HAL
open science

Integrating UML and B Specification Techniques

Hung Ledang, Jeanine Souquières

To cite this version:

Hung Ledang, Jeanine Souquières. Integrating UML and B Specification Techniques. The Informatik 2001 Workshop on Integrating Diagrammatic and Formal Specification Techniques, Sep 2001, Vienna, Austria, 8 p. inria-00107870

HAL Id: inria-00107870

https://inria.hal.science/inria-00107870v1

Submitted on 19 Oct 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Integrating UML and B Specification Techniques

Hung LEDANG Jeanine SOUQUIÈRES

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

Abstract

An appropriate approach for integrating UML and B specification techniques allows us to map UML specifications into B specifications. Therefore, we can formally analyze an UML specification via the corresponding B formal specification. This point is significant because B support tools are available. We can also use UML specifications as a tool 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.

In this paper, we address the problem of modeling UML behavioral diagrams in B, which is up to now an open issue. For this purpose, an approach for modeling in B class operations is proposed. We show a way to apply this approach for integrating collaboration diagrams into B specifications.

Keywords: *UML, class operation, B method, B abstract machine(BAM), B operation.*

1 Introduction

The Unified Modeling Language (UML)[16] 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, the fact that UML lacks a precise semantics is a serious drawback of object-oriented techniques based on UML.

On the other hand, B[1] is a formal software development method that covers software process from the abstract specification to the executable implementation. A strong point of B (over other formal methods) is support tools like AtelierB [20], B-Toolkit [2]. Most theoretical aspects of the method, such as the formulation of proof obligations, are done automatically by tools. Provers are also designed to run automatically and reference a large library of mathematical rules, provided with the system. All of these points make B been adapted in large scale industrial projects [3]. However, as a formal method, B is still difficult to learn and to use.

As cited many times in the literature [8, 13, 15, 18, 19], an appropriate combination of object-oriented techniques and formal methods can give a way that is applicable in the software industry. For this objective, we advocate integrating object-oriented and B techniques. Our approach is to propose derivation schemes from UML concepts into B notations. This UML-B integration has following advantages: (i) the construction of object-oriented specifications based on UML is formally controlled; (ii) the construction of B specifications becomes easier with the help of UML-based 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 developed 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.

At the present, we only consider the modeling of UML concepts in B. The problem of analyzing the derived B specification remains at a later stage. The works in [6, 13, 14, 15, 10, 17] presented a set of rules for mapping UML static diagrams into B. Certain elements in UML behavioral diagrams like state and transition have been partially treated. So far, the problem of modeling UML behavioral diagrams in B has been an open issue. In [5], Laleau and Mammar have presented a support tool for generating B specifications from UML diagrams of data intensive applications. Although they considered UML

collaboration diagrams, nothing new is added with respect to Nguyen's work [15]. The main reason is that existing works coincide the UML class with the BAM concept, but in fact, they do not coincide each with other. A class operation can affect the data from different classes but a B operation affects only data declared in the same BAM. For this reason, only basic class operations, which are local to classes, can be modeled. We cannot model non-basic class operations that concern several classes.

In this paper, we present an approach for integrating behavioral diagrams into B. The paper is structured as follows: the remainder of this section presents an example which is used throughout the presentation. In Section 2, a brief introduction of the B method and a brief discussion of related work are presented. In Section 3 we present an approach for modeling class operations into B. In Section 4 we go on to discuss the way to integrate UML collaboration diagrams into B. Finally, in Section 5, concluding remarks complete our presentation.

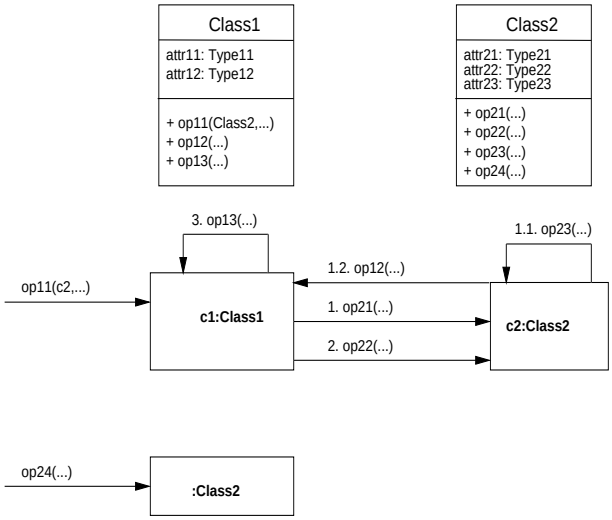

Figure 1: An UML specification

1.1 Example

Given an UML specification as described in Figure 1. There are two classes `Class1` and `Class2`. In each class we declare several attributes: `attr11` of type `Type11` and `attr12` of type `Type12` in `Class1`; `attr21`, `attr22` and `attr23` in `Class2`. We omit here the associations between two classes. There are three class operations in `Class1`: `op11` has an argument of type `Class2`, whereas `op12` and `op13` are local in `Class1`. In `Class2` there are four operations : `op21` and `op22` are called in the realization of `op11` of `Class1`; `op23` is called in the realization of `op21`. In addition, we also call `op13` of `Class1` in the realization of `op21`. The operation `op24` is local in `Class2` but is not called by any other operation. The dependency amongst class operations is shown in the collaboration diagrams associated to classes.

2 Integrating UML and B

2.1 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 BAMs 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. BAMs 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 B components.

2.2 Integrating UML and B: state of the art

In [13, 15], 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. Figure 2 shows a BAM and its data which are derived from the class `Class1` in Figure 1. As shown in Figure 2, the class `Class1` is formally derived by a BAM *Class1*. In *Class1* we declare a B deferred set *CLASS1*, which models the set of possible instances of the class `Class1`. The set of the effective instances of the class `Class1` is modeled by a B variable *class1* constrained to be a subset of *CLASS1*. For each attribute `attr1i`, a B variable *attr1i* is created and defined in the INVARIANT clause as a function from the B set *class1* into the B set *Type1i* modeling its associated type `Type1i`.

If the rules for modeling concepts of data aspects are formally defined and can be implemented in a piece of software, the rules for formalizing concepts of behavioral aspects must be intensively done. The main reason is that they have no appropriate solution for dealing with class

```
MACHINE Class1
SETS
   CLASS1; Type11; Type12
VARIABLES
   class1,attr11,attr12
INVARIANT
   class1 ⊆ CLASS1 ∧
   attr11 ∈ class1 ⟶ Type11 ∧
   attr12 ∈ class1 ⟶ Type12
...
END
```

Figure 2: A B representation for data of the class `Class1`

operations, which is the core concept in the behavioral diagrams. In fact, with existing rules, we cannot, in general, model class operation affecting data of several classes. In addition, the realization of non-basic class operations is also glossed over. Consider the modeling of the operation `op11` of the class `Class1`. If we model `op11` by a B operation *op11* in the BAM *Class1* (as suggested in [13, 15]), then to model the fact that `op11` has an argument of the type `Class2`, we must include the BAM *Class2* in the BAM *Class1*. In this case, the fact that the realization of `op11` calls two operations `op21` and `op22` in `Class2` cannot be modeled due to technical restrictions of B inclusion mechanism[1, 20].

3 Modeling class operations in B

3.1 The calling-called dependency

In the realization of a class operation (often represented by collaboration or activity diagrams) we make some calls to other class operations. A calling-called pair relates a class operation to one of its realization class operations. In the example (Figure 1) `op11` and `op21` form a calling-called pair where `op11` acts the "calling" role and `op21` acts the "called" role. We say `op11` is a "calling" operation of `op21` and `op21` is a "called" operation of `op11`.

3.2 Grouping data and operation in the same BAM

By grouping a class operation and its related data in the same BAM, the problem of modeling class operations becomes one of how B substitutions can be used to express the pre-/post condition of the operation. This is similar to model basic operations as described in [13]. Figure 3 shows a BAM *MachineA* which contains the B operation *op11* corresponding to the class operation `op11`; in the data declaration section (clauses SETS, VARIABLES and INVARIANT) of *MachineA* we notice the presence of the data which are derived from classes `Class1` and `Class2`.

```
MACHINE MachineA
SETS
   CLASS1; Type11; Type12; CLASS2;Type21;Type22;Type23
VARIABLES
   class1,attr11,attr12,class2,attr21,attr22,attr23
INVARIANT
   class1 ⊆ CLASS1 ∧
   class2 ⊆ CLASS2 ∧
   attr11 ∈ class1 ⟶ Type11 ∧
   attr12 ∈ class1 ⟶ Type12 ∧ ...
...
OPERATIONS
   ...
   op11(cc1,cc2,...) =
   pre
      cc1 ∈ class1 ∧
      cc2 ∈ class2 ∧ ...
   then
      /* modeling the effect of the non-basic operation op11, this time, is
      similar to modeling the effect of the basic operation op23 */
   end
END
```

Figure 3: Modeling of the operation `op11`

We may create a BAM for the whole set of classes of a component's object-oriented specification. Data of the created BAM are derived from the whole class diagram and the operations are all class operations. However, grouping all the class operations in the same BAM prevents us from modeling the calling-called dependency between class operations; for example, if the operation `op13` is modeled in the same BAM as `op11` then we are not able to express the calling-called dependency amongst them; this is because a B operation cannot call another one in the same BAM [1, 20]. In other words, we must create several BAMs for class operations in order to model the calling-called dependency amongst them. The following sections discuss how to allocate the class operations in BAMs.

3.3 Modeling the calling-called dependency amongst class operations

The intuitive idea is to separate a calling operation from its called operations; if an operation `OpA` calls another operation `OpB` then `OpA` and `OpB` are modeled in two different BAMs which we call *MachineA* and *MachineB*. In the implementation of *MachineA* we import *MachineB* so we make an invocation to B operation *OpB* in the implementation of the B operation *OpA*; in the case of neither `OpA` nor `OpB` calling the other, they are independent and we can model them either in the same BAM or in two BAMs;

```
IMPLEMENTATION MachineA_imp
REFINES MachineA
/* We implement the data in MachineA by the data in
   MachineB. But these two data set are identical.
   That is why MachineB is renamed. */
IMPORTS im.MachineB
INVARIANT
   class1 = im.class1 ∧
   class2 = im.class2 ∧
   attr11 = im.attr11 ∧...
...
OPERATIONS
      ...
      op11(cc1,cc2,...) =
      begin
          /* Each method invocation in collaboration diagrams
             is modeled as a B operation invocation. */
          im.op21(...);
          im.op22(...);
          im.op13(...);
    end
END

MACHINE MachineB
...
/* The data of MachineB are identical to the data of MachineA
   because they are derived from the same class diagram. */
SETS
   CLASS1;CLASS2; ...
VARIABLES
   class1,class2,attr11,...
INVARIANT
   class1 ⊆ CLASS1 ∧ ...
OPERATIONS
    op21(...) = ...
    op22(...) = ...
    op13(...) = ...
END
```

Figure 4: Modeling of the calling-called dependency amongst class operations

if they come from the same class, it is recommended to group them in the same BAM (this is the case for basic operations of a class).

In Figure 4 the BAM *MachineA* of Figure 3 is implemented in the implementation *MachineA_imp* which imports the BAM *MachineB*. In *MachineB*, we model operations `op13`, `op21` and `op22`, which are called in the realization of `op11` (Figure 1). As we can see, both *MachineA* and *MachineB* contain some data with the same name and properties; this is because they are all derived from the same class data concerned by `op11`. The INVARIANT clause in *MachineA_imp* must assert this remark. In addition, we must rename *MachineB* in the IMPORTS clause in order to clearly distinguish the data with the same name in *MachineA* and *MachineB* in the INVARIANT clause of *MachineA_imp*. Several remarks should be made:

- our approach for modeling the calling-called dependency relation amongst class operations is only appropriate if there is no cyclic calling-called dependency among class operations. Consider three class operations $\mathtt{Op}_1$, $\mathtt{Op}_2$ and $\mathtt{Op}_3$. Assume that: $\mathtt{Op}_1$ calls $\mathtt{Op}_2$; $\mathtt{Op}_2$ calls $\mathtt{Op}_3$ and $\mathtt{Op}_3$ calls $\mathtt{Op}_1$. So the BAM for $\mathtt{Op}_1$ is implemented by importing the BAM of $\mathtt{Op}_2$ which in turn is implemented by importing the BAM of $\mathtt{Op}_3$. Because $\mathtt{Op}_3$ calls $\mathtt{Op}_1$, the BAM of $\mathtt{Op}_3$ is implemented by importing the BAM of $\mathtt{Op}_1$. This situation is impossible in B [1, 20];

- there are, in general, two possibilities for modeling the calling-called dependency amongst class operations: (i) using B implementation construct and B importation mechanism and (ii) using B refinement construct and B inclusion mechanism. We prefer the first one due to the expressing capacity. In fact, in some cases we can use the B refinement/inclusion dual; this is the case, for example, when `OpA` modeled in *MachineA* which calls the `OpB` modeled in *MachineB* and calls no other operations modeled in *MachineB*; however if `OpA` also calls some other operations modeled in *MachineB* then the refinement/inclusion dual is not appropriate due to technical restrictions of the B inclusion mechanism [1, 20];

- the similar idea has been used in our previous work for modeling use cases [7]. We can also apply this approach to model events in state-chart diagrams. The B operation modeling an event is implemented by B operations modeling the triggered transition and its associated actions of the event. Our approach is "better" than the ones of Meyer and Sekerinski [14, 17] in the sense that it allows more than one actions, which are sequential, to be associated to a transition;

- however, our approach for modeling class operations only works without concurrence inside class operations. This is due to restrictions of B with respect to the implementation construct. In fact in a B implementation operation we cannot express two operation call concurrently. Dealing with concurrence will be envisaged in a later stage.

As described in [9], apart from cyclic calling-called dependency and without the concurrence inside class operations, we are able to derive the architecture of the B specifications from a UML specification comprising class and collaboration diagrams. The data in B specifications and the skeleton of B operations are also derived. These points will be detailed in the following sections.

4 Integrating collaboration diagrams and B

Given a class diagram and several collaboration diagrams for realizing certain class operations[1], this section presents the way to use collaboration diagrams in deriving B specification. As noticed in Section 3.3, we assume that there is no concurrent specification (only one thread) in all collaboration diagrams.

4.1 Generating architecture of the B specification

We use collaboration diagrams to establish the calling-called dependency amongst class operations. If there is no cyclic calling-called dependency amongst class operations, we are able to arrange class operations into layers (using two procedures "division" and "dummy-promoting" [9]) such that:

(i) there is no calling-called dependency amongst operations in the same layer;
(ii) the basic operations, which do not have any called operation, are in the bottom layer;
(iii) the system operations, which do not have any calling operation, are in the top layer;
(iv) the operations in a layer differing from the bottom layer only have called operations in the next lower layer. For this purpose, certain operation is duplicated in several layers.

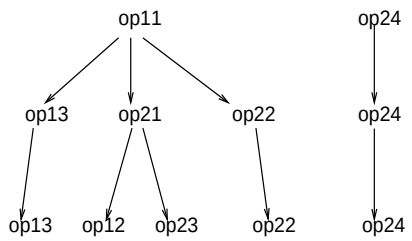

Figure 5: Layers of class operations

The class operation in Figure 1 are allocated in three layers as shown in Figure 5. The operations `op12`, `op13`, `op22`, `op23` and `op24` are in the bottom layer because they have no called operation. `op11`, having no calling operation, is in the top layer. `op21`, which is the called operation of `op11` and also the calling operation of `op23` and `op12`, is in the intermediate layer. Because `op11` depends on `op22` and `op13`, the two later ones are duplicated in the intermediate layer. In addition, `op24` must be duplicated in the top and the intermediate layer because it has no calling operation (that means that this is a system operation in the UML specification).

After the allocation, each layer gives rise to a BAM in which the B operations model the class operation in the associated layer[2]. From operation layers in Figure 5, we create three BAMs (Figure 6) : *System* encapsulates two operations `op11` and `op24` in the top layer; *Intermediate* for operations in the intermediate layer and *Basic* for operations in the bottom layer.

MACHINE *System*
/∗ The data of *System* are derived from the whole class diagram. ∗/
SETS
CLASS1;CLASS2;Types11...
VARIABLES
class1,attr11,attr12...
INVARIANT
$class1 \subseteq CLASS1 \wedge$
$class2 \subseteq CLASS2 \wedge$
$attr11 \in class1 \longrightarrow Type11 \wedge$
$attr12 \in class1 \longrightarrow Type12 \wedge$ ...
...
OPERATIONS
/∗ The operations of *System* come from the top layer in Figure 5. ∗/
$op11(cc1,cc2,...) = \ldots$
$op24(...) = \ldots$
END

MACHINE *Intermediate*
/∗ The data of *Intermediate* are identical to the data of *System*. ∗/
...
OPERATIONS
/∗ These operations comme from the intermediate layer. ∗/
$op21(...) = \ldots$
$op22(...) = \ldots$
$op24(...) = \ldots$
$op13(...) = \ldots$
END

MACHINE *Basic*
/∗ The data of *Basic* are also identical to the data of *System*. ∗/
...
OPERATIONS
/∗ These operations comme from the bottom layer. ∗/
$op12(...) = \ldots$
$op22(...) = \ldots$
$op23(...) = \ldots$
$op24(...) = \ldots$
$op13(...) = \ldots$
END

Figure 6: Skeleton of machines generated from operation layers in Figure 5

As described in Section 3.3, a BAM that does not belong to the bottom layer, is implemented by import-

[1] According to Booch et al. [4], we can also use activity diagrams to realize a non-basic class operation. Therefore, our approach for integrating collaboration into B can also be applied for activity diagrams.

[2] Remember that, the data in each BAM are derived from the whole class diagram.

ing the BAM for the next lower layer. In our example, the BAM *System* is implemented by importing *Intermediate*; *Intermediate* is implemented by importing *Basic*. In Figure 7, *System_imp* is the implementation of the BAM *System*. The BAM *Intermediate* is renamed in the IMPORTS clause of *System_imp*. The implementation of the B operation *op11* is made up by invocations to operations *op21*, *op22* and *op13* of the BAM *Intermediate*. This B implementation models the realization of `op11` by collaboration diagrams (Figure 1). To implement *op24* (of *System*), it is sufficient to call *op24* (of *Intermediate*).

```
IMPLEMENTATION System_imp
REFINES System
/* We use data of Intermediate to implement the data of System, but these
   two set of data are identical. This is why Intermediate is renamed. */
IMPORTS im.Intermediate
INVARIANT
   class1 = im.class1 ∧
   class2 = im.class2 ∧
   attr11 = im.attr11 ∧...
...
OPERATIONS
      /* Because op24 is duplicated in System and Intermediate, it is
       suffi cient to call im.op24 of Intermediate to implement op24 of System.*/
      op24(...) = im.op24(...)
      op11(cc1,cc2,...) =
      begin
         /* Each method invocation in UML collaboration
            diagrams is modeled as a B operation invocation. */
         im.op21(...);
         im.op22(...);
         im.op13(...);
    end
END
```

Figure 7: Specification component *System_imp*

Finally, we can decompose the BAM for the bottom layer into BAMs for classes and their non-fixed associations (if any). In our example, the BAM *Basic* is decomposed into BAMs *Class1* and *Class2*. In this case, all data and operations in *Basic* are distributed in the included BAMs (Figure 8).

4.2 Generating the content of B operations

It is easy to find that: corresponding to each non-basic class operation[3], there is a B abstract specification and a B implementation specification. The abstract content is in the BAM for the layer of the class operation and the implementation is in the corresponding implementation. The abstract content is made up by specification of the effect of class operation on the value of the manipulated objects. Whereas, in the implementation content, we model the realization of the considered class operation. Intuitively, each operation invocation in object-oriented specification is translated to a B operation invocation in B specification.

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 propose to attach to each class operation an OCL-based pre/-post specification. Hence, the abstract content of B operations can be derived by using OCL-B rules of Marcano [11]. The implementation content of B operations for non-basic class operations is derived from realization diagrams (usually collaboration or activity diagrams) of the considered operation.

```
MACHINE Basic
/* We use the clause EXTENDS to indicate that the data and operations
   of the Basic are distributed in Class1 and Class2.*/
EXTENDS
   Class1;Class2

END

MACHINE Class1
SETS
   CLASS1;Type11;Type12
VARIABLES
   class1,attr11,attr12
INVARIANT
   class1 ⊆ CLASS1 ∧
   attr11 ∈ class1 ⟶ Type11 ∧
   attr12 ∈ class1 ⟶ Type12 ∧...
...
OPERATIONS
    op12(...) = ...
    op13(...) = ...
END

MACHINE Class2
SETS
   CLASS2;Type21;Type22;Type23
VARIABLES
   class2,attr21,attr22,attr23
INVARIANT
   class2 ⊆ CLASS2 ∧
   attr21 ∈ class2 ⟶ Type21 ∧
   attr22 ∈ class2 ⟶ Type22 ∧
   attr23 ∈ class2 ⟶ Type23 ∧...
...
OPERATIONS
    op22(...) = ...
    op23(...) = ...
    op24(...) = ...
END
```

Figure 8: Decomposing *Basic* into machines for classes and non-fixed association.

[3]The class operation having a realization.

5 Conclusion

In this paper, we present an approach for modeling class operations in B. Our approach overcomes shortcomings of the existing approaches by taking into account: (i) the calling-called dependency amongst class operations and (ii) the binding between an operation and its concerned data. We also showed a way to apply our approach for modeling class operation to integrate UML collaboration and class diagrams into B specifications. Our approach is based on three procedures[9]:

division procedure, to divide class operations into layers according to the dependency amongst them;

"dummy-promoting" procedure, to modify the layered division obtained from the division procedure in order to ensure that operations in one layer differing from the bottom layer have only called operations in the next lower layer.

generic procedure, to translate UML specifications into B specifications. The generic procedure uses the division and "dummy-promoting" procedures to create the layered division of class operations. From these operation layers we automatically derive the architecture of the B specification. The data, the skeleton of B operations in the B specification are also automatically derived. It remains to fill up manually the body of B operations.

Our procedures can be implemented in a piece of software. The generic procedure can be extended to take into account state-chart and activity diagrams. Thus a complete framework for deriving B specifications from UML structure and behavior diagrams can be achieved. Hence, the conformance between two aspects (the structure and the behavior) of an UML specification can be formally verified by analyzing the corresponding B specification. This also means that we effectively have an appropriate and generalized solution for modeling in B the structure and the collaboration of design patterns which was mentioned in [13, 12] but only some typical patterns (the composite pattern, the client-server pattern) are treated.

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] R. Laleau and A. Mammar. An Overview of a Method and its support Tool for Generating B Specifications from UML Notations. In *The 15st IEEE Int. Conf. on Automated Software Engineering*, Grenoble (F), September 11-15, 2000.

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

[7] 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.

[8] H. Ledang. Formal Techniques in the Object-Oriented Development: an Approach based on the B method. In *the 11th PhDOOS Workshop: PhD Students in Object-Oriented Systems*, Budapest (H), http://www.st.informatik.tu-darmstadt.de/phdws/wstimetable.html, June 18-19, 2001.

[9] 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.

[10] A. Malioukov. An Object-Based Approach to the B Formal Method. 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.

[11] R. Marcano and N. Lévy. Transformation d'annotations OCL en expressions B. In *Journées AFADL'2001 : Approches Formelles dans l'Assistance au Développement de Logiciels*, Nancy (F), 11-13 juin, 2001.

[12] R. Marcano, E. Meyer, N. Lévy, and J. Souquières. Utilisation de patterns dans la construction de spécifications en UML et B. In *Journées AFADL'2000 : Approches Formelles dans l'Assistance au Développement de Logiciels*, janvier 2000.

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

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

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

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