UNDERSTANDING B SPECIFICATIONS WITH UML CLASS DIAGRAM AND OCL CONSTRAINTS

B. Tatibouët
Laboratoire d'Informatique de l'université de Franche-Comté
16, route de Gray 25030 Besançon cedex

I. Jacques
Laboratoire d'Informatique de l'université de Franche-Comté
16, route de Gray 25030 Besançon cedex

Keywords: B Formal Method, UML notation, object constraint language.

Abstract: B is a formal method (and a specification language) which enables the automatic generation of an executable code through a succession of refinements stemming from an abstract specification. A B specification requires a certain knowledge of mathematical notations (Classical logic and sets) as well as specific terminology (generalized substitutions, B keywords) which may in all likelihood leave a non-specialist of the B notation in the dark. To address this problem, we will extract graphic elements from B specification in an effort to render it more understandable. In a previous work, these visual elements are illustrated in a UML class diagram. These visual elements being insufficient they are completed by OCL constraints allowing to present the invariant and the operations of a B abstract machine.

1 INTRODUCTION

Formal methods are nowadays the most rigorous way to produce software. They provide techniques to ensure the consistency of a specification and to guarantee that some piece of code implements a given specification. Also many studies (Sekerinski, 1998; Meyer and Souquieres, 1999; Laleau and Mammar, 2000) have been carried out over the past few years regarding the generation of B (Abrial, 1996) specifications from OMT or UML (Unified Modeling Language) diagrams.

Given the fact that several significant formal development in B (For example industrial railway (Behm et al., 1999) and smard cards (Casset, 2002) applications) are mainly based on formal approach, we investigate the reverse approach : using graphical notations, such as UML diagrams, as a way to document B formal developments. Such critical applications must usually be accepted by independent certification authorities that are not necessarily expert in formal methods. Therefore, it makes sense to construct a graphical view from formal development as an additional documentation. It is expected that these more intuitive representations will be easier to accept by certifiers.

In our previous works (Hammad et al., 2002; Tatibouet et al., 2002; Jacques et al., 2005) concerning this extraction we particularly described a static view of B specification by a set of generation rules of UML class diagram completed by constraints written with the OCL language (Object Constraint Language).

In this article we will start by revisiting (Section 2) our approach. In the Section 3 we will show how to transform B invariants and B operations into OCL constraints. To conclude, we will examine the limits of our work as well as the prospects of their development.

2 FROM B SPECIFICATION TO UML CLASS DIAGRAM ON THE SCHEDULER'S EXAMPLE

2.1 Abstract Machines

Given that a specification of B is composed of abstract machines, the concept of the machine must find its counterpart in UML. In the techniques generating B from UML, the solution consists in representing a class with an abstract machine. This is a *natural* solution where the class in UML and the abstract machine in B make up an elemental granularity merging data and operations. This solution, then, is the one we

Tatibouët B. and Jacques I. (2006).
UNDERSTANDING B SPECIFICATIONS WITH UML CLASS DIAGRAM AND OCL CONSTRAINTS.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - ISAS*, pages 475-478
DOI: 10.5220/0002490504750478
Copyright © SciTePress

```
MACHINE
    Scheduler
SETS
    PID
CONCRETE_VARIABLES
    active, ready, waiting
INVARIANT
    active ⊆ PID ∧ ready ⊆ PID ∧
    waiting ⊆ PID ∧
    ready ∩ waiting = ∅ ∧
    (active = ∅ or (active ⊄ ready ∪ waiting)) ∧
    card (active) ≤ 1 ∧
    active = ∅ ⇒ ready = ∅
INITIALISATION
    . . .
END
```

Figure 1: Static part of the B scheduler example.

have chosen for the passage from a B specification to UML.

We present the result of these transformations through the scheduler's example : The three variables of the machine (figure 1) are **waiting**, **ready** and **active** which represent respectively the waiting, ready to be activated and active processes.

2.2 Abstract Sets and Sets Variables

In B, the abstract sets as well as the enumerated sets are defined in the clause **SETS**. From these sets it is possible to declare set variables or element of set variables when jointly using the **VARIABLES** clause to define a name and the **INVARIANT** clause to type this variable. For example, in the machine of Figure 1, **PID** is an abstract set and **active** a set included in **PID** (**active** is a subset of **PID**).

An abstract set such as **PID** is represented by a class (indicated here as **E_PID** in Figure 2) corresponding to the type of elements of the set and by an association between this class and the class representing the machine. The extremity of the association on the side of the class corresponding to the type of elements has a name (role) which is that of the set (in this case, **PID**). The set variables constitute associations, with their name acting as the role name on the side corresponding to the type of element.

An OCL constraint enables to express the inclusion between sets **PID** and **active** that does not appear on the class diagram. In constraint 1, the context within which the constraint is expressed is the **Scheduler** class and where **inv** means invariant. By using the **PID** role from that class, we obtain a set. The predefined function **includesAll** assures one that all the elements of the set in parameter (obtained from **active**

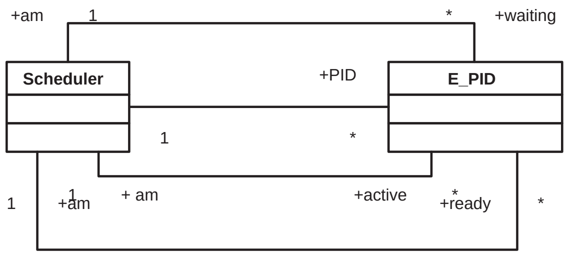

Figure 2: The class diagram corresponding to the Scheduler machine.

role) are included in the set obtained from **PID** role.

Constraint 1

context *Scheduler* ***inv*** *:*
  PID → includesAll (active)
  and PID → includesAll (ready)
  and PID → includesAll (waiting)

3 OBTAINING THE OCL CONSTRAINTS FROM B INVARIANTS AND OPERATIONS

3.1 Transforming B Invariants Into OCL Constraints

The rules, which allow to transform B predicates, B arithmetical expressions, B set expressions, B relations in OCL constraints, are presented in (Jacques et al., 2005).

In B, the invariant consists of a number of predicates separated by the conjunction operator ∧ (See figure 1). Each B predicates are transformed in OCL constraints separated by the OCL keyword **and**. The constraint is thus obtained:

Constraint 2

context *Scheduler* ***inv*** *:*
  and ready→intersection(waiting)→isEmpty()
  and active→isEmpty() or
    not (ready→union(waiting)→includesAll(active))
  and active→size() <= 1
  and active→isEmpty() implies ready→isEmpty()

3.2 Transformation of a B Operation in OCL

3.2.1 Presentation of B Operations

We will deal here with the translation in OCL of the clause **INITIALISATION** and the operations announced in the clause **OPERATIONS**. An operation is composed of a heading and a body as follows :

ResultsList ← OperationName (ParametersList) $\widehat{=}$
    body

The body of the operations and the initialization are expressed through substitutions which are mathematical notations making possible the modeling of predicate transformation. The heading includes an optional list of results implicitly typed in the body of the operation and an optional list of parameters which are explicitly typed if they exist thanks to the precondition substitution (***PRE** Predicate **THEN** Substitutions **END***). The latter allows, too, to establish the precondition on which an operation is called. That substitution makes up the body of the B operation and the other substitutions are included in the *Substitutions* part. The parameter typing and the preconditions are carried out in the *Predicate*.

The example of the **Scheduler** enables one to see the initialization (in the initial state, the three sets are empty) and two operations :

- **new** to create a new process and add it to **waiting**
- **activate** to activate a process of **waiting** and put it in **active** if the set is empty, add it to **ready** otherwise.

3.2.2 General Principle of Transformation

Each operation of an abstract B machine is associated to an operation of a class representing the machine. The initialization is represented by a specific method called **INITIALIZATION**. OCL allows one to associate an operation of a class to a precondition and a postcondition whose syntax is as follows :

***context** Class::Operation(Parameters)**:** Type*
  ***pre** : precondition*
  ***post** : postcondition*

The OCL precondition for an operation is obtained from the predicate of the precondition substitution. The postcondition is obtained in two stages :

1. The B postcondition is calculated from B substitutions through the mechanism of transformation into before-after predicate (Chapters 6.3.3 and 7.1.1. in (Abrial, 1996)).
2. The B postcondition is rewritten in OCL postcondition through the mechanism described in (Jacques et al., 2005).

The constraints obtained through the **Scheduler** example for the initialization and the operations **new** and **activate** are as follows:

```
MACHINE
    Scheduler
    . . .
INITIALISATION
    active, ready, waiting := ∅, ∅, ∅
OPERATIONS
    new(pp) ≙
        PRE
            pp ∈ PID ∧
            pp ∉ active ∧
            pp ∉ (ready ∪ waiting)
        THEN
            waiting := waiting ∪ {pp}
        END;
    activate(qq) ≙
        PRE
            qq ∈ waiting
        THEN
            waiting := waiting - {qq} ||
            IF (active = ∅) THEN
                active := {qq}
            ELSE
                ready := ready ∪ {qq}
            END
        END;
    . . .
END
```

Figure 3: Dynamic part of the B scheduler example.

Constraint 3

context Scheduler::INITIALIZATION()
  post : active→isEmpty() and ready→isEmpty()
    and waiting→isEmpty()
context Scheduler::new(pp : Elt_PID)
  pre : active→excludes (pp)
    and ready→union(waiting)→excludes(pp)
  post : waiting = waiting@pre→including(pp)
context Scheduler::activate(qq : Elt_PID)
  pre : waiting→ includes (qq)
  post : waiting = waiting@pre→excluding(qq)
    and if active@pre→isEmpty() then
      active = Set{qq}
    else ready = ready@pre→including(qq)
    endif

Actually, the postconditions have been obtained intuitively. As far as the operation **activate** is concerned, an equivalent postcondition is calculated in 3.3.

Remark 3.1 *In B, an operation can produce several results. In the UML-OCL translation the first result is considered as the result of the operation and the others are passed into parameters with **out** as passage type.*

3.3 Example of a Postcondition Obtained From Substitutions

The body of the **activate** operation is used again and the rules defined in (Abrial, 1996) are applied to find the B postcondition. The variables primed in B express the value of the variable after realizing the substitutions. In OCL, the before value is expressed by suffixing the variable with **@pre**.

```
    prd(waiting := waiting - {qq} ||
        IF (active = ∅) THEN active := {qq}
        ELSE ready := ready ∪ {qq}
        END)
⇔
    prd(waiting := waiting - {qq}) ∧
    prd(IF (active = ∅)
        THEN active := {qq}
        ELSE ready := ready ∪ {qq}
        END)
⇔
    waiting’ = waiting - {qq}) ∧
    prd((active = ∅ ⟹ active := {qq})
        [] (¬ (active = ∅) ⟹ ready := ready ∪ {qq})
    )
⇔
    waiting’ = waiting - {qq}) ∧
    (prd(active = ∅ ⟹ active := {qq})
    ∨ prd(¬ (active = ∅) ⟹ ready := ready ∪ {qq})
    )
⇔
    waiting’ = waiting - {qq}) ∧
    ((active = ∅ ∧ prd(active := {qq}))
    ∨ (¬ (active = ∅) ∧ prd(ready := ready ∪ {qq}))
    )
⇔
    waiting’ = waiting - {qq}) ∧
    ((active = ∅ ∧ active’ = {qq}))
    ∨ (active ≠ ∅ ∧ ready’ = ready ∪ {qq}))
    )
```

The constraint is deduced of this postcondition :

```
    waiting = waiting@pre→excluding(qq) and
    ((active@pre→isEmpty() and active = Set{qq})
     or (active@pre→notEmpty()
        and ready = ready@pre→including(qq))
    )
```

4 CONCLUSIONS AND PROSPECTS

On the whole, the passage from a B specification to an UML-OCL modeling has proved possible throughout the sample of examples that we processed.

The OCL constraints generated seem readable and understandable to us though less expressive and concise than the predicates expressed in B. That is partly due to the operators directly available in B.

The next stage of our work consists in assessing the possibility to jump forth and back between B and UML specifications.

REFERENCES

Abrial, J. (1996). *The B-Book -Assigning Programs to Meanings*. Cambridge University. Press. ISBN 0-521-49169-5.

Behm, P., Benoit, P., A.Faivre, and J-M.Meynadier (1999). METEOR : A successul application of B in a large project. In *FM’99 : World Congress on Formal Methods*, pages 369–387. LNCS 1709.

Casset, L. (2002). Development of an Embedded Verifier for Java Card Byte Code Using Formal Methods. In *FME’02 : Formal Methods Europe*. LNCS 2391.

Hammad, B., Tatibouet, A., Voisinet, J., and Wu, W. (2002). From a B Specification to UML Statechart Diagrams. In *4th International Conference on Formal Engineering Methods, ICFEM’2002*, pages 511–522, Shangai, China. LNCS 2495.

Jacques, I., Tatibouet, B., and Voisinet, J. (2005). Generation of OCL Constraints from B Abstract Machines. In *SERP’05, 2005 International Conference on Software Engineering Research and Practice*, Las Vegas, Nevada, USA.

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

Meyer, E. and Souquieres, J. (1999). A Systematic Approach to Transform OMT Diagrams to a B Specification. In *FM’99*, LNCS 1708, pages 875–895. Springer-Verlag.

Sekerinski, E. (1998). Graphical design of reactive systems. In *B’98 : Recent Advances in the Development and Use of the B-Method*, LNCS 1393. Springer-Verlag.

Tatibouet, B., Hammad, A., and Voisinet, J. (2002). From a B Specification to UML Class Diagrams. In *2nd IEEE International Symposium on Signal Processing and Information Technology, ISSPIT’2002*, pages 5–10, Marrakech, Morocco. IEEE.