Developing Specifications by using Operators: a Process to guarantee correctness by construction

Dieu Donné Okalas Ossami, Jeanine Souquières, Jean-Pierre Jacquot

Developing Specifications by using Operators: a Process to guarantee correctness by construction

Dieu Donné Okalas Ossami, Jeanine Souquières, and Jean-Pierre Jacquot

LORIA - Université Nancy 2 - UHP Nancy 1
Campus scientifique, BP 239
54506 Vandœuvre-lès-Nancy Cedex - France
Email: {okalas,souquier,jacquot}@loria.fr

Abstract. We propose a process model for the development of specifications based on the notions of multi-view state and of operators. A specification state consists of a UML view and a B expression of the same specification. Operators model design decisions as the simultaneous transformation of UML and B descriptions. A specification development is a sequence of applications of operators. We define a notion of coherence of a specification state which allows us to define formally a notion of correctness for operators. Thus, the development process guarantees that the specification can be safely verified and validated.

Keywords: consistency, correctness, verification, validation, operator, development process, multi-view, UML, B.

1 Introduction

Experience has shown that the most critical and least supported phases of the software life cycle are requirement analysis and specification. Errors and misconceptions in the requirements will be passed on the system specifications and from them down the process to show up ultimately in the programs. Formal specifications could greatly help in reducing the amount of errors because of the absence of ambiguity in formal texts and the availability of powerful analysis techniques and prototyping tools. However, formal specifications are hard to write and, more importantly, hard to read; this raises the problem of the validation of the specification. We believe that the effective availability of tools supporting specification development could greatly help in promoting the use of formal specifications by practitioners. Tool support should include guidance during the specification development process; it should enable users to develop specifications in an intuitive fashion by separating the use of design concepts from the technical details of how they are captured in specification languages. The specification development process should be problem oriented instead of language oriented.

Validation requires users of the system to be able to “read” the specification, hence the importance of graphical notations. Verification requires a formal notation. The current issue is that no single language offers both kinds of notation. Is it possible to combine graphical notations and formal languages? Currently, there are two mains streams of specification languages: graphical notations such as UML [22] and mathematical notations such as B [1]. Our goal is to design a framework where both kinds of notations can be used together to fulfill the needs of all the people involved. Our approach aims

at capitalising on existing languages rather than at defining a new one. This allows us to reuse the efforts that have been out in the production of industrial tools such as Rational Rose[1] or ArgoUML[2] for the edition of UML diagrams, and such as *l'Atelier B* [26], B-Toolkit[3], or B4Free [4] for the formal verification of specifications. Our framework supports multi-view specification activity by providing assistance during the development process. Its key is the notion of operator: the development of a specification is defined as a sequence of steps, each of which maps a development state to the next by the application of an operator.

The formalisation of object-oriented concepts has prompted many research works. Three general approaches are identified in the literature: (1) extension of formal notations with object-oriented concepts, (2) extension of object-oriented notations with formal notations, and (3) method integration between object-oriented and formal notations. Z++ [12] and Object-Z [5] are examples of the first approach where Z [25] is supplemented with object-oriented concepts and notations. In the second approach, parts of the informal specifications expressed in natural language are replaced by formal statements expressed in a well-known formal language, e.g. Syntropy [6]. In the third approach, transformation rules are defined which translate specification written in one formalism into an "equivalent" specification written in another formalism. One instance of this approach is UML to B transformation: it allows specifiers to use formal techniques and tools to check the specification. Transformation provides us with automated support to generate a B specification from UML diagrams [10, 18, 14, 23] taking into account OCL constraints [15, 17]. Another instance is B to UML transformation: it eases the validation by the generation of UML diagrams (class diagrams and state diagrams) from a B specification [7, 8, 27].

One major problem in UML and B integration approaches is maintaining the consistency when the specification evolves. Currently, UML and B integration approaches offer either UML to B [16, 24] or B to UML transformations [28] but not both in the same tool. Several reasons account for this state of affair, but the net result is the practical impossibility to define a process where both kinds of transformation can be symmetrically used. As a consequence, UML to B or B to UML transformation induces a sequential development process where: (i) a new specification in the target formalism is generated each time the rules are used. Thus, any information that was previously added in the generated specification is lost and must be redesigned; (ii) the modifications brought into the generated specification cannot be retrofitted. This raises the issue of consistency between the current B specification and its corresponding UML specification [11].

The paper is organised as follows. Section 2 presents the approach with a definition of the consistency relation between two developments steps to ensure the correctness of the construction. Section 3 presents a selection of development steps on the generalised railroad crossing case study. Section 4 concludes the paper.

[1] http://www-306.ibm.com/software/rational
[2] http://www.argouml.tigris.org

2 Description of the approach

Our approach aims at modelling a process for developing specifications expressed simultaneously in an object oriented notation graphical (UML) and in a formal notation (B). Formal and semi-formal descriptions are built by successive approximations. Operators are the central notion: they capture strategies and design concepts. They enable the user to develop specifications in an intuitive fashion by separating the use of design concepts from the technical details of how they are captured in the chosen specification languages. Different development strategies can be models as libraries of operators. It is possible to provide users with flexible development processes.

2.1 Specification state

The process model we have developed is strongly inspired by the transformation approaches. The final specification results from a sequence of applications of transformers: *operators*. An operator is applied to a *specification state* and produces a new specification state.
A specification state consists in two views. The UML view provides users with a graphical notation and access to validation tools. The B view provides users with a formal notation and access to verification tools. The fundamental point is that the views are *two different expressions* of the *same specification*. A state is noted as:

$$Spec = \langle SpecUML, SpecB \rangle$$

2.2 Development operators and development step

A development operator transforms simultaneously the UML and the B views [19]. Often, the application of an operator requires some input from the user. This is modelled as operator parameters. An operator consists of:

- an application condition; it is a predicate on the current development state. It must hold for ensuring the preservation of the consistency property of the specification *Spec* after the application of the operator;
- a description of the actions on *SpecUML* and *SpecB*, noted $\mathcal{O}_{UML}$ and $\mathcal{O}_B$ and
- hints about operators that can be applied next; those can be used to assist users in following a development strategy.

Libraries of development operators support the modelling of strategies. At each development step, the specifier chooses an operator in the library and provides it with the required parameters. If the application condition holds on the correct state, the actions are conducted to lead to a new development state.

2.3 Correctness by construction

Our development model is based on the idea that applying a “correct” operator on a “correct” state leads to a new “correct” state. The question is now to define what “correct” means precisely.

We propose to define the correction of a specification state by a *consistency relation* between its views. Let us denote $\mathcal{R}_{el_C}$ the consistency relation between *SpecUML* and *SpecB*.

Let $\mathbb{T}_{U \to B}$ be the set of UML to B transformation rules [13, 18] which associate each UML artifact with one or more B artifacts. These transformations are relative to UML 1.x [21]. $\mathcal{R}_{el_C}$ is defined as a conjunction of four conditions:

1 *Syntactic conformance.* It states that both *SpecUML* and *SpecB* must be well-formed. It ensures that the specification conforms to abstract syntax specified by the meta-model, i.e. UML meta-model or B abstract syntax tree. Let $\mathcal{WF}(SpecUML)$ and $\mathcal{WF}(SpecB)$ be two predicates defining if a UML and a B specifications are well-formed.

2 *Local consistency.* It requires that both specifications must be internally consistent. That means they do not contain contradictions, but they could be incompletely defined. We write it *consistent(SpecUML)* and *consistent(SpecB)*.

Global consistency is defined with respect to UML to B transformation rules designed by Meyer, Souquières and Ledang [13, 18].

3 *Elements traceability.* It states that for any elements of $ID(SpecUML)$, e_U, that can be transformed by a rule T, there exists in $ID(SpecB)$ a set of artifacts $\{e_B\}$ resulting from the application of T to e_U.

4 *Semantic preservation.* It states that any statement ϕ satisfying the semantics of *SpecUML* must satisfy *SpecB*. The semantics of *SpecUML* is defined as $\mathbb{T}_{U \to B}(SpecUML)$. This means that UML artifacts that have no B semantics defined in $\mathbb{T}_{U \to B}$ are not concerned by the consistency relation $\mathcal{R}el_C$. This has important implications throughout the verification process. For example, it is well known that checking pairwise integration of a set of software specifications is only possible if one is able to transform them into a semantic domain supported by tools. B is our semantic domain and any UML statement that has no B formalisation cannot be verified in our framework.

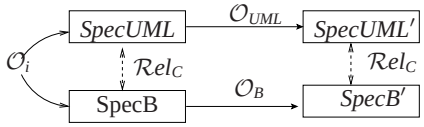


Fig. 1. Correctness of an operator $\mathcal{O}_i$

We use the B theorem prover to prove that a statement ϕ holds in *SpecB* (condition (2)) and due to *condition (3)*, we derive the consistency of *SpecUML*, and therefore the consistency of the multi-view specification *Spec*.

Definition 1 (Consistency relation)

SpecUML $\mathcal{R}el_C$ *SpecB* :

(1) $\mathcal{WF}(SpecUML) \wedge \mathcal{WF}(SpecB)$

(2) *consistent(SpecUML)* $\wedge$ *consistent(SpecB)*

(3) $\forall\ e_U.(e_U \in ID(SpecUML_{|_{\mathbb{T}_{U\to B}}})$ [a] $\Rightarrow$
$\exists\ \{e_B\},\ T.(\{e_B\} \subseteq ID(SpecB) \wedge T \in \mathbb{T}_{U\to B} \wedge T(e_U) = \{e_B\}))$

(4) $\forall\ \phi.(\mathbb{T}_{U\to B}(SpecUML)$ [b] $\vDash \phi \Rightarrow SpecB \vDash \phi))$

[a] $SpecUML_{|_{\mathbb{T}_{U\to B}}}$ denotes the restriction of *SpecUML* to elements for which there is a transformation rule to B defined in $\mathbb{T}_{U\to B}$

[b] $\mathbb{T}_{U\to B}$(SpecUML) denotes the application of the set of UML to B transformation rules on *SpecUML*

The notion of correctness for operators is then easy to derive: an operator is correct iff its application on a state where *SpecUML* and *SpecB* are consistent and its application condition holds leads to a state where $\mathcal{O}_{UML}(SpecUML)$ and $\mathcal{O}_B(SpecB)$ are consistent, see Figure 1. Proof that an operator is correct may not be easy but once done, it ensures that *all* applications of that operator produce consistent specifications.

3 A small case study

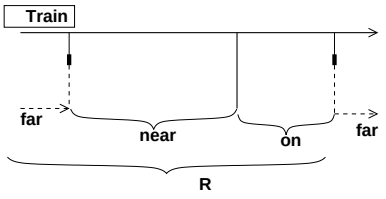


Fig. 2. The generalised railroad crossing

We consider the generalised railroad crossing case study, called the *GRC* problem [9]. The system to be specified aims at controlling a gate at a railroad crossing so that trains can safely go through. The informal text describes the problem as a monitoring of trains. The GRC lies in a region of interest *R*, as presented in Figure 2. Trains travel in one direction through *R*, which is decomposed into three regions : *far*, *near* and *on*. The regions determine the position of trains in *R*. Each of these regions has a light signal, which may be *red*, *yellow* or *green* and which is seen by the train when it leaves the previous region and enter the next. A train leaves a region when its last carriage leaves the region. It is at this point that lights state may change and require the train to stop.

We present three development steps to illustrate our approach; we start from this informal description. For each step, we give the idea we follow, the operator chosen in the library with its parameters and the new state produced by the application of the operator on the current state of the specification. In the new specification state, the new UML part is written in bold face and the new B part is in a box. For more information on the definition of operators, see [19]. For space reasons, we will concentrate on the definition of the train component.

3.1 First development step

From the informal requirement, we identify three states (*far*, *near*, *on*) and three events (*enter*, *cross*, *leave*) which change the state of the train when it arrives, crosses or leaves the region *R*. This leads us to use the classical specification technique of introducing a state machine to model the description. This technique is captured in the operator called *Model-StateMachine*. The required parameters are easily extracted from the text description and the application of the operator leads to the development state presented in Figure 3. The resulting UML view is composed of a class diagram with one class (*Train*) and an enumeration (the states), and of a state-transition diagram. Three machines and a refinment have been introduced in the B view.

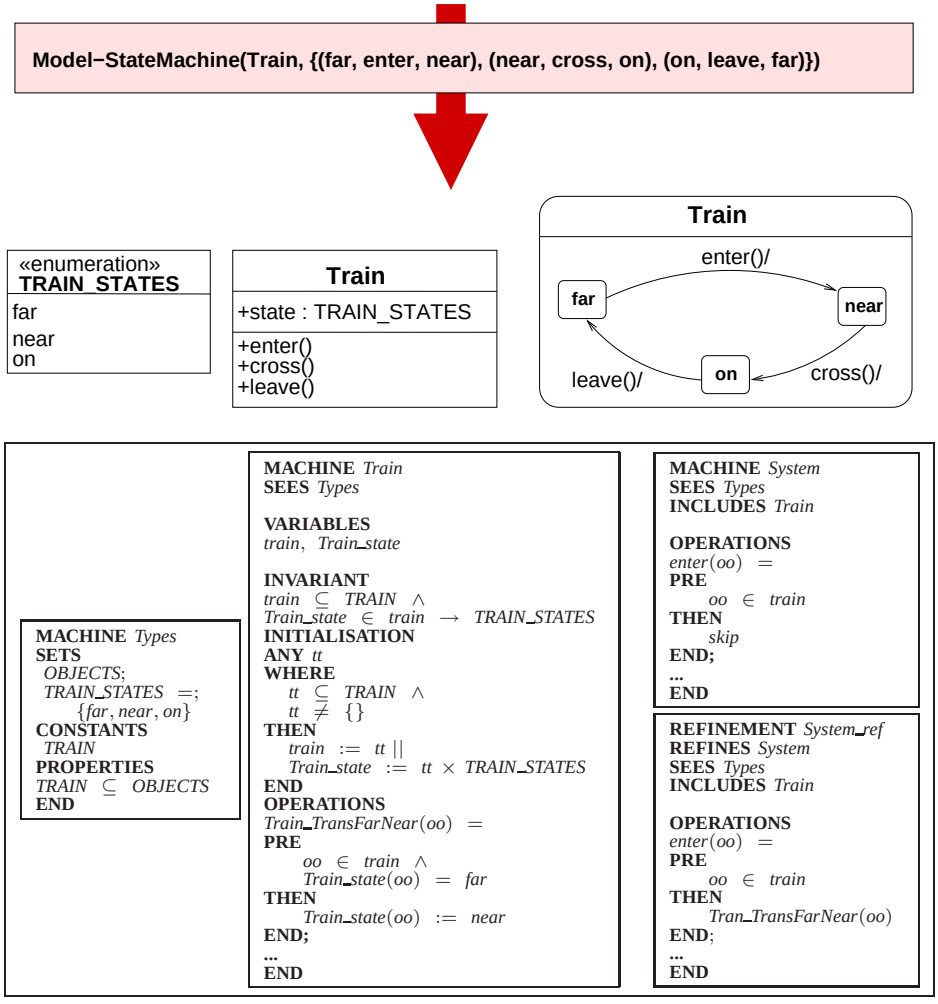
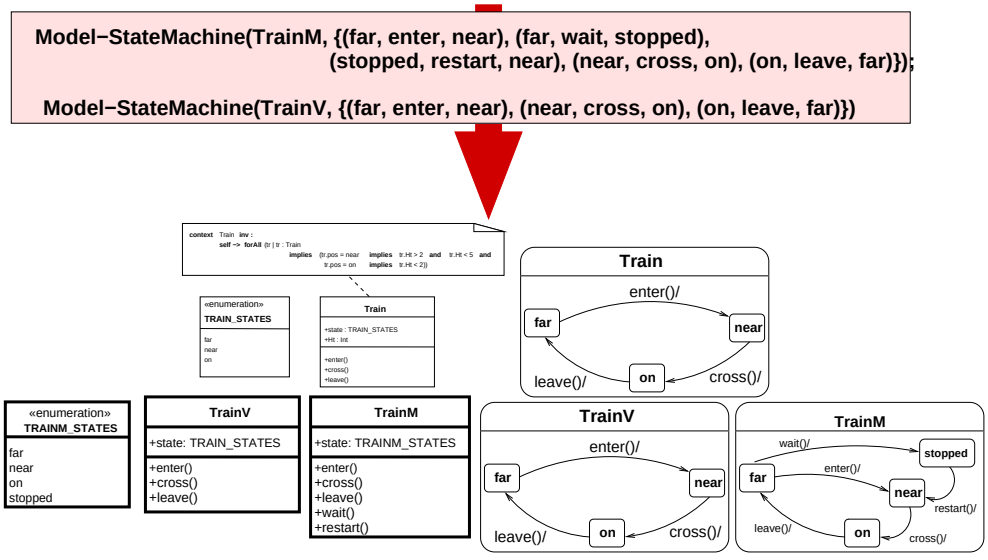


Fig. 3. Application of the Model-StateMachine operator at the beginning of the development

3.2 Introduction of different kinds of train



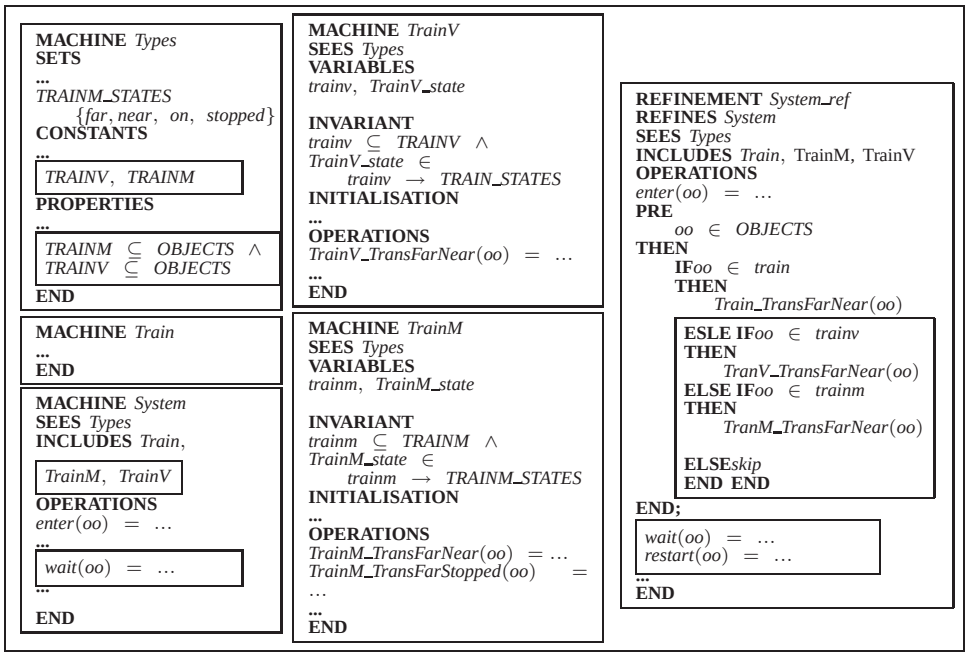


Fig. 4. Introduction of two kinds of trains

Further analysis of the problem indicates that different kinds of trains are authorised to travel on the GRC: freight trains and passenger trains. The following characteristics are identified:

- freigth trains can stop when they reach the state *far* after the event *wait* occurs. They go from the state *stopped* to the state *near* when the event *restart* occurs;
- passenger trains are of two types, *TGV* and *TrainCorail*.

To introduce the different trains, we have a choice between at least two development approaches: passenger and freight trains can be modeled independantly from the *Train* entity, or they can be modeled as specialisation of the *Train* entity. Let us use the first approach which corresponds to a bottom-up strategy.
We use again the *Model-StateMachine* operator, once for the freigth trains (*TrainM*) and once for the passenger train (*TrainV*). The new specification state is presented in Figure 4. Two classes, one enumeration, two state diagrams have been introduced in the UML view. Two new machines have been introduced and three other entities (*Types*, *System*, and *System_ref*) have been updated in the B view.

Note that there is a development step missing here; namely the one concerning the addition of the invariant with associated attribut Ht on which the invariant is expressed in the context the class Train.

3.3 Generalisation

The bottom-up approach lead to introduce three unconnected entities. A close look on the diagrams and machines reveals strong similarities. In fact, we have modeled twice the same general behaviour. Moreover, we have now enough knowledge of the problem to realize that *enter*, *cross* and *leave* are three instances of the same behaviour: *move*. This situation is quite common while developing specification and can be solved by generalising.
A generalisation operator `Generalize-Operation` models this approach. We select the parameters to indicate that *TrainV* and *TrainM* are subkinds of *Train* and that one operation, *move*, replaces the other three.
The new specification state is presented Figure 5. We can note that the UML view has been augmented with inheritance relations and the attribute lists of the classes have been adapted. The B view shows modifications in the classes. It should be noted that the B view undergoes many modifications, but all of them are systematic and easily computed.
Each operator that we used in this development case study can be proved to be correct. Hence, we are ensured that a verification of the B view with a B prover and a validation on the UML view are two checking processes for the same specification.

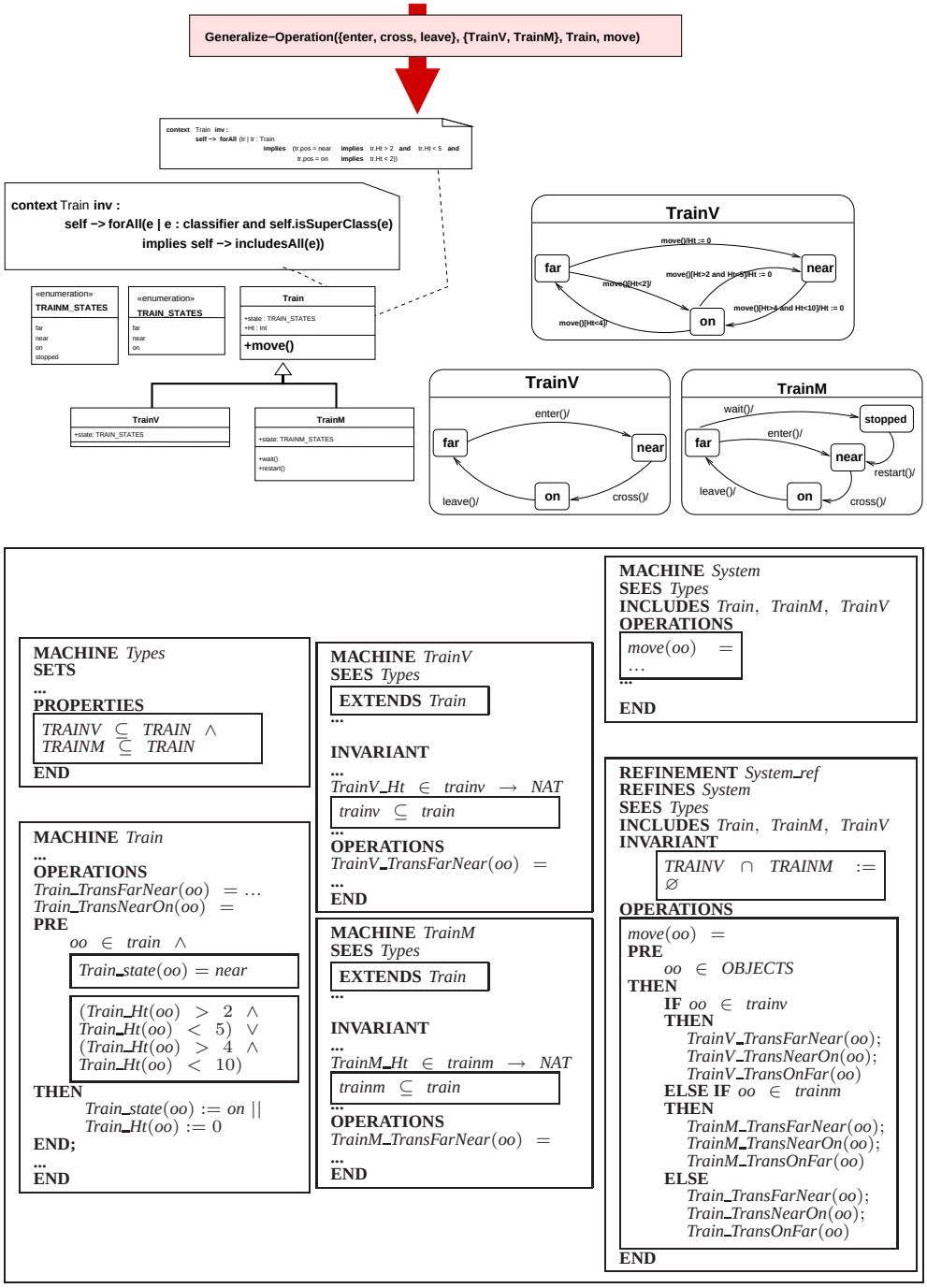


Fig. 5. Application of the Generalize-Operation operator on the specification state of FIG 4

In ordr to avoid indeterminisms in the state diagram associated to the super-class Train due to the replacement of the enter, cross and leave by move, somme guards have been added to corresponding transitions. This is done by using the appropriate operator present in operators' library.

4 Conclusion

This paper presents a specification development process which integrates the use of several formalisms. The key notion is the operator which models and implements a property of correctness for the evolution of a multi-view specification. The idea to mix different formalisms is not new but was hampered by the problem of maintaining the consistency between the two specifications. Operators solve this problem. They enable users to develop specifications in an intuitive fashion by separating the use of design concepts from the technical details of how they are captured in specification languages. They offer flexibility since it is possible to define libraries of operators capturing alternative definitions of particular concepts and strategies. They allow us to model the development of a specification as a process of successive approximation. The purpose of operators is to capture the specifiers' knowledge.

The benefits of the approach can be summed up as follows:

- separation of concern. Operators enable the specifier to focus on methodological issues and on problem solving issues rather than to focus on how to express them in the target languages;
- documentation. The use of two complementary languages, one graphical and object-oriented and the other formal, makes the specification easier to understand and help the developers to verify and refine the system under development;
- support for guidance. At any stage of the construction process, the specifier knows what remains to be done. Libraries of operators with a liberal use of the "remain to be done" clause can be constructed to model and enforce particular development strategies. In addition, operators preconditions lower the risk of mis-using operators;
- correctness by construction. As the correctness of each operator has been defined, the specification obtained by the application of operators is proved to be correct.

Operators can be compared with specification templates introduced in [29], where a template formalises a Lotos specification style for OSI as a fragment of specification text that can be conveniently retrieved and inserted in a specification. To enhance the value of such templates and to increase their generality, templates are parameterised.

An implementation of this framework with some operators is under development. It is an extension of the *ArgoUML+B* [16] platform, allowing to automatically transform some UML diagrams to B specifications (*ArgoUML+B* is based on the *ArgoUML*[3] project, dedicated to the edition and design of UML diagrams). This extension includes *SmartTools* [2, 20] to dynamically represent B specifications as instances of the B AST (abstract syntax tree), taking into account the multi-view specification.

[3] http://www.argouml.tigris.org

References

[1] J.R. Abrial. *The B Book -Assigning Programs to Meanings.-*. Cambridge University Press, 1996. ISBN 0-521-49619-5.

[2] I. Attali, C. Courbis, P. Degenne, A. Fau, J. Fillon, D. Parigot, C. Pasquier, and C. S. Coen. SmartTools: a development environment generator based on XML technologies. In *In XML Technologies and Software Engineering. ICSE workshop*, 2001.

[3] Oxford(UK) B-Core(UK) Ltd. *B-Toolkit User's Manual*. 1996.

[4] B4Free. avaible at : http://www.b4free.com/.

[5] D.A. Carrington, D. Duke, R. Duke, P. King, G.A. Rose, and G. Smith. Object-Z: An object-oriented extension to Z. In *Formal Description Techniques II, FORTES'89*, pages 281–296, 1990.

[6] S. Cook and J. Daniels. Let's get formal. *Journal of Object-Oriented Programming (JOOP)*, pages 22–24 and 64–66, 1994.

[7] F. Houda and S. Merz. Transformation de spécifications B en diagrammes UML. In *Proceedings of AFADL'04*, 2004.

[8] A. Idani and Y. Ledru. Object Oriented Concepts Identification from Formal B Specifications. In *9th Int.Workshop on Formal Methods for Industrial Critical Systems, FMICS'04*, 2004.

[9] L. Jansen and E. Schnieder. Traffic control system case study: Problem description and a note on domain-based software specification. Technical report, Colorado State University, January, 2000.

[10] R. Laleau and F. Polack. A Rigorous Metamodel for UML Static Conceptual Modelling of Information Systems. In *Advanced Information Systems Engineering. 13th Int. Conf., CAiSE 2001*, volume 2068 of *LNCS*, pages 402–416. Springer, 2001.

[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 – Formal Specification and Development in Z and B*, pages 517–534, 2002.

[12] K. Lano. Z++, an object-orientated extension to z. In *Proceedings of the Fifth Annual Z User Meeting*, pages 151–172. Springer-Verlag, 1991.

[13] H. Ledang and J. Souquières. Modeling class operations in B: application to UML behavioral diagrams. *ASE2001: 16th IEEE Int. Conf. on Automated Software Engineering, IEEE Computer Society*, 2001.

[14] H. Ledang and J. Souquières. Integrating Formalizing UML Behavioral Diagrams with B. *Workshop on Integration and Transformation of UML models*, 2002.

[15] H. Ledang and J. Souquières. Integration of UML and B Specification Techniques: Systematic Transformation from OCL Expressions into B. In *APSEC 2002, IEEE Computer Society*, 2002.

[16] H. Ledang, J. Souquières, and S. Charles. ArgoUML+B : Un outil de transformation systématique de spécifications UML vers B. In *Proceedings of AFADL'03*, 2003.

[17] R. Marcano and N. Levy. Using B formal specifications for analysis and verification of UML/OCL models. In L. Kuzniarz, G. Reggio, J. L. Sourrouille, and Z. Huzar, editors, *Workshop on Consistency Problems in UML-based Software Development*, pages 91–105, 2002.

[18] E. Meyer and J. Souquières. A systematic approach to transform OMT diagrams to a B specification. In *Proceedings of the Formal Method Conference*, number 1708 in LNCS, pages 875—895. Springer-Verlag, 1999.

[19] D. Okalas Ossami, J. Souquières, and J-P. Jacquot. Consistency in UML and B multi-view specifications. In LNCS, editor, *Proceeding of the International Conference on Integrated Formal Methods, IFM'05*, number 3771, pages 386–405, 2005.

[20] D. Parigot and C. Courbis. avaible at : http://www-sop.inria.fr/smartool/.

[21] J. Rumbaugh, I. Jacobsen, and G. Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley, 1997.

[22] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998. ISBN 0-201-30998-X.

[23] C. Snook, M. Butler, and I. Oliver. Towards a UML profile for UML-B. Technical report, DSSE-TR-2003-3, Electronics and Computer Science, University of Southampton, 2003.

[24] C. Snook and M. Buttler. U2B: a tool for combining UML and B. Avaible at http://www.ecs.soton.ac.uk/ cfs/U2Bdownloads/.
[25] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.
[26] STERIA. *Manuel de référence du langage B*. -ClearSy-, novembre, 1998.
[27] B. Tatibouet and J.-C. Voisinet. Generating statecharts from B specifications. In *16th Int. Conf. Software & Systems Engineering and their applications, ICSSEA'2003*, 2003.
[28] B. Tatibouet and J.C. Voisinet. jBtools and B2UML : a plateform and a tool to provide a UML class diagram since a B specification. In *ICSSEA : 14th Int. Conf. on Software and Systems Engineering and Their Applications*, volume 2, 2001.
[29] K. J. Turner. Relating architecture and specification. *Computer Networks and ISDN Systems*, April 1996.