Integration of UML Views using B Notation

Hung Ledang, Jeanine Souquières

Integration of UML Views using B Notations

Hung LEDANG Jeanine SOUQUIÈRES

LORIA - Université Nancy 2 - UMR 7503
Campus scientifique, BP 239
54506 Vandœuvre-les-Nancy Cedex - France
E-mail: {ledang,souquier}@loria.fr

Abstract

The translation from UML specifications to formal B specifications gives a way to analyse rigorously UML specifications via their corresponding B formal specifications. This point is significant thanks to B support tools. This paper reports our experiences on UML-into-B translation that emphasise on the integration of different kinds of UML diagrams into B specifications.

Keywords : *UML, class operation, use case, event, B method.*

1. Introduction

The Unified Modelling Language (UML)[18] has become a de-facto standard notation for describing analysis and design models of object-oriented software systems. The graphical description of models is easily accessible. Developers and their customers intuitively grasp the general structure of a model and thus have a good basis for discussing system requirements and their possible implementation. However, since the UML concepts have English-based informal semantics, it is difficult even impossible to design tools for verifying or analysing formally UML specifications. This point is considered as a serious drawback of UML-based techniques.

To remedy such a drawback, one approach is to develop UML as a precise (i.e well defined) modelling language. The pUML[1] (precise UML) group has been created to achieve this goal. However the main challenge [6] of the pUML is to define a new formal notation that has been up to now an open issue. Furthermore, the support tool for such a new formalism is perhaps another challenge.

In waiting for a precise version of UML and its support tool, the necessity to detect semantic defects inside UML specifications should be solved in a pragmatic approach (cf. [20, 5]) : formalising UML specifications by existing formal languages and then analysing UML specifications via the derived formal specifications. In this perspective, using the B language [1] to formalise UML specification has been considered as a promising approach [19, 16, 17, 15]. By formalising UML specifications in B, one can use B powerful support tools like AtelierB [21], B-Toolkit [3] to detect and analyse inconsistencies and defects inside UML specifications [12]. On the other hand, we can also use UML specifications as the starting point to develop B specifications which can be then refined automatically to the executable code [2, 9].

This paper outlines our research on UML-B integration. Section 2 recalls briefly the B method. Section 3 presents our approach to integrate UML views into one B specification. Three derivation procedures, which are based on use cases, class operations and events, are introduced. Concluding remarks in Section 4 complete our presentation.

2. The B method

B [1] is a formal software development method that covers a software process from specification to implementation. The B notation is based on set theory, the language of generalised substitutions and first order logic. Specifications are composed of abstract machines that are similar to modules or classes. Each abstract machine (Figure 1) consists of a set of variables, invariance properties relating to those variables and operations. The state of the system, i.e. the set of variable values, is only modifiable by operations which must preserve the invariant.

To express the post-conditions of operations, B provides the *generalised substitutions*, which can be used to specify the non-determinism (at abstract specification level) and also the determinism (at implementation specification level). This point is a notable difference with respect to Z and VDM, which use only logic expressions. The gener-

[1] http:://www.cs.york.ac.uk/puml/

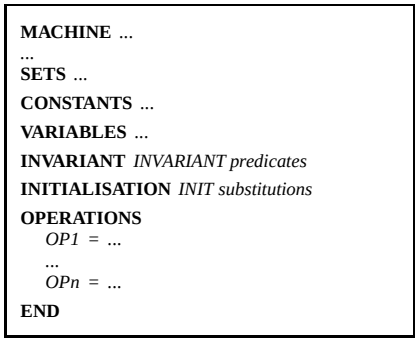

Figure 1. The model of an abstract machine

alised substitutions provide a more familiar frame to specifiers by integrating the essential methodological aspects like invariant and refinement.

Refinement is the term given to the process of taking a specification through a sequence of design steps towards implementation. In general, a distinction is frequently made between procedural or algorithmic refinement, in which only the algorithmic component of an operation is refined. The other form of refinement is data refinement, in which the state of the machine is changed, that is, a net set of variables is chosen to model the behaviour. The B method supports both procedural and data refinements. In this paper, we show a way to use the B procedural refinement in modelling UML behavioral concepts. We have also discovered the similitude between the B data refinement and the so-called "model refinement" in the Catalysis approach [8].

At every stage of the specification, proof obligations ensure that operations preserve the system invariant. A set of proof obligations that is sufficient for correctness must be discharged when a refinement is postulated between two B components. Hence, by supporting proved refinement, B allows to go progressively from an abstract specification (non deterministic) to a deterministic specification that can be translated into a programming language (ADA, C or C++).

Another characteristic of the B method is that it have been designed to be easily automated. The generation of proof obligations (of the invariant preservation and of refinement correctness) obeys the simple rules that can be easily implemented in a piece of software. Furthermore, the support tools like AtelierB and B-Toolkit provide utilities to discharge automatically and interactively the generated proof obligations. Analysing the non-discharged proof obligations with the B support tools is an efficient and practical way to detect errors encountered during the specification development.

Finally, beside the refinement, B provide also structuring primitives like **INCLUDES**, **IMPORTS**, **USES** and **SEES** so that abstract machines can be composed in various ways. Thus, large systems can be specified in a modular way, possibly reusing parts of other specifications.

3. Translating UML diagrams into B

Generally speaking, the transformation from UML specifications into B is carried out using derivation schemes and derivation procedures. The derivation schemes are defined generically for UML concepts like class, attribute, class operation etc. The derivation procedures show the way to integrate different types of UML diagrams into one B specification.

3.1. Modelling UML concepts in B

Meyer and Souquières [16] and Nguyen [17], based on the previous work of Lano [10], have proposed the derivation schemes from UML structural concepts into B. Each class, attribute, association and state is modelled as a B variable. The properties of those concepts like the cardinality or additional constraints are modelled as B invariants. The inheritance relationship between classes is also modelled as b B invariant predicate between B variables for the classes in question. It appears that all the expected semantics of UML structural elements are totally expressed using B notations if those semantics can be originally expressed in OCL (cf. Section 4.3).

In [11, 13, 14] we have proposed approaches for modelling UML behavioural concepts. Each UML behavioral concept - *use case*, *class operation*, *event* - is firstly modelled by a B operation in which the expected effects of such a concept on related data is specified directly on the derived data. The B operation for use cases, class operations and events may be refined afterward as detailed in the following sub-sections.

3.2. From use case diagrams to B specifications

In [11] we have presented an approach for building B specifications from use-case models. Each use case is modelled as a B operation. To express in B the pre- and post-conditions of use cases, we propose modelling each use case and its involved classes in the same abstract machine[2]. For each use case having included and extended use cases, its B operation is implemented or refined by making invocations to B operations of the included and extended use cases.

By structuring use cases [7], we can organise use cases into levels. The use cases at level one corresponds to "user-goal" use cases. The use cases, which are the included use

[2]Otherwise, if we distribute data derived from different classes into different abstract machines, then by technical restriction of B, it is diffi cult even impossible to express in B the pre- and post conditions of use cases involving several classes.

cases of the ones at level one, are said at level two and so on. The bottom level of use cases is composed of basic operations of classes. It is to be noted that a use case can be at several levels. We derive for each use case level an abstract machine whose operations model use cases at that level and whose data are derived from all classes. The B variables derived from the class diagram are initialised (clause **INITIALISATION**) according to the object diagram. The abstract machine for one level is implemented (refined) by importing (including) the abstract machine in the next lower level (if any). The abstract machine in the bottom level are decomposed into abstract machines for classes and associations in the class diagrams.

3.3. From realization diagrams to B specifications

In [13] we have proposed a general approach for modelling class operations. We model each class operation as a B operation in an abstract machine. As for use case, we group the class operation and its involved data in the same abstract machine. In addition, we use the calling-called class operation dependency to arrange derived B operations into abstract machines. A calling-called pair relates a class operation - the calling operation - to one of its realization class operations - the called operation. We determine the calling-called class operation dependency from interaction diagrams and activity diagrams, which are often used to represent the realization of class operations (cf. chapters 19 and 27 in [4]). The B operation of the called operation is called in the implementation or refinement of the B operation of the calling operation. That means : (i) the abstract machine for the called operation is imported (included) in the implementation (refinement) of the abstract machine for the calling operation and (ii) we use B implementation (refinement) operation to model the realization of class operations.

Given a class diagram and realization (interaction or activity) diagrams for certain class operations, we have proposed a derivation procedure as followed :

1. establishing the calling-called class operation dependency from realization diagrams ;
2. if there is no circular calling-called class operation dependency[3], we are able to arrange class operations into layers (using two procedures : "division" and "duplication" [13]) such that :
   (a) there is no calling-called dependency among operations in the same layer ;
   (b) the basic operations, which do not have any called operation, are in the bottom layer ;
   (c) the system operations, which do not have any calling operation, are in the top layer ;
   (d) the operations in a layer differing from the bottom layer only have called operations in the next lower layer. For this purpose, certain operations are eventually duplicated in several layers by using the "dummy-promoting" procedure ;
3. after the allocation,
   (a) each layer gives rise to an abstract machine in which the B operations model the class operation in the associated layer[4] ;
   (b) an abstract machine that does not belong to the bottom layer, is implemented by importing the abstract machine for the next lower layer ;
   (c) the abstract machine for the bottom layer into abstract machines for classes and their associations (if any).

3.4. From state-chart diagrams to B specifications

Our approach to model events [14] is in two stages :

1. creating a B abstract operation for each event. In the B abstract operation, the expected effects of the event is directly specified on the data derived from class data related by the event. Consequently, an event and its related data are modelled in the same abstract machine ;
2. implementing (or refining) the B operation in the first step by calling B operations for the triggered transition and actions.

We have extended the approach above to deal with the eventual deferred events as well as eventual asynchronous communication between state-charts. From those modelling, a derivation procedure has been proposed to build a B specification given a set of classes and their state-chart diagrams :

1. creating an abstract machine $System$ to model the events that are sent from the outside to state-charts or the events that are sent asynchronously between state-charts. The data in $System$ are mainly derived from class diagrams and states. However, if there is any deferred event or asynchronous communication between state-charts, then there should be extra data for those eventual concepts ;

[3] This is still an open issue due to technical restrictions of the B language.

[4] Remember that, the data in each abstract machine are derived from the whole class diagram and they are initialised according to the object diagrams

2. creating an abstract machine $Basic$. The data in $Basic$ are identical to data in $System$. The operations in $Basic$ model transitions and actions in state-charts. In the case having data for deferred events or asynchronous communication between state-charts, there should be auxiliary operations on those data. In case using the implementation construct, there should be operations for state-checking and guard conditions ;

3. decomposing $Basic$ into abstract machines for classes and eventual machines for associations between classes ;

4. implementing (or refining) $System$ by importing (or including) $Basic$.

4. Concluding remarks

Our results provide a complete framework for deriving B specifications from UML structure and behavioral diagrams. The translation from UML diagrams into a B specification is done in a systematic way. Different UML views are combined, giving a single specification. The derivation schemes are sufficiently generic to apply for requirements, analysis, design or even implementation models. This point justifies partially our choice of B as the formal notation to integrate UML and formal specification techniques. Another arguments for the choice of B come from B powerful support tools which have been recognised in industrial projects for their capabilities to generate (automatically) and to prove (automatically and interactively) the proof obligations.

4.1. B-based rigorous analysis on UML specifications

In [12], we have pointed out several kinds of analysis on UML specifications, thanks to B support tools : `(i)` the consistency of the class invariant ; `(ii)` the conformity of object and state-chart diagrams regarding the class diagrams ; `(iii)` the conformity of class operations, use cases regarding the class invariant ; `(iv)` the class operation calling-called dependency and `(v)` the use case structuring.

4.2. Modelling the UML refinement dependency in B

Wills and D'Souza [8] have introduced three refinement types : *operation refinement*, *model refinement*, and *action refinement*. The *action refinement* can be compared with the structuring of use cases, so the proposal for modelling use cases in Section 3.2 can be applied to model in B the *action refinement*. The approach for modelling class operations in Section 3.3 is totally appropriate for *operation refinement*. The *model refinement* can be compared with the data refinement in B. Hence B provides a good support to formalise the refinement dependency in UML.

4.3. Ongoing work

The translation of OCL expressions into B are being investigated. That enables the generation of B abstract operations for behavioural concepts as well as the generation of B expressions for supplementary class invariants and guard conditions, which are supposed to be specified using OCL in UML specifications. The rules for generating B refinement operations in derived B specifications are also considered. In addition, the prototype Argo/UML+B to implement the UML-B derivation schemes is currently developed.

References

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

[2] A. Amelot and D. Dollé. Le raffinement automatique. Available at http://www3.inrets.fr/B@INRETS/Events/2001-ESTAS/actes/MTI-2001-ESTAS.*, 2001. Slides.

[3] B-Core(UK) Ltd, Oxford (UK). *B-Toolkit User's Manual*, 1996. Release 3.2.

[4] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unifi ed Modeling Language User Guide*. Addison-Wesley, 1998. ISBN 0-201-57168-4.

[5] J.M. Bruel. Integrating Formal and Informal Specification Techniques. Why? How? In *the 2nd IEEE Workshop on Industrial-Strength Formal Specifi cation Techniques*, pages 50–57, Boca Raton, Florida (USA), 1998. Available at http://www.univ-pau.fr/∼bruel/publications.html.

[6] J.M. Bruel, J. Lilius, A. Moreira, and R.B. France. Defining Precise Semantics for UML. In *Object-Oriented Technology*, LNCS 1964, pages 113–122, Sophia Antipolis and Cannes (F), June 12-16, 2000. ECOOP 2000 Workshop Reader.

[7] A. Cockburn. Structuring Use Cases with Goals. http://members.aol.com/acockburn/papers/usecases.htm, 1997.

[8] D. F. D'Souza and A. C. Wills. *OBJECTS, COMPONENTS AND FRAMEWORKS WITH UML : The Catalysis Approach*. Addison-Wesley, 1998.

[9] R. Laleau and A. Mammar. A Generic Process to Refine a B Specification into a Relational Database Implementation. In *ZB 2000: Formal Specifi cation and Development in Z and B*, LNCS 1878, York (UK), August/September 2000. Springer.

[10] K. Lano. *The B Language and Method : A Guide to Practical Formal Development*. FACIT. Springer-Verlag, 1996. ISBN 3-540-76033-4.

[11] H. Ledang. Des cas d'utilisation à une spécification B. In *Journées AFADL'2001 : Approches Formelles dans l'Assistance au Développement de Logiciels*, Nancy (F), 11-13 juin, 2001. http://www.loria.fr/∼ledang/publications/afadl01.ps.gz.

[12] H. Ledang and J. Souquières. Formalizing UML Behavioral Diagrams with B. In *the Tenth OOPSLA Workshop on Behavioral Semantics: Back to Basics*, Tampa Bay, Florida (USA), October 15, 2001. http://www.loria.fr/∼ledang/publications/oopsla01.ps.gz.

[13] H. Ledang and J. Souquières. Modeling Class Operations in B: Application to UML Behaviral Diagrams. In *ASE2001: the 16th IEEE International Conference on Automated Software Engineering*, pages 289–296, Loews Coronado Bay, San Diego (USA), November 26-29, 2001. IEEE Computer Society. http://www.loria.fr/∼ledang/publications/ase01.ps.gz.

[14] H. Ledang and J. Souquières. Contributions for Modelling UML State-Charts in B. In *IFM 2002: Third International Conference on Integrated Formal Methods*, LNCS 2335, pages 109–127, Turku (Fin), May 15-17, 2002. Springer-Verlag. http://www.loria.fr/∼ledang/publications/ifm2002.ps.gz.

[15] E. Meyer. *Développements formels par objets: utilisation conjointe de B et d'UML*. PhD thesis, LORIA - Université Nancy 2, Nancy (F), mars 2001.

[16] E. Meyer and J. Souquières. A systematic approach to transform OMT diagrams to a B specification. In *FM'99 : World Congress on Formal Methods in the Development of Computing Systems*, LNCS 1708, Toulouse (F), September 1999. Springer-Verlag.

[17] H.P. Nguyen. *Dérivation de spécifi cations formelles B à partir de spécifi cations semi-formelles*. PhD thesis, Conservatoire National des Arts et Métiers - CEDRIC, Paris (F), décembre 1998.

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

[19] C. Snook and M. Butler. Verifying Dynamic Properties of UML Models by Translation to the B Language and Toolkit. Technical Report DSSE-TR-2000-12, Declarative Systems & Software Engineering Group, Department of Electronics and Computer Science University of Southampton, September 2000. Available at http://www.dsse.ecs.soton.ac.uk/techreports/2000-12.html.

[20] C. Snook and R. Harrison. Practitioners Views on the Use of Formal Methods: An Industrial Survey by Structured Interview. *Information and Software Technology*, 43:275–283, March 2001.

[21] STERIA - Technologies de l'Information, Aix-en-Provence (F). *Atelier B, Manuel Utilisateur*, 1998. Version 3.5.