CONSISTENCY CHECK OF CLASS DIAGRAM AND SEQUENCE DIAGRAMS USING B-METHOD

Waitaya Sricharunrat [*1] and Wiwat Vatanawood[2]

Department of Computer Engineering, Faculty of Engineering,
Chulalongkorn University, Pathumwan, Bangkok, Thailand.

ABSTRACT

This paper proposes a systematic mean of consistency check for UML class diagram and its related sequence diagrams representing the critical scenarios using B-Method. The B-Method is a formal specification modeling which is used to describe the semantics of system in terms of mathematical notations – set theory and first-order predicate logic. In our approach, a class diagram and its related sequence diagrams are formally translated into B Abstract Machine (BAM) using a set of our translation rules. Our translation rules generate the semantics of both structural and behavioral properties of the UML class diagram and sequence diagrams.

This paper focuses on two parts. Firstly, the formalization of the UML class diagram - a collection of classes and their relations such as association, aggregation, composition, generalization or inheritance, is investigated and defined for the structural property. Secondly, the formalization of UML sequence diagrams – a collection of scenarios which illustrate the major interactions between related classes as to achieve a specific goal, is defined for the behavioral property and verified against their original structure in class diagram. Moreover, we formally define the complex operations within the critical sequence diagrams by exploiting the calling-called dependency between class operations from Hung Ledang's work. The formal specification in BAM is finally generated and verified by B-Toolkit.

KEYWORDS: UML, Class Diagram, Sequence Diagrams, B-Method, Formal Specifications Modeling, B Abstract Machine

1. INTRODUCTION

UML (Unified Modeling Language) is the language used to analyze and design software system. Both developers and users usually prepare a UML class diagram and its related sequence diagrams to describe the structural and behavioral properties of the target software system. Practically, they have to finish a large number of UML class diagrams and sequence diagrams and the verification of the UML diagrams must be tediously conducted to walkthrough the consistency among the diagrams. An alternative of the systematic approach to deal with these problems is to exploit the formal specifications modeling to ease the consistency checking. The formal specification modeling is a formal description of a software system in terms of mathematical notations as to help prove of syntactical and semantic correctness [1]. Therefore, both developers and users understand the software system model in the same way. [2], [3]

This paper proposes an approach to formally define class diagram and sequence diagrams into formal specifications called BAM (B Abstract Machine). Firstly, the approach will translates all attributes of classes from class diagram and relationships among those classes. Secondly, such approach will translate operations of class diagram from critical scenarios of sequence diagrams. Our approach applies Hung Ledang's Calling-Called Dependency concepts on how to build the hierarchical structure of the related class operations [4], [5]. The result of the translation in BAM statements helps to check the consistency of software system which is represented by class diagram and sequence diagrams.

This paper is organized as follows. Section 2 presents overview of backgrounds. Section 3 explains overview of our purposed scheme, while a case study of cash money transfer bank will be

* Corresponding author.
E-mail: Waitaya.S@Student.chula.ac.th, wiwat@chula.ac.th

described in section 4. Section 5 illustrates the formalization of UML class diagram and sequence diagrams. Finally, in section 6, the conclusion of this work is discussed.

2. OVERVIEW OF BACKGROUNDS

2.1 The Calling-Called Dependency between Class Operations [4], [5]

Hung Ledang proposed an approach to build the relationship among the class operations into hierarchical tree, called the Calling-Called dependency between class operations, to help construct a BAM. Hung Ledang divided the class operations regarding their calling behaviors into 2 groups; 1) Non-Basic operations are the class operations that typically call the other operations during their run-time activities and 2) Basic operations are the operations that typically not call the other operations.

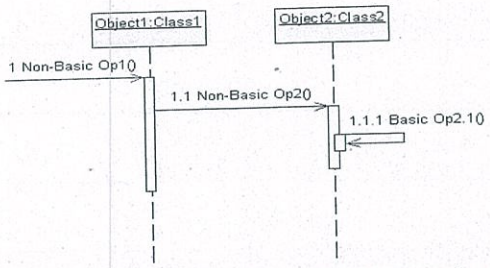


Figure 1.Non-Basic and Basic Operations from critical scenario of sequence diagram

In figure 1, both operation Op1( ) and Op2( ) are Non-Basic operations that invoke the other operations at least once. For example, the operation Op1( ) calls Op2( ) once while the operation Op2( ) also calls Op2.1( ). The operation Op2.1( ) is Basic operation that does not call the other operations at all. The Non-Basic operation Op1( ) is a calling-operation, and the Basic Op2.1( ) is called-operation. While the Non-Basic operation Op2( ) is both calling-operation and called-operation as shown in figure 2.

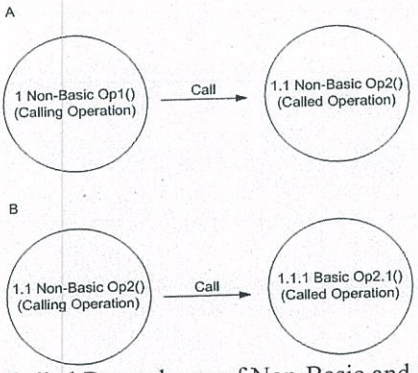
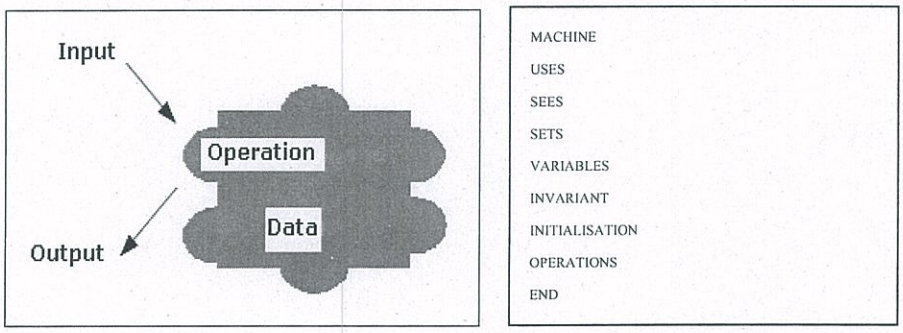


Figure 2.Calling-Called Dependency of Non-Basic and Basic Operations

2.2 B-Method [1]

The B-Method represents a formal specifications modeling that can be used in software development life cycle. The specifications method focuses on the concepts of modularity and information hiding. The BAM notations are used to specify a module to represent each class or object. Each module is defined to encapsulate structural and behavioral properties. The relationship between the BAMs can be defined to represent their collaborations. In fact, developer practically considers a BAM module to a class and utilizes them for developing many complex systems. The structure of BAM is graphically illustrated in figure 3A and the essential clause names of BAM syntax is listed in figure 3B.



(A) (B)

Figure 3.The Structure of BAM
(A) Information hiding and (B) Importance Clauses in BAM

3. OUR PURPOSED SCHEME

We propose a scheme of translating both UML class diagram and related sequence diagrams into BAM specifications and eventually conduct the syntactical and semantic consistency checking using B-Toolkit program [6]. The overview of our proposed scheme is shown in figure 4. We begin to consider the given class diagram and map each class to a BAM module with attributes. A set of BAM skeleton modules is generated with the corresponding attributes. The relations among classes are considered as well to create the relations of BAM accordingly. The related interaction among classes in sequence diagrams will be considered especially on the class operations and the operations of the BAM modules are completely appended. In B-Methods, a BAM implementation module is expected to describe the details of called operations sequences. We provide a set of rules to generate the associated BAM implementation modules. The formal specifications in BAM – the BAM modules with relations and their implementation modules, will be finally gathered and refined. We provide a set of translation rules to cope with activities mentioned above. The B-Toolkit program is used to do the consistency checking. Both developers and users will be guided and provided with our systematic scheme to evaluate their software system model in the early stage.

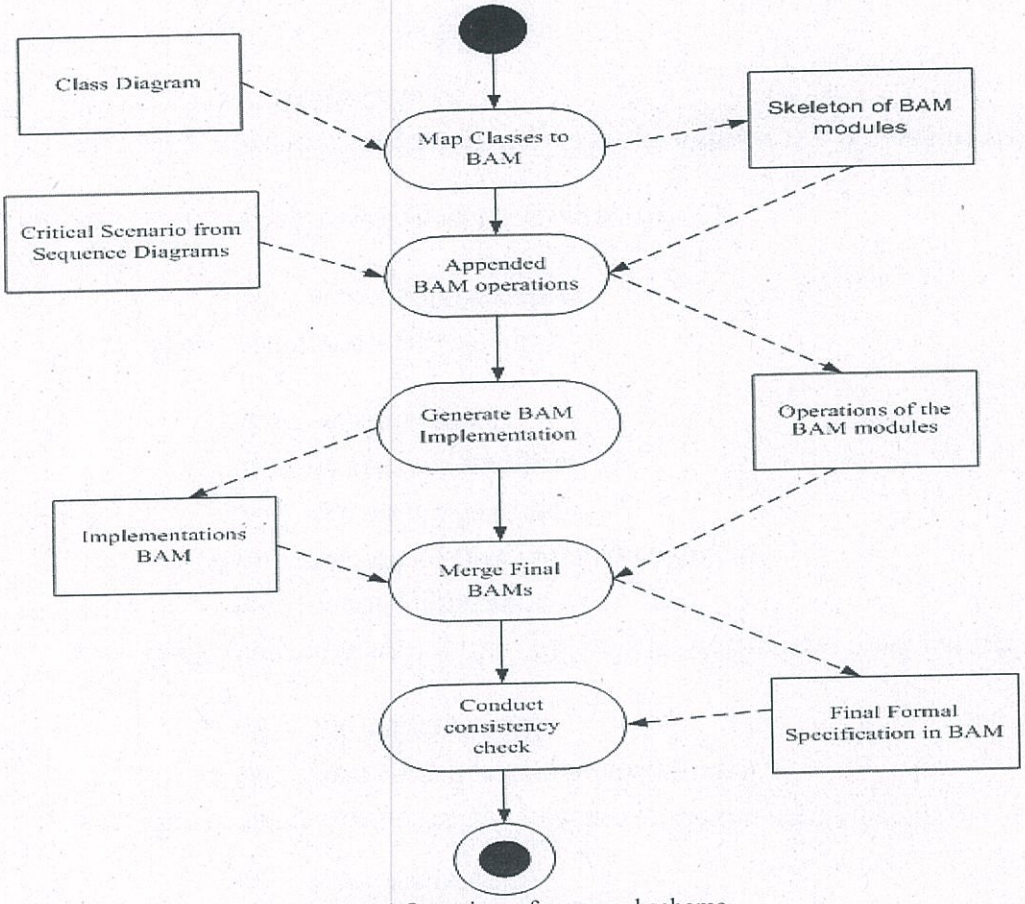


Figure 4.Overview of purposed scheme

4. CASE STUDY

In this section, we introduce a case study of Tuition fee payment system. The class diagram, in figure 5, shows a set of classes named as class *Bank, Student, StudentAccount, UniversityAccount, UndergraduateStudent* with attributes, operations and their relations. The types of relations are drawn with multiplicity notations to describe the structural property of the Tuition fee payment system. To demonstrate one of the payment scenarios, a sequence diagram, in figure 6, shows the interaction between classes to conduct the transfer cash from *StudentAccount* to *UniversityAccount*. A student requests *Bank* to do the operation *transferCash()*. The *Bank* performs the requested operation by asking the *StudentAccount to* do the *withdrawMoney()* operation and asking the *BankAccount* to do the *depositMoney()*. To order to withdraw the money, the *StudentAccount* will perform the called operation *decreaseAmount()*.

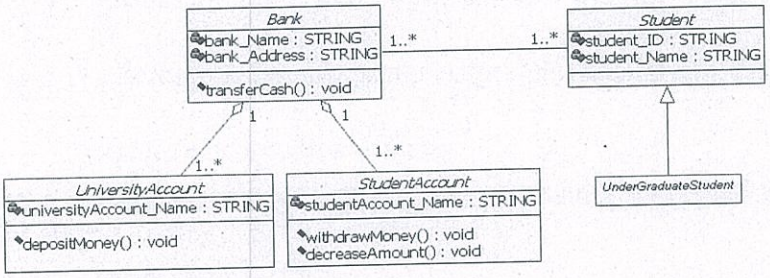


Figure 5.Class diagram of a Tuition fee payment system

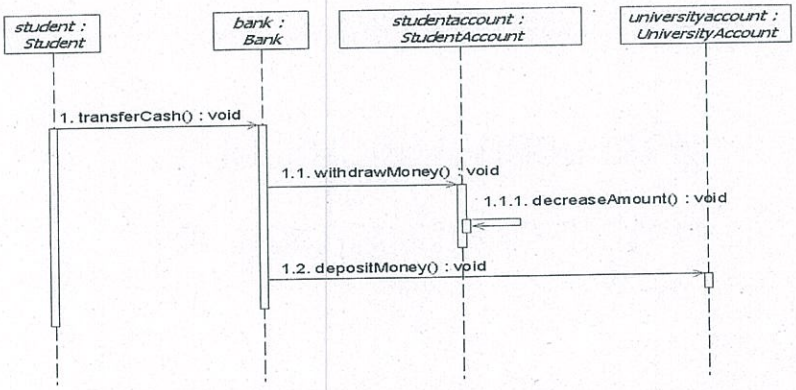


Figure 6.Sequence diagram of a cash payment scenario

5. TRANSLATION OF UML CLASS DIAGRAM AND SEQUENCE DIAGRAMS INTO BAM

This section will distinguish between the rule for translation of class diagram and of sequence diagrams by using a sample in case study of tuition fee payment system as follow

1. Generate BAM of BasicClass and BAM of Class will be described in section 5.1.
2. Generate BAM of Relation between classes are association, aggregation and composition. These will be provided in section 5.2.
3. Generate BAM of sub class inherited all of attributes from super class will be explored in section 5.3
4. Generate BAM of Relation (or BAM implicit relation) between sub class inherited from super class and the other class will also be presented in section 5.3.

5.1 Translating UML Class Diagram

In this section, we demonstrate the translating of UML class diagram into BAM gradually. We create several BAM modules to each class and named it accordingly in the MACHINE clause, for example, *BasicStudent* and *Student* to represent class *Student*. All of the attributes of each class are defined into VARIABLES clause. The types of attributes will be considered as sets in SETS clause while INVARIANT clause define the domain set of each variable found in VARIABLES clause. The INITIALISATION clause contains the initial preconditions of each essential attributes in a BAM module. The USES clause will represent the relation between a BAM and the others. The BAM modules of *BasicStudent* and *Student* are shown in figure 7.

MACHINE BasicStudent
...
SETS BASICSTUDENT
VARIABLES basicstudent,
student_ID, student_Name
INVARIANT basicstudent $\subset$ BASICSTUDENT $\wedge$
student_ID $\in$ STRING $\wedge$ student_Name $\in$ STRING
INITIALISATION basicstudent := ϕ ||
student_ID := null || student_Name := null
END

MACHINE
Student
USES
BasicStudent
END

Figure 7.The BAM modules of class *Student*

5.2 Translating the Relations between Classes

An association between two classes in UML is formally defined as a BAM module with "Asso" prefix to its name. The association is considered as a set of order pair of Cartesian product of two relating classes. The multiplicity of the association will be defined as well to represent the number of instance of each class which has relationship. Table 1 shows the mapping between predicates for variety of multiplicity.

Table 1.The Multiplicity and its mapping predicates

Predicate	Multiplicity
RelName $\subset$ BASICCLASS1 $\times$ BASICCLASS2 dom(RelName) = basicclass1 $\wedge$ ran(RelName) = basicclass2 $\wedge$ $\forall$ (xx,yy).(((xx $\in$ dom(RelName)) $\wedge$ (yy $\in$ ran(RelName))) $\rightarrow$ card((RelName)[{xx}]) $\geq 0 \wedge$ card((RelName)$^{-1}$[{yy}] ≥ 0)	* Or 0…*
RelName $\subset$ BASICCLASS1 $\times$ BASICCLASS2 dom(RelName) = basicclass1 $\wedge$ ran(RelName) = basicclass2 $\wedge$ $\forall$ (xx,yy).(((xx $\in$ dom(RelName)) $\wedge$ (yy $\in$ ran(RelName))) $\rightarrow$ card((RelName)[{xx}]) $\geq 1 \wedge$ card((RelName)$^{-1}$[{yy}]) ≥ 1)	1…*
RelName $\subset$ BASICCLASS1 $\times$ BASICCLASS2 dom(RelName) = basicclass1 $\wedge$ ran(RelName) = basicclass2 $\wedge$ $\forall$ (xx,yy).(((xx $\in$ dom(RelName)) $\wedge$ (yy $\in$ ran(RelName))) $\rightarrow$ card((RelName)[{xx}]) = 1 $\wedge$ card((RelName)$^{-1}$[{yy}]) = 1)	1
RelName $\subset$ BASICCLASS1 $\times$ BASICCLASS2 dom(RelName) = basicclass1 $\wedge$ ran(RelName) = basicclass2 $\wedge$ $\forall$ (xx,yy).(((xx $\in$ dom(RelName)) $\wedge$ (yy $\in$ ran(RelName))) $\rightarrow$ card((RelName)[{xx}]) $\geq 0 \wedge$ card((RelName)[{xx}]) $\leq 1 \wedge$ card((RelName)$^{-1}$[{yy}]) $\geq 0 \wedge$ card((RelName)$^{-1}$[{yy}]) ≤ 1)	0…1

The sample of a BAM module for the association between class *Bank* and *Student*, in figure 5, is shown in figure 8. The Aggregation and composition in UML are defined in the similar steps. We use "Aggr" and "Compo" as the prefix to theirs names respectively. Figure 9 shows the composition between *Bank* and *StudentAccount*. In order to implement the aggregation and composition between two classes, the container class has to carry the implicit references to all contents classes. Therefore, we automatically add a reference variable to the container class. As shown in figure 10, the class *Bank* contains a set of *StudentAccount* object so that the reference to *StudentAccount* object called *RefStudentAccountID* is added.

MACHINE Asso_Bank_Student
USES BasicBank, BasicStudent
VARIABLE asso_bank_student
INVARIANT asso_bank_student $\subset$ BASICBANK $\times$ BASICSTUDENT $\wedge$
dom(asso_bank_student) = basicbank $\wedge$ ran(asso_bank_student) = basicstudent $\wedge$
$\forall$ (xx,yy).(((xx $\in$ dom(asso_bank_student)) $\wedge$ (yy $\in$ ran(asso_bank_student)))
$\rightarrow$ card((asso_bank_student)[{xx}]) $\geq$ 1 $\wedge$ card((asso_bank_student)$^{-1}$[{yy}]) $\geq$ 1)
...
END

Figure 8.A sample of the BAM for association

MACHINE Compo_Bank_StudentAccount
USES BasicBank, BasicStudentAccount
VARIABLE compo_bank_studentaccount
INVARIANT compo_bank_studentaccount $\subset$ BASICBANK $\times$ STUDENTACCOUNT $\wedge$
dom(compo_bank_studentaccount) = basicbank $\wedge$
ran(compo_bank_studentaccount) = basicstudentaccount $\wedge$
$\forall$ (xx,yy).(((xx $\in$ dom(compo_bank_studentaccount)) $\wedge$
(yy $\in$ ran(compo_bank_studentaccount)))
$\rightarrow$ card((compo_bank_studentaccount)[{xx}]) $\geq$ 1 $\wedge$
card((compo_bank_studentaccount)$^{-1}$[{yy}]) = 1)
...
END

Figure 9.A sample of BAMs for composition

MACHINE BasicBank
...
VARIABLES
...
RefStudentAccountID
INVARIANT
...
RefStudentAccountID ∈ STRING
INITIALISATION
...
RefStudentAccountID := null
END

Figure 10.A sample of implicit reference for container class

5.3 Translating Sub Class Inherited from a Super Class

When a sub class is inherited from a super class, the sub class will have almost the same properties as super class. All of the non-private attributes and operations will be implicitly copied from super class, as well as various relations. Technically, to generate a BAM for a sub class, we can copy the BAM of super class and paste into the BAM of sub class and do the refinement. In figure 11, a BAM for the sub class *UnderGraduateStudent* is shown, all of the attributes and operations are copied from the super class *Student* and the variable names are refined to avoid the name clashing.

MACHINE BasicUnderGraduateStudent
...
SETS BASICUNDERGRADUATESTUDENT
VARIABLES basicundergraduatestudent,
undergraduatestudent_ID, undergraduatestudent_Name
INVARIANT basicundergraduatestudent ⊂ BASICUDERGRADUATESTUDENT ∧
undergraduatestudent_ID ∈ STRING ∧ undergraduatestudent_Name ∈ STRING
...
END

Figure 11. A sample of basic BAM for the sub class UnderGraduateStudent

Moreover, the association between the super class Bank and Student will be implicitly inherited to the sub class UnderGraduateStudent as well. We define the extra implicit association BAM between sub class Bank and UnderGraduateStudent as shown in figure 12.

MACHINE ImplicitAsso_Bank_UnderGraduateStudent
USES BasicBank, BasicUnderGraduateStudent
VARIABLES implicitasso_bank_undergraduatestudent
INVARIANT
implicitasso_bank_undergraduatestudent
$\subset_{pa}$ BASICBANK × BASICUNDERGRADUATESTUDENT ∧
dom(implicitasso_bank_undergraduatestudent) = basicbank ∧
ran(implicitasso_bank_undergraduatestudent) = basicundergraduatestudent ∧
∀ (xx,yy).(((xx ∈ dom(implicitasso_bank_undergraduatestudent)) ∧
(yy ∈ ran(implicitasso_bank_undergraduatestudent)))
→ card((implicitasso_bank_undergraduatestudent)[{xx}]) ≥ 1 ∧
card((implicitasso_bank_undergraduatestudent)$^{-1}$[{yy}]) ≥ 1)
...
END

Figure 12.A sample of implicit association BAM for the sub class UnderGraduateStudent

5.4 Appending the Operations to BAM from Sequence Diagrams

After a set of BAMs is generated from a class diagram, these BAMs must be appended with their operations described in the related sequence diagrams. Typically, the sequence diagram shows the operation names invoked and the correspondent operations between two classes. Using Hung Ledang's technique called Calling-Called Dependency Model [4], we can classify the operations into "Basic" and "Non-Basic" operation groups. The Basic operations will be appended into OPERATIONS clause of

basic BAM modules and the Non-Basic operations will be appended into the related original BAM modules.

As shown in figure 13, every operation names in sequence diagrams will be appended into the BAMs completely.

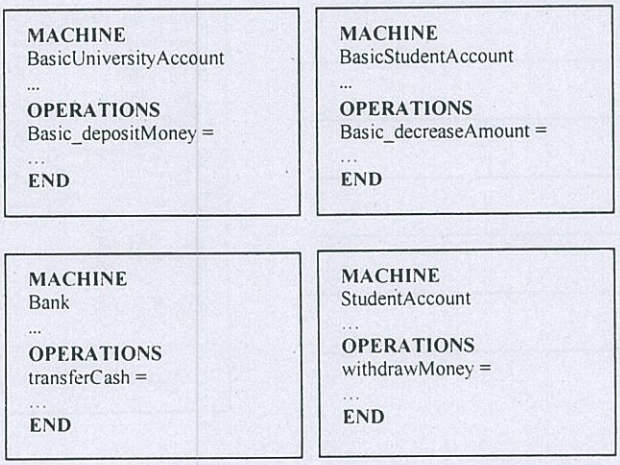


Figure 13.A sample of operation names appended into BAMs

5.5 Generating the BAM Implementation modules

In order to illustrate the sequence of the calling-called operations for each scenario described by a sequence diagram, we generate the extra BAM implementation modules to refine these sequences of invoked operations. As shown in figure 14, the BAM implementation module of *StudentAccount* shows that the *withdrawMoney* operation in BAM will call another operation named *Basic_decreaseAmount*. Another example is the BAM implementation module of *Bank*. The *transferCash* operation in BAM will call two other operations named *withdrawMoney* and *Basic_depositMoney* in sequential order.

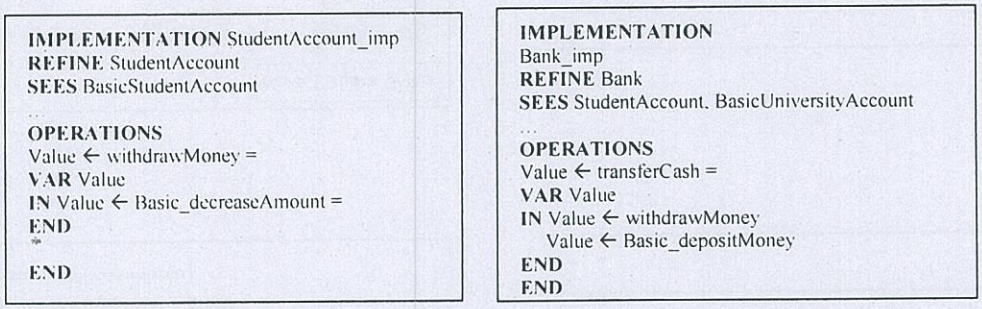


Figure 14.A sample of the BAM implementation modules

6. DISCUSSION

This paper has been further developed from Hung Ledang's research, which allow the user to specify multiplicity of each class appropriately in BAM of relation such as association, aggregation, composition and BAM of implicit relation too. Therefore these can support "association class" of UML class diagram, and develop calling - called operations of critical scenario from sequence diagrams. These help to increase flexibility of UML design from developer.

7. CONCLUSION

We propose an alternative of the consistency checking for UML class diagram and its related sequence diagrams. Given a set of class diagram and sequence diagrams, we propose a set of translation rules to map a class, its attributes, operations and relations between classes into a formal specification notation called BAM. The translation rules guide to generate a set of BAMs systematically and do some automatic refinement of the specification as well. The varieties of relations are covered such as association, aggregation, composition, inheritance. The BAMs will be refined using a set of BAM implementation modules by describing the sequence order of the invoked operations in each scenario of the software system.

A case study of Tuition fee payment system is briefly described and the examples of the BAMs are shown. The final BAM specification has been checked by B-Toolkit program and the result has no conflict and applicable.

REFERENCES

[1] Abrial. J- R. **1996** *The B – Book Assigning Programs to Meanings*. Cambridge University Press.

[2] Grady Booch. Jame Rumbaugh and Ivar Jacobson. 1998 *The Unified Modeling Language User Guide*. Addison Wesley.

[3] Arlow. J.. Neustadt. I..**2002** *UML and The Unified Process Pratical Object – Oriented Analysis and Design*. Addison Wesley.

[4] Ledang .H.and Souquières. J. **2001** Integrating UML and B Specification Techniq ues. *Workshop at Informatik,2001* .

[5] Ledang .H.and Souquières.J. **2001** Modeling Class Operations in B: a case study on the pump component. *Technical Report A01-R-011. Laboratory Lorrian de Recherche en Informatique et ses Applications, 2001.*

[6] Lano .K.and Haughton.H. **1996** *Specification in B: An Introduction using the B Toolkit*. Imperial College Press.