich their UML models, they designed a language called μB with a B-like syntax, while we chose to use existing OCL and Alf languages. Finally, they designed a UML profile to tag UML elements with many B concepts, while the profile we defined for SysML is as non-intrusive and lightweight as possible. We believe that our choices lead to a process requiring less knowledge of the B method, while taking into account how the B method is industrially used.

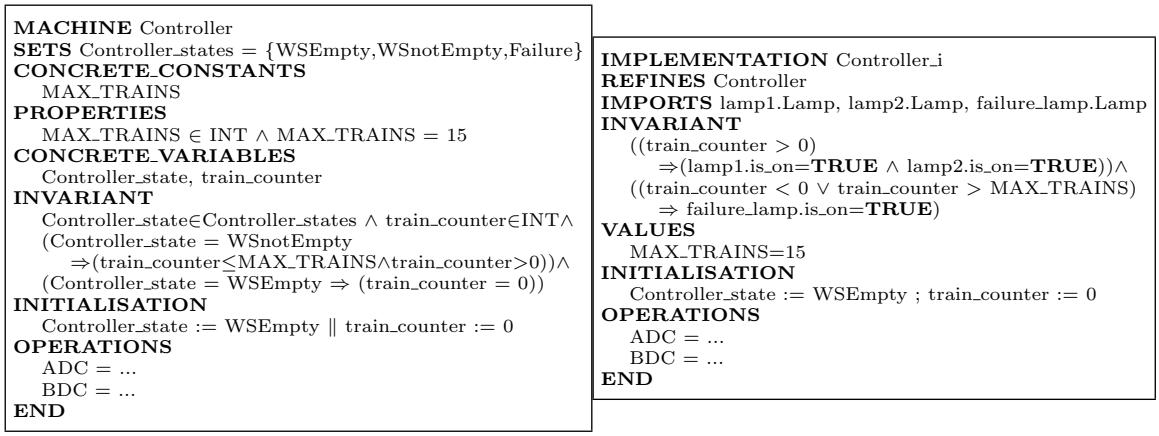
Many other works offered ways to transform UML models into B. Lano, Clark and Androutsopoulos [13] worked on the translation of UML-RSDS models into B. Laleau and Polack [11] suggested a transformation process oriented towards Information Systems (IS) modeled in UML, using an extension of UML called IS UML. Meyer and Souquières [16] originally worked on the translation of Object Modeling Technique (OMT) models. However, all these approaches only worked with B abstract modules and *includes* links, while we focused and developed modules and *imports* links.

We already mentioned the work of Sekerinski on state machines [20]. He is one of the few who went through most possibilities offered by state machines: composite states, orthogonality, spontaneous transitions, etc. However, he does not consider structure modeling with blocks or classes, and only provide a way to translate a state machine diagram into a single B abstract machine.

To our knowledge, the work of Laleau, Semmak, Matoussi and Gnaho [12, 15, 14] is the only other attempt to combine SysML and the B method in a same process. Their work focuses on the SysML requirement diagram and is based on the following observation: relationships between SysML requirements are quite limited compared to more elaborated goal-based approaches. Their solution consists in adding KAOS (Keep All Objectives Satisfied) concepts inside SysML by defining a SysML profile. Then, they define a transformation from this extended SysML towards Event-B – a recent event based alternative to the B method – which turns requirements into B events. While this work may lead to an interesting goal-based approach for SysML, it only focuses on SysML requirements and is quite far from our own objective of modeling complete functional systems using blocks and state machines.

6. CONCLUSION

In this paper, we looked at how to provide early V&V for the SysML language using the existing tool-supported B method. Our goals were to consider models designed at the implementation level and to use this formal method in a scalable way. Our contribution relies in the alignment of SysML with the B method in three steps. The first one is the definition of a subset of the B method that focuses on good decomposition, using *imports* and *sees* links between B modules. The second step is a search for semantic similarities between SysML and the previously identified subset



```
MACHINE Controller
SETS Controller_states = {WSEmpty,WSnotEmpty,Failure}
CONCRETE_CONSTANTS
   MAX_TRAINS
PROPERTIES
   MAX_TRAINS ∈ INT ∧ MAX_TRAINS = 15
CONCRETE_VARIABLES
   Controller_state, train_counter
INVARIANT
   Controller_state∈Controller_states ∧ train_counter∈INT∧
   (Controller_state = WSnotEmpty
      ⇒(train_counter≤MAX_TRAINS∧train_counter>0))∧
   (Controller_state = WSEmpty ⇒ (train_counter = 0))
INITIALISATION
   Controller_state := WSEmpty || train_counter := 0
OPERATIONS
   ADC = ...
   BDC = ...
END
```

```
IMPLEMENTATION Controller_i
REFINES Controller
IMPORTS lamp1.Lamp, lamp2.Lamp, failure_lamp.Lamp
INVARIANT
   ((train_counter > 0)
      ⇒(lamp1.is_on=TRUE ∧ lamp2.is_on=TRUE))∧
   ((train_counter < 0 ∨ train_counter > MAX_TRAINS)
      ⇒ failure_lamp.is_on=TRUE)
VALUES
   MAX_TRAINS=15
INITIALISATION
   Controller_state := WSEmpty ; train_counter := 0
OPERATIONS
   ADC = ...
   BDC = ...
END
```

Figure 4: B module obtained from the Controller block of the SysML model of Figure 2.

of the B method, in order to define a subset of the SysML language semantically close to the subset of B. Our retained subset relies on block definition and state machine diagrams. We had to define a SysML profile in order to fill a gap with some B concepts. The third step relies on found similarities to define a transformation with a semantic gap as little as possible. We implemented it using the Kermeta language. We applied this SysML to B transformation on a concrete example from the rail industry, and managed to prove all safety properties.

This work is a first experiment limited to simple data structures and without making real use of the B method refinement capabilities. Aligning both languages aims to preserve semantics during the transformation, but such preservation is not proved thus far. Moreover, this work lacks effective support of traceability using SysML mechanisms and errors identified by the B method cannot be easily linked to the original SysML model. Further work will extend this V&V approach to other DSMLs and formal methods, with a particular focus on bidirectional transformation for traceability and reflecting identified errors on the original model.

7. REFERENCES

[1] J.-R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, 1996.

[2] J.-R. Abrial. Formal methods in industry: achievements, problems, future. In *SE'2006*, 2006.

[3] F. Badeau and A. Amelot. Using B as a high level programming language in an industrial project: roissy VAL. In *ZB'05, LNCS 3455*. Springer, 2005.

[4] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. Météor: A successful application of B in a large project. In *FM'99, LNCS 1708*. Springer, 1999.

[5] E. Bousse. Requirements management led by formal verification. Master's thesis, INSA Rennes, 2012. `http://people.irisa.fr/Benoit.Combemale/research/2012/bousse_erwan_report.pdf`.

[6] E. Bousse. SysML to B translator. `http://www.irisa.fr/triskell/Software/protos/sysml2b/index_html?set_language=en`, 2012. web.

[7] E. M. Clarke. My 27-year quest to overcome the state explosion problem. In *LICS '09*. IEEE CS, 2009.

[8] ClearSy. B Language Reference Manual - Version 1.8.7. Technical report, 2010.

[9] S. Demri, F. Laroussinie, and P. Schnoebelen. A parametric analysis of the state-explosion problem in model checking. *J. Comput. Syst. Sci.*, 2006.

[10] A. Idani and Y. Ledru. Dynamic graphical UML views from formal B specifications. *Inf. Softw. Technol.*, 48:154–169, March 2006.

[11] R. Laleau and F. Polack. Coming and going from UML to B : a proposal to support traceability in rigorous IS development. In *ZB 2002, LNCS 2272*. Springer, 2002.

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

[13] K. Lano, D. Clark, and K. Androutsopoulos. Uml to b: Formal verification of object-oriented models. In *IFM'2004, LNCS 2999*. Springer, 2004.

[14] A. Matoussi, F. Gervais, and R. Laleau. An Event-B formalization of KAOS goal refinement patterns. Technical Report TR-LACL-2010-1, LACL, University of Paris-Est (Paris 12), 2010.

[15] A. Matoussi, F. Gervais, and R. Laleau. A goal-based approach to guide the design of an abstract Event-B specification. In *ICECCS*, 2011.

[16] E. Meyer and J. Souquières. A Systematic Approach to Transform OMT Diagrams to a B Specification. In *FM '99*. Springer, 1999.

[17] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. Weaving executability into object-oriented meta-languages. In *Proceedings of MODELS/UML'2005*, Oct. 2005.

[18] OMG. Action Language for Foundational UML (Alf), Concrete Syntax for a UML Action Language, FTF – Beta 1. Technical report, 2010.

[19] OMG. OMG Systems Modeling Language (OMG SysML), V1.3. Technical report, 2012.

[20] E. Sekerinski. Graphical design of reactive systems. In *B'98, LNCS 1393*. Springer, 1998.

[21] C. Snook and M. Butler. UML-B: Formal modelling and design aided by UML. *ACM Trans. on Software Engineering and Methodology*, 15:92–122, 2006.