A generic process and its tool support towards combining UML and B for safety critical systems*

Akram Idani
INRETS†
ESTAS-Team
Villeneuve d'Ascq

Jean-Louis Boulanger
HEUDIASYC‡
University of Technology
Compiègne

Laurent Philippe
HEUDIASYC
University of Technology
Compiègne

Abstract

The complex requirements of software systems justify the use of the best existing techniques to guarantee the quality of specifications and to preserve this quality during the programming phase of a software life-cycle. On the one hand, visual specification languages (such as UML) have been widely used for specifying, visualizing, understanding and documenting software systems, but they suffer from the lack of precise semantical basis. Hence, these languages are practical for large-scale information systems rather than for industrial applications which address safety challenges. On the other hand, formal methods (such as B) are specifically used for safety critical systems in order to rigorously check their correctness but they lead to complex models which may be difficult to read and understand. These complementarities between formal and graphical techniques motivate our aim to develop a tool for combining both methods starting from UML and producing B models. Existing works which tried to integrate UML and B proposed model-to-model transformation techniques which miss a clear and explicit conceptual foundations. In order to circumvent this shortcoming we propose a metamodel-based transformation technique which is founded by a set of structural and semantic mappings between UML and B. A reusable and evolutive framework is then developed to assist derivation of formal B specifications from UML diagrams.

Keywords: Integrated methods, B, UML, Safety, Meta-modeling, Model Driven Architecture (MDA).

*This work is supported by the French regional project RT3-TUCS dedicated to UML technology for complex systems certification and the French national project ANR-ACI-SAFECODE about safety components design.

†The French National Institute for Transport and Safety Research. ESTAS Team: 20 rue Elisée Reclus, BP 317 F-59666 Villeneuve d'Ascq, France. E-mail: Akram.Idani@inrets.fr

‡HEUDIASYC laboratory, UMR 6599, University of Technology of Compiègne. Centre de recherches de Royallieu. Software Engineering Department. B.P. 20529, 60205 Compiègne, France. E-mail: boulange@hds.utc.fr

1 Introduction

Linking formal and semi-formal specification techniques has been a challenge since several years. This is justified by the complementary aspects and the cross contributions of these two techniques. On the one hand, the formal specifications are expressed in languages with quite precise syntax and semantics. Indeed, they are built on a solid theoretical basis allowing automated validations, which is important to develop reliable systems. On the other hand, semi-formal specifications are defined around textual or graphic languages which are useful to develop structured systems, but they are defined on a rather weak semantics [16, 6]. Having a method which guarantees the development of structured as well as reliable systems is one of the primary objectives of software engineering. This explains the actual evolutions of the formal and semi-formal methods: the evolution of formal methods tends to integrate the modularity principle in order to provide structuring and clearness. We notice, for example, the concept of abstract machine in B, schema in Z, and module in VDM. Still, the evolution of semi-formal methods is often done by integrating mathematical notations to express some characteristics (or constraints) of the system (OCL [20] for example). This combination of graphical and formal notations brings more precise semantics to graphical formalisms, and favour a wider use of formal method tools.

Our aim is to develop an object oriented modeling approach (based on UML) for safety critical systems, especially for railway applications [15]. Thus, the use of formal method tools is primordial to ensure precision and consistency of various UML models. In this paper we use B [2] in order to provide a formal meaning to UML diagrams. The resulting formal model can then be enriched, refined, etc, and the use of formal methods tools, such as provers and model-checkers, becomes possible. Furthermore, this formal development process is known to lead to high qual-

ity developments. For example, in [4], P. Behm and his co-authors report that the Meteor subway development reached the "zero-fault" objective. Recently, B is successfully used as a high-level programming language on the VAL shuttle for Roissy Charles de Gaulle airport [3].

Existing works [9, 5] which tried to integrate UML and B have had the same motivations. Moreover, they proposed model-to-model transformation techniques which miss a clear and explicit conceptual foundations. In this paper we propose a MDA-based transformation technique which is founded by a set of structural and semantic mappings between UML and B meta-models. A reusable and evolutive framework is then developed to assist derivation of formal B specifications from UML diagrams.

Section 2 presents our methodological orientations to use UML for safety critical systems. Section 3 addresses why and how we apply the MDA approach on UML to B transformations. In section 4, we give a brief overview of our MDA-based UML-to-B tool. Finally, section 5 draws the conclusion and the perspectives of our work.

2 Safe UML: our approach

It is well known that a failure in safety-critical systems such as nuclear, railway, aeronautics or automobile may endanger human life or the environment. This is especially true when more and more electronics are embedded in these systems. For this reason, many standards were conceived to normalize the systems design, implementation and exploitation. Examples of these standards are IEC 880 in nuclear system, CENELEC EN50126, EN50128 and EN50129 in railway system in Europe, and DO-178 and DO-254 in aeronautic system. Above all these standards is the general IEC 61508 [1] which covers all systems based on electric, electronics and programmable electronics.

These standards give a lot of methodological issues and advocate for two major trends: (*i*) the use of automated formal reasonings to increase confidence in such systems, and (*ii*) the use of comprehensible and accessible notations (such as UML) to reduce certification effort for the design phase. So, for applications which need to fulfill these safety standards, it seems inescapable to follow a development process based on both formal and graphical notations (*e.g.* UML and B).

Fig. 1 shows how our activities are integrated within a classical development process based on UML. The left hand side of the figure shows our previous work [10, 15, 14] which is intended to determine a well-defined UML core accompanied by a set of safety commandments (inspired by the cited standards). The interest of this first step is to guide stakeholders when using UML in a safety critical software development process with certification settings.

On the right hand side of the figure, the current step of our work, which is investigated in this paper, builds formal models from the various UML documents produced during the previous step. This second step improves the state of the art by:

- exploiting the projection of the safety guidelines on the translation rules.
- defining explicitly mappings between UML and B in terms of their meta-models.

As a result, certifiable code can be produced by refinements and assisted by proofs. Indeed, the B formal model produced from the said safe UML core is subject to automated formal reasonings in order to check its correctness and to generate source code.

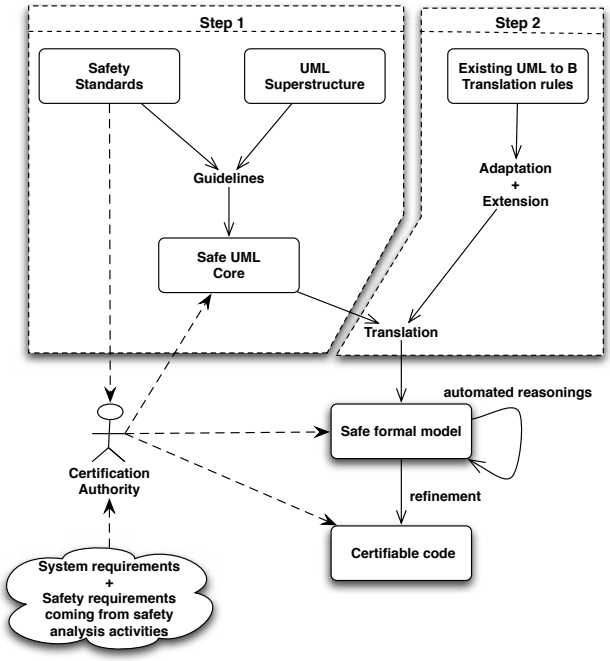

Figure 1: Proposed approach [10]

3 MDA for UML to B

3.1 Why?

A significant effort has been devoted by the research community in order to combine UML and B [9], but the transformation process is subject to some drawbacks:

(*i*) The model-to-model transformation rules present a lack of conceptual basis which prevents to assume or to prove that semantics of models are preserved during the transformation.

(*ii*) Transformations are not explicitly defined which makes it difficult to know on what semantic basis the transformation has taken place.

(*iii*) A variety of personalized UML-to-B tools exist [17, 12, 13, 7], but they are not evolutive (because of the two previous reasons) and produce distinct B specifications from the same UML model.

In order to circumvent these shortcomings, we developed a MDA[1] (Model Driven Architecture) framework intended to involve a generic and automated transformation process from UML to B.

3.2 Contribution: a generic transformation process

The MDA is a development process that aims to separate business logic from underlying platform technology. Otherwise, in MDA a Platform Independent Model (PIM) of a system is formalized and a Platform Specific Model (PSM) is derived from the PIM using transformations.

The overall model transformation process in MDA is defined in terms of meta-model projections: given the meta-models of different modeling languages, a set of transformation rules is defined explicitly using a transformation language. Benefits of MDA raise from the fact that meta-models and transformations are normalized by the meta-language standard of UML called MOF (Meta Object Facility). Indeed, this confers a homogeneous, evolutive and reusable framework to model integration.

We apply this technique in order to define a generic UML-to-B transformation process. Indeed, a generic transformation [11] is a representation of a set of specialized transformations that can be used to manipulate an existing model. As integration of heterogeneous models (B and UML) is described by a sequence of high-level mappings, the transformation process is then defined independently of the kind of transformation and it can be applied to a wide range of translations from UML to B. In MDA the transformation process leans on a precise explicitation of links between models in a meta-level, and then effective transformation is done by instantiating these links. Such an approach allows not only to fill the previously cited gaps of existing UML-to-B transformation approaches, but also to be reused for all these techniques. Fig. 2 shows how we are inspired by the MDA transformation technique in order to put in practice derivation of B specifications from UML models.

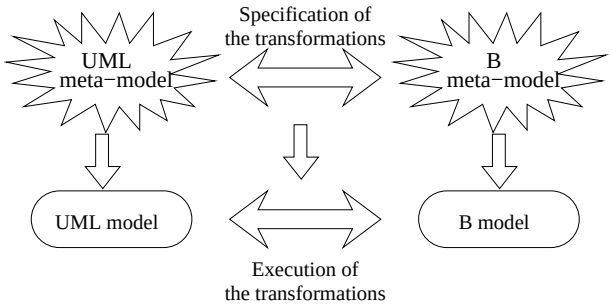

Figure 2: Overview of the proposed UML to B process

3.3 How?

Numerous MDA tools exist to support transformation languages and hence automatically achieve model transformations. In our approach, we use this reusable MDA transformation framework to combine UML and B. So, we first specify a B meta-model based on the MOF; and then we define projections from the UML meta-model to the proposed B meta-model. We have encoded meta-models and projections in the oAW[2] (openArchitectureWare) tool [19] which supports MDA. The tool instantiates the projections on base of effective UML models and then automatically produces their counterparts in form of B models.

It should be noted that in this paper our intention is not to propose a complete B meta-model and present all transformation rules. We only present some fragments of the B meta-model[3], to focus on our approach and its tool support.

3.3.1 Specification of B Abstract machines

An abstract machine, represented by the meta-class *BMachine* (Fig. 3), consists of several clauses allowing to specify the static and dynamic parts of a system. We don't make this distinction by clauses in the B meta-model because we don't find counterparts in UML. It is, certainly, possible to propose an extension of UML to take into account such a structuring. However, we limit our study here, on basic UML and B constructs.

A *BMachine* is composed by data (*BData*), operations (*BOperation*), an invariant (*BInvariant*), a set of parameters (*BMachineParam*), and finally the initialization of variables (*BInitialisation*). The static

[1] http://www.omg.org/docs/ptc/03-08-02.pdf

[2] http://www.openarchitectureware.org/

[3] A complete B meta-model can be found in [8].

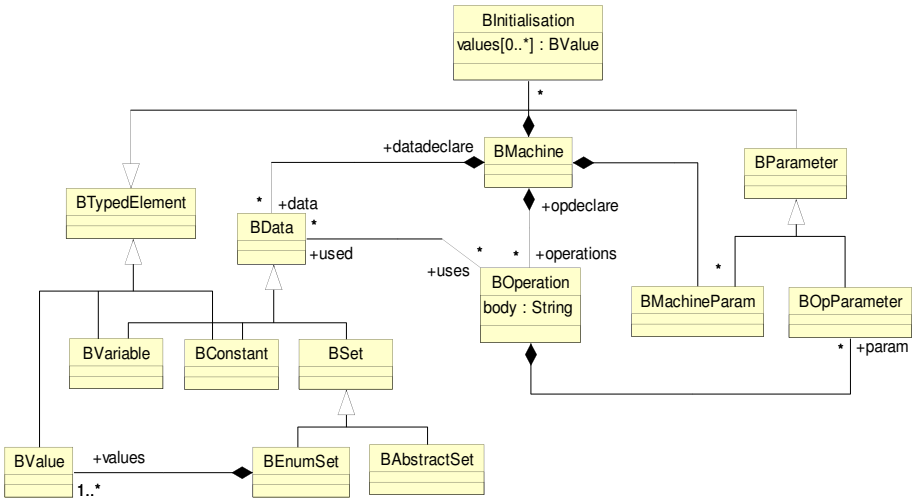

Figure 3: Simplified UML specification of B machines

part corresponds to the declarations of data (*BData*), precisely sets (*BSet*), constants (*BConstant*) and variables (*BVariable*). Note that in a B machine the characterization of these data is specified in terms of properties of the constants (clause PROPERTIES) and invariants (clause INVARIANT). As we don't specify B machine clauses in the B meta-model, we just consider that these data are typed elements (meta-class *BTypedElement*) and they are simply characterized by a typing mechanism (Fig. 4).

3.3.2 Specification of B typing mechanism

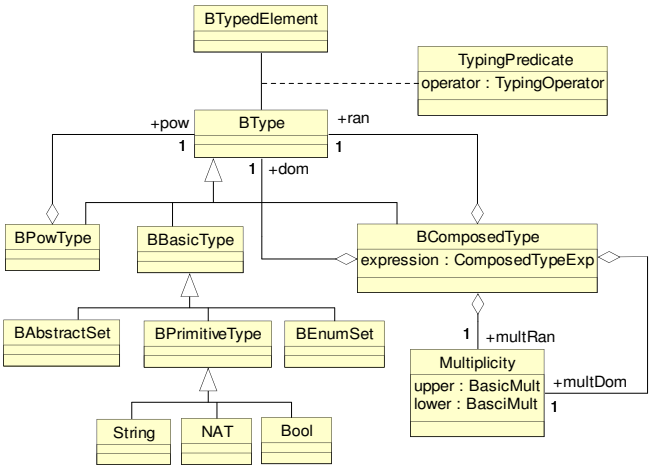

Figure 4: Specification of B typing

Our B meta-model represents an abstract syntax of the most usually used typing constructs in the B language. This is sufficient to cover all typing predicates produced by the existent UML to B transformation rules. Hence, we don't illustrate on a meta-level the bases (or axioms) of B. For example, a variable R can be defined in the invariant by:

$$R = \{r | r \in \mathbb{P}(E_1 \times E_2) \wedge \forall x, y, z \cdot ((x, y) \in r \wedge (x, z) \in r \Rightarrow y = z)\}$$

with E_1 and E_2 two abstract sets (instances of the meta-class *BAbstractSet*). However, such a variable is usually defined by: $R \in E_1 \nrightarrow E_2$ and corresponds to a partial function. Our meta-model is thus appropriate for this second construction of R. Fig. 4 represents a part of the meta-model corresponding to the B typing mechanism.

A *BTypedElement* is linked to a *BType* by the associative meta-class *TypingPredicate*. This one specifies the typing predicates in a B machine on base of the particular operators of equality ($=$), membership ($\in$) and inclusion ($\subset$, $\subseteq$). In the B theory typing is simply carried by the set membership notion. This set can be a basic set, or defined by type constructors ($\times$, $\mathbb{P}$). However, since we don't represent the B theory, but rather the B syntax, we consider the various typing operators (meta-class *TypingOperator* in Fig. 5).

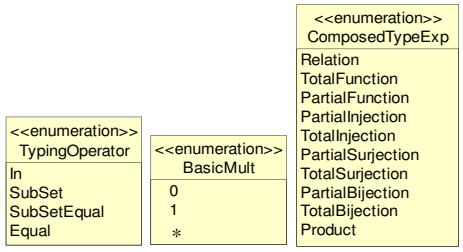

Figure 5: Enumerated meta-classes

Distinction between basic and composed types is carried out by meta-classes *BBasicType* and *BCom-*

posedType. Basic types are predefined types (such as BOOL, NAT, String, etc), and also abstract and enumerated sets defined in the B machine. Composed types are types built by functional relations ($\leftrightarrow$, $\rightarrow$, etc) or by a cartesian product. Each *BComposedType* is thus defined by types forming its domain (role `+dom`) and its range (role `+ran`), and a composed type expression (*ComposedTypeExp* in Fig. 5). Consequently, overlaps of composed types can be easily represented.

The *Multiplicity* meta-class, is defined particularly to represent functional relations. Although these multiplicities are not explicitly defined in B, we introduce them in the meta-model for their adequacy with the transformation of UML associations into B relations (Tab. 1).

	multDom		multRan	
ComposedType	lower	upper	lower	upper
Relation ($\leftrightarrow$)	0	*	0	*
Partial function ($\nrightarrow$)	0	*	0	1
Total function ($\rightarrow$)	0	*	1	1
Partial injection ($\rightarrowtail$)	0	1	0	1
Total injection ($\rightarrowtail$)	0	1	1	1
Partial surjection ($\twoheadrightarrow$)	1	*	0	1
Total surjection ($\twoheadrightarrow$)	1	*	1	1
Partial bijection ($\rightarrowtail\!\!\!\!\twoheadrightarrow$)	1	1	0	1
total bijection ($\rightarrowtail\!\!\!\!\twoheadrightarrow$)	1	1	1	1

Table 1: Possible instances of *Multiplicity*

3.3.3 A simple example

Consider the class diagram *Train_Wagon* shown in Fig. 6. It models an example from a railway context. Two kinds of train are specified: TGV (high speed Train) and TER[4] (ordinary train). A train may own wagons, and wagons may be connected to other wagons (their successors). Wagons can be either first or second class.

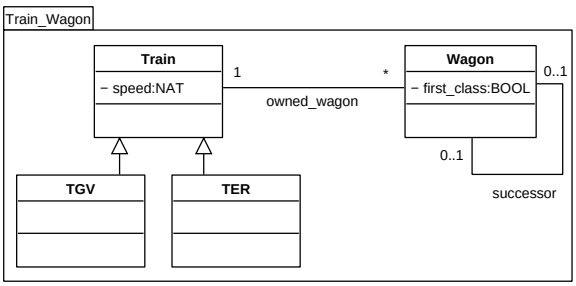

Figure 6: Train_Wagon package

For illustration, we consider simple data structure translation rules proposed by [18], and which produce a unique machine for every UML package. A class is translated into an abstract set (*i.e.* instance of *BAbstractSet* meta-class). A subclass S of a class C leads to an instance of *BVariable* with the typing predicate $S \subseteq C$. Associations and class attributes are traduced by functional relations with respect to the multiplicity table (Tab. 1). As the diagram contains only mono-valued attributes, then only total functions are applicable to translate these attributes. Thus, the data structure of the resulting B specification is as the following:

`MACHINE` *Train_Wagon*
`SETS`
Train ; *Wagon*
`VARIABLES`
TER, TGV, speed, first_class
owned_wagon, successor
`INVARIANT`
$TER \subseteq Train \ \wedge \ TGV \subseteq Train \ \wedge$
$owned\_wagon \in Wagon \rightarrow Train \ \wedge$
$sucessor \in Wagon \rightarrowtail\!\!\!\!\twoheadrightarrow Wagon \ \wedge$
$speed \in Train \rightarrow \texttt{NAT} \ \wedge$
$first\_class \in Wagon \rightarrow \texttt{BOOL}$

4 Our MDA-based tool to transform UML into B

The main guideline we follow to realize the UML-to-B tool is the concept of transparency from the final user point of view. In other words, the tool must be "user-friendly". Moreover, we choose to realize it in a "modern" way, using quite innovator softwares and an object-oriented language (Java). Our UML-to-B tool is based on Eclipse IDE[5], which is an open-source development platform for building, deploying and managing softwares. It is an extensible and universal platform that allows tool developers to add functionalities via tool plug-ins. In our work, the plug-in we use is oAW (openArchitectureWare). This plug-in is composed by numerous tools which allow many MDA-based operations such as: code generation, model-to-model transformation, model verification using constraints, loading/storing models, etc. We use it as a transformation engine and a code generator.

4.1 Overview

The UML and B meta-models are encoded in Eclipse IDE thanks to the Eclipse Modeling Framework (EMF) and more precisely thanks to eCore tool. Indeed, the eCore language used to create models in EMF is conform to the MOF. Furthermore, EMF allows Java code generation from meta-models. This makes it possible to create effective java objects which are up to the encoded meta-models. Translation rules

[4]Express Regional Transport

[5]http://www.eclipse.org/

```
import uml2; import b;
create b::BMachine this translate(uml2::Package p):
   setName(p.name) ->
   data.addAll(p.ownedMember.typeSelect(Class).select(e|e.superClass.isEmpty).setBSets()) ->
   data.addAll(p.ownedMember.typeSelect(Class).reject(e|e.superClass.isEmpty).setBVar(data)) ->
   data.addAll(p.ownedMember.typeSelect(Association).setAsso(data)) ->
   data.addAll(p.ownedMember.typeSelect(Class).reject(e|e.ownedAttribute.isEmpty).setAttrs(data)) ->
   this;
private create b::BAbstractSet this setBSets(uml2::Class c) :
   setName(c.name) -> this;
private create b::BVariable this setBVar(uml2::Class c, List[b::BData] data) :
   setName(c.name) ->
   setType(data.typeSelect(BAbstractSet).selectFirst(e|e.name.matches(c.superClass.first().name))) ->
   this;
```

Figure 7: Example of rules written in xTand module of oAW

are written in oAW (using xTend tool). On this basis, the overall development methodology we adopted to transform UML into B can be summarized in three steps:

(1) *Instantiation*: in this step the input file, which is an XMI serialization of the UML diagram, is parsed. This instantiates the UML meta-model and then produces an effective UML object up to the UML meta-model and ready to be transformed into a B object.

(2) *Transformation*: this step is managed by our transformation engine. It executes an instance of transformation rules on base of the previously produced UML object, and then it generates the corresponding B object.

(3) *Serialization*: this last step generates a B machine source code from the B object (produced by the transformation step). This code generation is performed by oAW thanks to its module xPand.

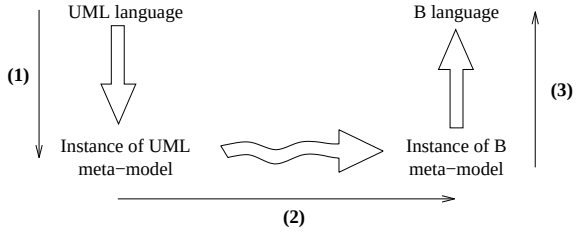

Figure 8: Practical use of oAW

4.2 Transformation and serialization rules

4.2.1 Transformation

Fig. 7 gives a part of the xTand module of oAW in which we have written rules used to produce machine *Train_Wagon* from the class diagram of Fig. 6. These rules start by importing UML and B meta-models. The portion of UML meta-model appointed here is given in Fig. 9.

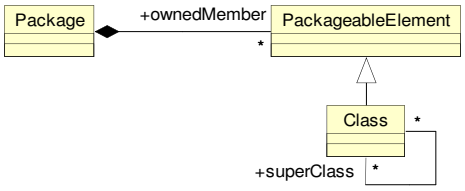

Figure 9: Fragment of UML meta-model

Three rules are presented: (*i*) `translate`: creates a *BMachine* from a package, (*ii*) `setBSets`: generates a *BAbstractSet* for each class in the package, and (*iii*) `setBVar`: generates a *BVariable* from each subclass in the package.

4.2.2 Serialization

Application of the transformation rules of Fig. 7 on the package *Train_Wagon* produces a Java object instance of the meta-class *BMachine*. This object is also linked to instances of meta-classes *BAbstractSet* and *BVariable*. The serialization step aims at interpreting this Java object in order to produce the effective B machine source file. The xPand component of oAW allows to write serialization rules in terms of the B meta-model.

The advantage is that the serialization rules are expressed once and for all and are applicable on every object instance of the B meta-model. Serialization rules conform to the transformation approached in section 3.3.3 are given in Fig. 10.

4.3 Java integration

In order to develop an autonomous tool, we have to run the transformation and the serialization rules in an Eclipse Action, implemented in Java. As a result,

```
≪ IMPORT b ≫
≪ DEFINE file FOR BMachine ≫
    ≪ FILE name + ‘‘.mch’’ ≫
    MACHINE ≪ name ≫
    SETS
    ≪ FOREACH data.typeSelect(BAbstractSet)
           AS dabs SEPARATOR ’;’- ≫
        ≪ dabs.name- ≫ ≪ ENDFOREACH ≫
    VARIABLES
    ≪ FOREACH data.typeSelect(BVariable)
           AS dvar SEPARATOR ’;’- ≫
        ≪ dvar.name- ≫ ≪ ENDFOREACH ≫
    INVARIANT
    OPERATIONS
    ≪ ENDFILE ≫
≪ ENDDEFINE ≫
```

Figure 10: Example of serialization rules in xPand

the final user will just have to push a button (button UML2B in Fig. 11) to obtain the B model from a UML diagram.

Fig. 11 shows the output of the tool after running rules of Figs. 7 and 10 on the *Train_Wagon* package. In the oAW plug-in, the Eclipse Action is developed thanks to the “Facades” mechanisms. They consist of particular classes which allow to manipulate xTend and xPand files with Java. Thus, in order to develop the UML2B Eclipse Action we used both XTendFacade and XpandFacade.

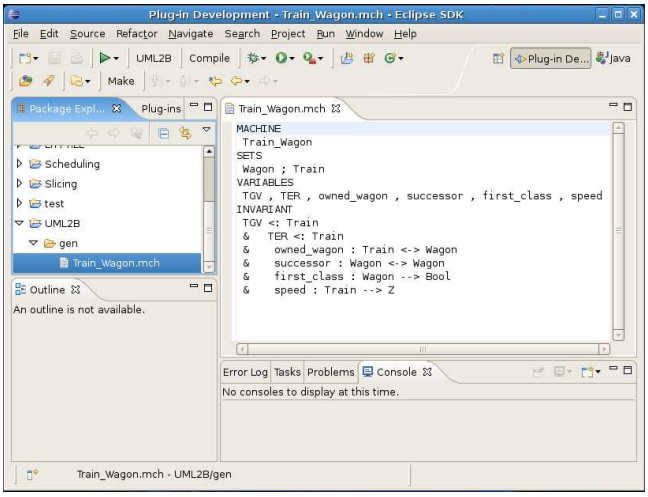

Figure 11: Screenshot of UML-to-B tool

4.4 Advantages of the tool and future directions

Contrary to the existing translation tools from UML to B, our tool is based on a generic transformation process. Indeed, it can be easily adapted to any existing translation approach by simply implementing transformation and corresponding serialization rules. So the basic idea of this integration by meta-modeling is to make a constraints and provide some basic features that B and UML modeling notation should have. According to this the exchange tool can be easily built. However, our tool was applied to simple UML examples. We note that UML models may be more complex and build structures in the target model which do not directly correspond to any individual element in the source model. Scaling up to such realistic sizes may bring interesting new problems to our tool.

In this paper we considered a simple translation in which a class diagram is traduced by a unique B machine. However, more complicated translations exist such as those proposed by [12] where a B machine is produced for each class in the class diagram. Such a translation will lead to as many B machines as UML classes and hence composition links between these B machines must be considered. Thus, in order to reuse our MDA-based framework for such an approach we will need to extend the B meta-model to B compositions. Actually, we are working on these extensions in order to encode, in oAW, the more complete B meta-model.

In our work (Fig. 1), we select the most pertinent UML to B translations and apply them in safety critical systems (railway applications). So our tool must allow to run every existing rule in order to evaluate whether the resulting B specification is useful or not for these particular systems. Further experiments are needed to broaden the scope of our tool and address this challenge. Actual version of the tool takes into account only UML structural features (*i.e.* class diagram). We plan in the future to also integrate transformations of UML behavioural features (*i.e.* state/transition diagrams, etc).

5 Conclusion

Derivation of B specifications from a UML model aims, on the one hand, to use UML as a starting point of the development process, and in the other hand, to use B tools to analyze and prove the correction of resulting formal specifications. These specifications can also be exploited to check constraints expressed initially in OCL or to produce source code after a succession of proven refinements.

This paper has proposed an original approach and its tool support for the derivation of B specifications from UML diagrams. First, a B meta-model is proposed to give an abstract UML syntax to the B language. Then, existing translation rules from UML to

B proposed by other research teams are interpreted in term of projections from the UML meta-model to the B meta-model and they are encoded in the oAW plug-in of Eclipse IDE. The proposed tool instantiates the transformation rules on base of an effective UML model and automatically produces its counterpart B specification.

The UML to B translation approaches have received a lot of interest in the last decade from the scientific community. Moreover, they still personalized and unusable for large-scale applications. We hope that the technique described in this paper will contribute to their integration outside of the traditional formal/semi-formal methods communities, and lead to their applicability in a realistic industrial context.

References

[1] Functional safety of electrical / electronic / programmable electronic safety-related systems, 2000.

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

[3] F. Badeau and A. Amelot. Using B as a High Level Programming Language in an Industrial Project: Roissy VAL. In *ZB*, volume 3455 of *LNCS*, pages 334–354. Springer, 2005.

[4] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. METEOR: A successful application of B in a large project. In *FM'99*, volume 1709 of *LNCS*, pages 369–387. Springer-Verlag, 1999.

[5] J.-L. Boulanger and P. Bon. BRAIL requirement analysis. In *Proceedings of FORMS'04*, pages 221–229.

[6] H. Fecher, J. Schnborn, M. Kyas, and W.-P. Roever. 29 New Unclarities in the Semantics of UML 2.0 State Machines. In *ICFEM 2005*, volume 3785 of *LNCS*, pages 52–65. Springer, 2005.

[7] L. Hazem, N. Levy, and R. Marcano-Kamenoff. UML2B : un outil pour la génération de modèles formels. In *AFADL'2004*, 2004.

[8] A. Idani. *B/UML : Mise en relation de spécifications B et de descriptions UML pour l'aide la validation externe de développements formels en B*. PhD thesis, Université de Grenoble - France, November 2006.

[9] A. Idani. *Computer Software Engineering Research*, chapter UML2B vs B2UML: Bridging the gap between formal and graphical software modelling paradigms. Nova Science Publishers, 2007. (18 pages), ISBN: 1-60021-774-5.

[10] A. Idani, D.-D. Okalas Ossami, and J.-L. Boulanger. Commandments of UML for safety. In *2nd International Confernce on Software Engineering Advances*. IEEE CS Press, August 2007.

[11] J. Kovse and T. Härder. Generic XMI-Based UML Model Transformations. In *8th International Conference on Object-Oriented Information Systems, OOIS'02*, volume 2425 of *LNCS*, pages 192–198, UK, 2002. Springer-Verlag.

[12] R. Laleau and A. Mammar. An Overview of a Method and Its Support Tool for Generating B Specifications from UML Notations. In *15th IEEE Int. Conf. on Automated Software Engineering*, pages 269–272, 2000. IEEE CS Press.

[13] H. Ledang, J. Souquières, and S. Charles. Argo/UML + B: un outil de transformation systématique de spécification UML en B. In *AFADL'2003*, pages 3–18, Jan. 2003.

[14] D.-D. Okalas Ossami, J.-M. Mota, and J.-L. Boulanger. A model process towards modeling guidelines to build certifiable UML models in the railway sector. In *Proceedings of the 7th International SPICE Conference (Software Process Improvement and Capability dEtermination)*, 2007.

[15] D.-D. Okalas Ossami, J.-M. Mota, L. Thiry, J.-M. Perronne, and J.-L. Boulanger. A method to model guidelines for developing railway safety-critical systems with UML. In *Proceedings of the International Conference on Software and Technologies (ICSOFT'07)*, Barcelone, Spain, 2007.

[16] A. J. H. Simons and I. Graham. 30 Things that go wrong In Object modelling with UML 1.3. In H. Kilov, B.Rumpe, and I. Simmonds, editors, *Behavioral Specifications of Businesses and Systems*, chapter 17, pages 237–257. Kluwer Academic, 1999.

[17] C. Snook and M. Butler. U2B-A tool for translating UML-B models into B. In J. Mermet, editor, *UML-B Specification for Proven Embedded Systems Design*, 2004.

[18] C. Snook and M. Butler. UML-B: Formal modeling and design aided by UML. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(1):92–122, 2006.

[19] M. Völter. openArchitectureWare 4 - The flexible open source tool platform for model-driven software development. 2006.

[20] J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Precise Modeling With UML*. Addison-Wesley Object Technology Series. Addison-Wesley Professional, 1998.