An Overview of a Method and its Support Tool for Generating B Specifications from UML Notations

Régine Laleau and Amel Mammar
CEDRIC-IIE(CNAM)
18 allée Jean Rostand, 91025 Evry, France
{*laleau, mammar*}*@iie.cnam.fr*

Abstract

This paper presents, through an example, an overview of our method which generates B specifications from an application described using UML notations. We are interested in data intensive applications. This allows us to automatically generate basic update operations from class diagrams. Then these operations are combined to elaborate more complex transactions described in UML by state and collaboration diagrams. The obtained B machines are directly usable in AtelierB and proofs can be performed allowing the consistency of the application to be checked. Finally the outlines of the prototype support tool are described.

1. Introduction

UML [10] is now standardized by the OMG. In spite of real efforts, it is not sufficiently formal, specially if proof of consistency between the different components has to be achieved. B is a formal method developed by Abrial [1]. It is a complete method and supports a large segment of the development life cycle: specification, refinement and implementation. It ensures, thanks to refinement steps and proofs, that the code satisfies its specification. It has been used in significant industrial projects and commercial case tools are available in order to help the specifier during the development process. These are the main arguments to use B rather than Z (type checking and tool assistance in proof, but limited tool support for refinement and no automated proof), VDM (standard semantics, good tool support, but relatively primitive structuring of formal presentation) or another formalism.

In this framework, our objective is to provide designers with a tool that automatically derives B specifications from UML specifications, for the design of data intensive applications. The main characteristic of such systems is that they manage massive amounts of data. The complexity of the design is to ensure that these data are handled safely whereas operations that act upon are rather simple.

Direct related works are those of Dupuy[4] and Bruel[2]. They propose a method with a support tool for generating formal specifications from informal object-oriented notations. The work of Dupuy consists of translating an UML application (described with the Rose toolkit) into Z specifications. It is focused on the generation of basic operations and their preconditions from a class diagram. More complex operations are not considered. In the second work, the object-oriented method is Fusion. The method is not dedicated to a specific domain. Thus it is not possible to define generic operations. Z specification skeletons are generated, which must be completed by the designer.

Up to now, among UML concepts which have a rather well defined semantics accepted by the UML community, we have extracted a subset adapted to our domain. This excludes concepts such as aggregation/composition and inheritance. Indeed, for example, inheritance of associations, aggregation or state diagrams have not been clarified yet. For this subset, a formal semantics has been defined [7]. Then, a set of rules which translate the precise UML concepts into B specifications have been specified [5].

The aim of this paper is to give an overview of the method through an example (Section 2) and to describe the main features of the prototype support tool (Section 3).

2. Generating B Specifications from UML Diagrams

2.1. Overview of the method

The generation is performed into three steps:

1. The UML class diagram is first elaborated to represent the static structure of the system. Then the corresponding B specifications are generated, completed with basic update B operations calculated from the diagram.

2. Transactions are elaborated by using UML state and collaboration diagrams. State diagrams describe the local behaviour of objects. Collaboration diagrams show how objects collaborate to perform functions of the system. The diagrams are annotated by formal definitions issued from the previous step. Then the formal specification is completed with the translation of these diagrams.

3. Proofs of the global specification can be achieved with a prover dedicated to the B method.

The different diagrams are edited under the Rose environment [11]. Information not graphically expressed is put in the documentation field of the relevant element.

The approach is illustrated through a simple example in the following sections.

2.2. Class Diagrams in UML and B

The class diagram of the running example is described in Figure 1. Property $\{K\}$ specifies that an attribute or a set

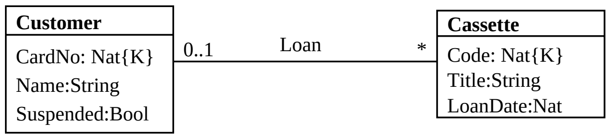

Figure 1. Class diagram of the example

of attributes is the key of a class. *Suspended* is a boolean that specifies that a customer cannot loan because he has overdue loans (more than 90 days late).

In B, each class is mapped into a B machine which contains an abstract set of all possible instances and a variable representing the set of existing instances. Class *Cassette* becomes:

$$\begin{array}{ll} \textbf{SETS} & CASSSETTE \\ \textbf{VARIABLES} & Cassette \\ \textbf{INVARIANT} & Cassette \subseteq CASSETTE \end{array}$$

Each monovalued attribute is modelled as a function($\rightarrow$) between the set of existing instances and the attribute type. A key is translated by a total injective function($\rightarrowtail$).

$$\begin{array}{ll} \textbf{VARIABLES} & Code, Title, LoanDate \\ \textbf{INVARIANT} & Code \in Cassette \rightarrowtail NAT \wedge \\ & Title \in Cassette \rightarrow STRING \wedge \\ & LoanDate \in Cassette \rightarrow NAT \end{array}$$

Each association is modelled as a relation ($\leftrightarrow$) between two sets of existing instances. The relation become a function, an injection,... depending on the multiplicity of the roles.

$$\begin{array}{ll} \textbf{VARIABLES} & Loan \\ \textbf{INVARIANT} & Loan \in Cassette \nrightarrow Customer \end{array}$$

A machine is created for an association if it is subjected to independent operations, that is operations on links between existing objects. Otherwise, this variable is defined in one of the class machine, according to the multiplicity.

In database applications, basic update operations, which are application-independent, can be automatically generated from the class diagram. We generate such operations for inserting or deleting objects of a class or links of an association and for modifying an attribute value. For example, the operation that deletes all the loans of a set of customers is specified as follows:

$\mathbf{B\_DelLoanCustomers(Cu)} =$
PRE $Cu \subseteq ran(Loan)$
THEN $Loan := Loan \rhd Cu$
END

2.3. Specification of the Transactions

Data behaviour is described by transactions triggered by a message called transaction message. UML state and collaboration diagrams are used to specify which basic operations are involved and under which conditions.

A state diagram is elaborated for every relevant class or association. It describes how its basic operations are used. Figure 2 describes a state diagram of *Customer*. A customer can be either "InOrder" or "Suspended". When the event *CancelCustomer* occurs, either the customer is deleted(call to the basic operation *B_DelCustomer*) if he has no current loan, or suspended (call to the basic operation *B_ChangeSuspended*) if he has overdue loans.

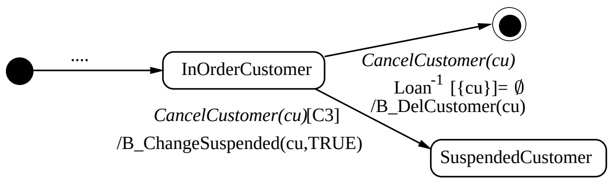

Figure 2. State diagram of Customer

An expression of C3 could be:

$$\exists ca.(ca \in Loan^{-1}[\{cu\}] \wedge$$
$$CurrentDate - LoanDate(ca) > 90)$$

In order to make the generation of B specifications more complete, some rules need to be followed:

– both states and guards are defined by predicates using B notations. For example:
$InOrderCustomer(cu) =$
$cu \in Suspended^{-1}[\{FALSE\}];$

– actions involved in transitions correspond to calls of operation of the related class.

If a transaction message involves several classes, a collaboration diagram elaborates the decomposition of this message into internal messages on each class. In Figure 3, the collaboration diagram for the transaction message *CustomerCancellation* specifies that a customer must be canceled (sending of a *CancelCustomer* event described in the state diagram of *Customer*), that the cassettes he has loaned are deleted (call to the basic operation *B_DelCassette*) and that the loans themselves are also deleted (call to the basic operation *B_DelLoanCustomers*).

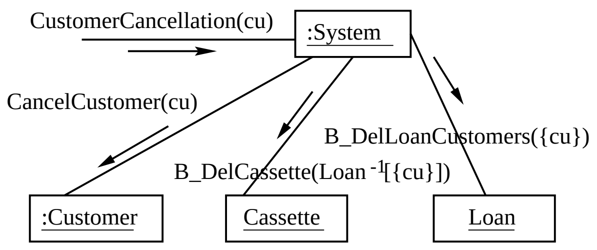

Figure 3. Example of a collaboration diagram

Once the diagrams have been achieved, the corresponding formal specifications can be generated. The class diagram produces basic machines containing the main variables of the system together with basic operations that modify them. The behavioural diagrams complete this architecture by defining machines which include basic machines and have no variables.

First, state diagrams are translated. A new abstract machine is created which includes the corresponding basic machine and uses the necessary machines (those whose variables are used in guard conditions). Each event is translated by an operation. For example, the *CancelCustomer* event is translated by:

```
CancelCustomer(cu) =
PRE    $(InOrderCustomer(cu) \wedge Loan^{-1}[\{cu\}] = \emptyset)$
       $\vee(InOrderCustomer(cu) \wedge C3)$
THEN
       SELECT $(Loan^{-1}[\{cu\}] = \emptyset)$
         THEN $B\_DelCustomer(cu)$
       WHEN $C3$
         THEN $B\_ChangeSuspended(cu, TRUE)$
       END
END
```

The precondition of the operation contains explicitly the predicate of the source state and the guards, and implicitly the preconditions of the basic operations specified on the transitions. Thus we ensure that the operation is called only if the object is in the source state and if the guard is true. We generate a proof obligation for each transition that expresses the fact that the operation related to the transition preserves the invariant of the machine and also establishes the predicate of the target state.

Collaboration diagrams are translated into one abstract machine called *System* which includes all the state diagram machines and the basic machines corresponding to classes which have not an associated state diagram. There is one operation for each transaction message. The operation corresponding to *CustomerCancellation* is:

```
CustomerCancellation(cu) =
PRE    $cu \in Customer$
THEN
       $CancelCustomer(cu)\ ||$
       $B\_DelCassette(Loan^{-1}[\{cu\}])\ ||$
       $B\_DelLoanCustomers(\{cu\})$
END
```

The implicit precondition of this operation is the conjunction of the precondition of the operations *CancelCustomer*, *B_DelCassette* and *B_DelLoanCustomers*.

Proof obligations are also generated that ensure that all the operations (and thus the transactions) preserve the global invariant of the machines describing the whole application.

3. Automatic Translator

Thanks to the precise definitions of the UML core concepts we have proposed, derivation rules have been established [5],[9], and then an automatic translator has been built. The prototype support tool is implemented in OCaml language [8] which is well suited for implementing translation between different languages. In order to build a reliable tool, a formal semantics for UML models has been defined [7]. The basic semantics is specified by metamodels (as in UML [10]), expressed either in a diagrammatic form or with B notation sets. Additional semantics used to clarify the constraints among metamodel components are specified in the form of B invariant clauses. Using these formal metamodels, the tool consists of four logical parts:

a) An abstract syntax of UML: each kind of diagrams is described in an abstract syntax form expressed in OCaml. The OCaml representation is a straightforward translation of the relevant formal UML metamodel.

b) An abstract syntax of B language: it represents the structure of a B machine and its different clauses.

c) A set of checking functions that allow part of the consistency inside any diagram but also between the different diagrams to be checked.

d) A set of generating functions that achieve the translation from UML diagrams into B specifications. They are implemented by OCaml functions working only on the two abstract syntax.

The tool permits the automatic generation of B specifications from a UML description of an application. Some interesting features can be emphasized. The inputs and outputs of the tool are represented in abstract syntax form, thus the tool is independent of commercial toolkits for UML and B. The generated machines are directly usable in the AtelierB toolkit [3]. Thus the proof phase can be directly started in order to establish the correctness of the specifications. OCaml has allowed a rapid development of the prototype and the resulting code is rather concise: 1800 lines are enough for the automation of all diagrams. Furthermore, the OCaml functions are a straightforward translation of the derivation rules.

Nevertheless, up to now, some limits of the tool are to be stated. Some deep semantic analysis are missing on the diagrams. For example, we would like to check that the predicate of a source state is not always false or contradictory. The generated specification is not always optimal. Indeed, in some cases, precondition of operations representing events can be simplified. The amount of automatically generated B specifications strongly depends on the way the state and collaboration diagrams are annotated. More the annotations are formal, more the B specifications are complete. Otherwise, the designer has to complete the generated formal skeletons in order to achieve some consistency checks and formal analyses.

4. Conclusions

This paper gives, through an example, an overview of our method which generates B specifications from an application described using UML notations. Then the outlines of the prototype support tool are presented. The obtained B machines are directly usable in AtelierB. Thus proofs can be performed. In general, it is not the easier activity of the specification. AtelierB generates proof obligations and try to automatically demonstrate them. In case of failure, proofs must be achieved manually. We think it is possible to help the designer during this last task by providing him with a set of proof tactics that could be defined by taking into account the specific domain (database applications) we consider.

Now we are currently working on two complementary axes. The first one consists of extending the tool to automatically generate database implementations (data and programs). We have already defined a set of generic rules for relational implementations [6], based on the refinement process of B. Thus the design and coding phases will be largely automated. Secondly we would like to extend the subset of UML concepts we consider by integrating for example inheritance and agregation/composition.

In Memoriam

We would like to dedicate this paper to Professor Philippe Facon, who died tragically on September 1st, 1999. He was the leader of our research team and was one of the guiding forces of the project presented above.

References

[1] J. R. Abrial. *The B-Book*. Cambridge University Press, 1996.

[2] J. M. Bruel and R. B. France. A Formal Object-Oriented CASE Tool for the Development of Complex Systems. *7th European Workshop on Next Generation of Case Tools*, Crete, Greece, May 1996.

[3] DIGILOG groupe STERIA. *Atelier B, Manuel de référence*. DIGILOG, BP 16000, 13791 Aix-en-Provence Cedex 3 France, 1996.

[4] S. Dupuy, Y. Ledru, and M. Chabre-Peccoud. An Overview of Roz: a Tool for Integrating UML and Z Specifications. *12th International Conference CAISE'00*, Stockhom, Sweden, June 2000.

[5] P. Facon, R. Laleau, and A. Mammar. Combining UML with the B Formal Method for the Specification of Database Applications. *Research Report*, CEDRIC Laboratory, Paris, September 1999.

[6] R. Laleau and A. Mammar. A Generic Process to Refine a B Specification into a Relational Database Implementations. *International Conference ZB2000, LNCS*, Springer-Verlag, York, UK, September 2000, forthcoming.

[7] R. Laleau and F. Polack. Metamodels for Static Conceptual Modelling of Information Systems. *Workshop "Defining Precise Semantics for UML", ECOOP 2000*, Cannes, France, June 2000.

[8] X. Leroy. *The Objective Caml System, Documentation and User's Manual*. INRIA, France, 1999.

[9] H. P. Nguyen. Dérivation de spécifications formelles B à partir de spécifications semi-formelles. *PHD thesis*, CEDRIC Laboratory, Paris, France, December 1998.

[10] OMG. *Unified Modeling Language Specification*. BetaR1 Release, available on line, www.rational.com/uml., April 1999.

[11] Rational Software Corporation. *Rational Rose - Using Rational Rose 98*. 1998.