Formal techniques in the object-oriented software development: an approach based on the B method

Hung Ledang

Formal techniques in the object-oriented software development: an approach based on the B method

Hung LEDANG

LORIA - Université Nancy 2 - UMR 7503
Campus scientifique - BP 239
54506 Vandœuvre-lès-Nancy Cedex - France
E-mail: ledang@loria.fr

Abstract

We address the problem of modeling in B object-oriented specifications. The contribution of this research is to give a way to a formal verification of object-oriented specifications by analyzing the corresponding B specifications. This is significant where B support tools are available. We can also use object-oriented specifications as tools for building B specifications. Thus, an approach for a practical and rigorous software development, which is based on object and B from the requirements elicitation to the executable code, is proposed.

Keywords: *object-oriented specification, class operation, B method, B abstract machine, B operation.*

1. Introduction

1.1. Combination of object and formal techniques in an unified software development process

One of goals of the DEDALE project[1] is to develop a practical and rigorous software development process. Object-oriented techniques are practically considered as quite popular techniques in software industry. However, in order to assure the rigorousness for the software development process, we often use formal techniques. A so-called B[1] is a formal software development method that covers software process from the abstract specification to the executable implementation. The reasons for this selection are: (i) B ensures that the code satisfies its specification; (ii) it has several support tools (AtelierB [16], B-Toolkit [2])and (iii) it is well adapted with large scale industrial projects [3]. In order to have practical and rigorous goals together, we consider the combination of object and B techniques.
An appropriate combination of object-oriented techniques and the B technique can give a way that is applicable in the industry. For this objective, we propose to integrate these two formalisms, i.e derivation schemes from object concepts into B notations are proposed. This object-B integration has following advantages: (i) the construction of object-oriented specifications is formally controlled; (ii) the construction of B specifications becomes easier with the help of object-oriented specifications. From the informal description of requirements, we successively build the object models with different degrees of abstraction. These models cover from conceptual models through logical design models to the implementation models of the software. This also means that the built models are successively refined. We verify the consistency of each object model by analyzing the derived B specification. We verify the conformance among object models by analyzing the refinement dependency among them that is formally expressed in B.

[1] http://www.loria.fr/equipes/dedale/

1.2. 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 generalized substitutions and first order logic. Specifications are composed of abstract machines that are similar to modules or classes. They consist 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. Machines can be composed in various ways. Thus, large systems can be specified in a modular way, possibly reusing parts of other specifications. Refinement of a B model allows developers to derive a correct implementation in a systematic way. Refinement can be seen as an implementation technique but also as a specification technique to progressively augment a specification with more details. 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 machines.

1.3. Integrating objects and B: state of the art

In [12, 14], Meyer and Nguyen have proposed a set of precise rules for modeling in B the concepts of static aspects of a system such as class, attribute, association and inheritance. These rules are formally defined and can be implemented in a piece of software; otherwise the rules for formalizing concepts of behavioral aspects must be ameliorated due to their restrictions and their ambiguities. As an example, the existing rules [12, 14, 13, 15] cannot deal with class operations, which concern several classes. Although the current research works have some advantages but they only use B abstract machine and B inclusion mechanism to model object concepts. Therefore we cannot model the calling-called dependency among class operations. Moreover, the explicit distributing class data into different B abstract machines prevents us from modeling the effects of operations, which concern data from several classes.

1.4. Thesis objectives

Our work is a complementary of Meyer's thesis [12]. Its primary goal is to augment rules for object-B transformations. We emphasize on modeling behavioral aspects of object-oriented specifications which are described in UML diagrams (use case diagrams, collaboration diagrams, etc.). The second goal of this thesis is to study the combination of the refinement of object and B. Therefore, a rigorous process based on objects and B for software development can be achieved. In addition, the support tool for automatic derivation from UML notations into B [12], is extended to take into account the new object-B derivation schemes.

2. Our obtained result and current research works

2.1. Modeling of use cases

In [9] we have presented an approach for building B specifications from use-case models, which are utilized to express functional requirements of a considered software. Our approach is based on the structure of use cases [5] and the complementary between the use-case model and the class model [8]. The B specification for a use case model consists of B abstract machines which their operations correspond to use cases. The machines derived from classes and associations are used to implement the machines, which are generated from use cases. We treat use cases from users' point of view where use cases are independent and non-conflict each other. It is also noticed that structure of use cases has not any relation with the design of use cases, which is realized by collaboration diagrams. The collaboration diagram modeling is discussed in the following section.

2.2. Modeling of class operations

An appropriate proposal for modeling of class operations allows us to model interaction diagrams such as collaboration diagrams or sequence diagrams. In [10] we have proposed a general approach for modeling all class operations. Our approach differs from previous works by : (i) grouping an operation and its concerned data in the same B abstract machine; (ii) separating a calling operation from its called operations in different B abstract machines and (iii) using B implementation construct and B importation mechanism for modeling the calling-called dependency among operations. Two procedures have been proposed to allocate the class operations into layers. The **division procedure** divides the set of class operations into layers such that operations in the same layer are independent and relying on operations in lower layers (if any). The **"dummy-promoting" procedure** duplicates several operations in several layers so that an operation depends only on operations in the next lower layer (if any). The division and the "dummy-promoting" procedures with existing rules for modeling class diagrams are used in another procedure that provides a generic framework for building the B specification of a component from its object-oriented specification. Each obtained operation layer from the division and "dummy-promoting" procedures generates a B abstract machine. A machine that does not belong to the bottom layer, is implemented by importing the machine for the next lower layer. The generated machine of the lowest layer is decomposed into machines for classes and their non-fixed associations.

2.3. Generating the content of B operations

As noticed in [10], at present we can only automatically derive the architecture of B specifications from object-oriented specifications. The data, the skeleton of B operations in the B specification are also automatically derived. In order to complete B specifications, we must fill up the body of B operations. For the purposes of a complete automation of transformation, we envisage generating the content of B operations from class and collaboration diagrams. The derivation schemes in [12] are appropriate to generate B operations of basic operations such as constructor or destructor of a class or the operations that modifies or queries the value of class attributes. For each remaining class operation, which calls several other operations, we generate : (i) a B operation specifying the pre-/post conditions of the operation and (ii) the implementation of this B operation. Especially, we model the operation call from considered operation to its called operations.
We also have envisaged two solutions : (i) we propose embedding B notations in the description of class operations in class and collaboration diagrams. Thus the task of generating the content of B operations becomes re-copying descriptions of operations from object-oriented specifications into B specifications; (ii) we try to directly calculate the content of B operations in B abstract machines and in implementations from class and collaboration diagrams. This calculation is done with the existing B operations' content of basic class operations.

2.4. Modeling refinement constraints between two object models

Figure 1 presents an example of the refinement relation between two object design models. In this figure the package "`Package1`" is refined by the package "`Package2`". Both packages are supposed to contain an object-oriented specification of the same system or the same component at two abstract degrees. The refinement link is stipulated by a dependency relation between two packages. This relation is bounded with a constraint which expresses the gluing invariant of the refinement dependency.
We derive one B abstract machine for each package : the machine *Package1* for the package `Package1`, the machine *Package2* for the package `Package2`. Each machine only models class operations that are system operations [6]. There are two solutions for modeling the conformance of `Package2` in comparison with `Package1`. The first solution (as showed in Figure 1) is to consider that the B abstract

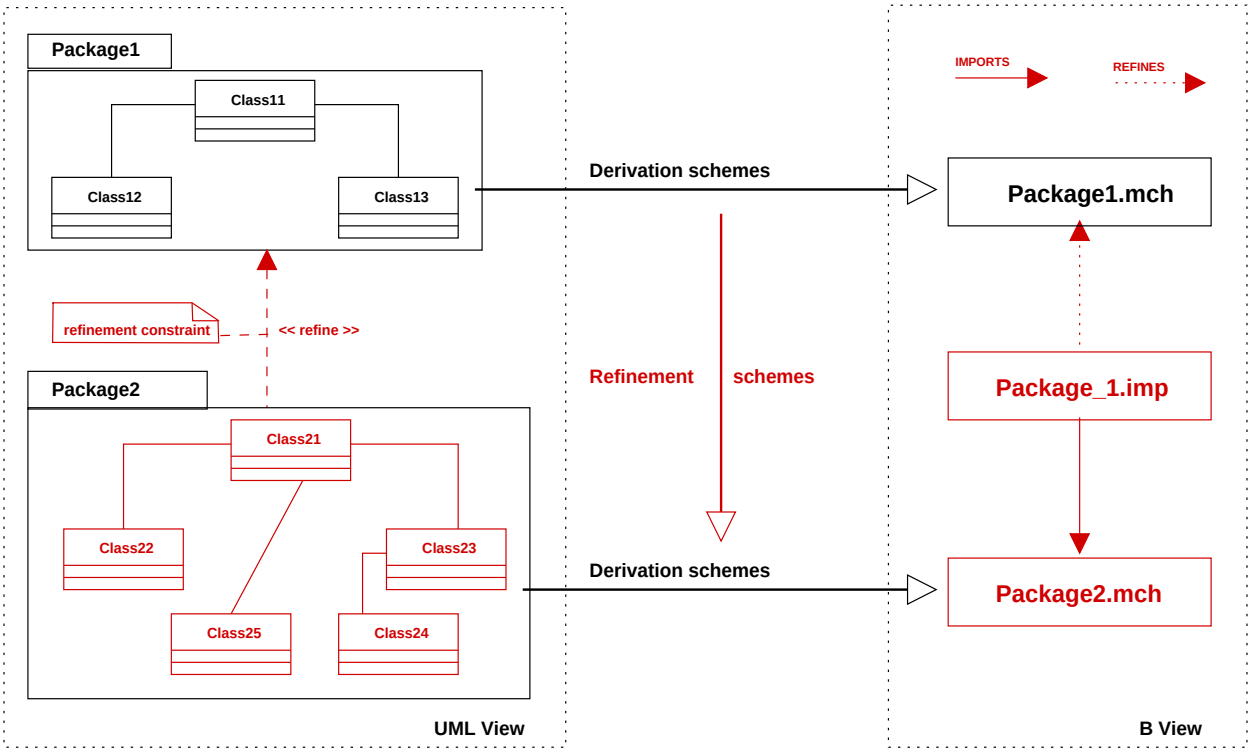

Figure 1. Graphic visualisation of refinement

machine *Package2* is used to implement the machine *Package1*. The refinement constraints between `Package1` and `Package2` are modeled as gluing invariant in the implementation of *Package1*. Another solution is to consider *Package2* as a refinement of *Package1*. In this case we create *Package2* as a refinement instead of an abstract machine. We prefer the first solution because it is more straightforward if we have several levels of abstraction; in addition, it shares some points with modeling approaches for use cases and class operations in Sections 2.1 and 2.2

3. Discussions

3.1. Methodology for evaluation of the proposed approaches

We validate our proposals by non trivial case studies. The work in [9] has been experimented with a case study on a controlling system for accessibility of buildings [11]. The work in [10] has been experimented with the pump component of a controlling system for petrol dispensing, customer payment handling and petrol tank level monitoring [6].

3.2. Modeling of class dependency

According to Booch et al. [4], a dependency between two classes `Class1` and `Class2` states that we use `Class2` in the description of operations of `Class1` or as an attribute type of `Class1`. Our approach for modeling class operations in Section 2.2 means that the first case of class dependency is solved. The second type of the class dependency has been modeled by the formalizing rules of Meyer [12]. So the problem of modeling the class dependency is handled.

3.3. Modeling of refinement

Wills and D'Souza [7] have introduced four refinement types : operation refinement, model refinement, object refinement and action refinement. The refinement relation discussed in Section 2.4 corresponds to the model and operation refinements. The action refinement can be compared with the structuring of use cases, so our approach for modeling use cases in Section 2.1 can be applied for modeling the action refinement. Hence, the object refinement, which composes all action refinement, model refinement, object collaboration and the operation refinement, can also be modeled by an appropriate combination of approaches described in Sections 2.1, 2.2 and 2.4.

3.4. concluding remarks

The contribution of this work is that it proposes a framework for software development based on objects and B. The approach for use case modeling in [9] gives a possibility to use B tools for the formal verification and analysis of object-oriented requirements models. Our approach for class operation modeling in [10] combined with object-B transformation rules of Meyer [12] also gives a possibility to use B tools for the formal verification and analysis of the object-oriented design models.

References

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

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

[3] P. Behm, P. Desforges, and J.-M. Meynadier. MÉTÉOR: An Industrial Success in Formal Development, April 1998. An invited talk at the 2nd Int. B conference, LNCS 1939.

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

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

[6] D. Coleman, P. Arnold, St. Bodoff, Ch. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development : The Fusion Method*. Prentice Hall, 1994.

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

[8] M. Glinz. A Lightweight Approach to Consistency of Scenarios and Class Models. In *Proceedings of the Fourth International Conference on Requirements Engineering*, Illinois (USA), June 10-23, 2000. Available at http://www.ifi .unizh.ch:80/groups/req/staff/glinz/activities.html.

[9] H. Ledang. Des cas d'utilisation à une spécifi cation B. In *Journées AFADL'2001 : Approches Formelles dans l'Assistance au Développement de Logiciels*, Nancy (F), june 11-13, 2001.

[10] H. Ledang and J. Souquières. Modeling class operations in B : a case study on the pump component. Technical Report A01-R-011, Laboratoire Lorrain de Recherche en Informatique et ses Applications, March 2001. Available at http://www.loria.fr/∼ledang/publications/UML01.ps.Z.

[11] Y. Ledru, G. Padiou, and J. Jaray. Étude de cas: Système de contrôle d'accès. http://www-lsr.imag.fr/afadl2000/EtudeDeCas/, 2000.

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

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

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

[15] E. Sekerinski. Graphical Design of Reactive Systems. In D. Bert, editor, *B'98: Recent Advances in the Development and Use of the B Method - 2nd International B Conference*, LNCS 1393, Montpellier (F), April 1998. Springer-Verlag.

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