Journal of Theoretical and Applied Information Technology
30th November 2013. Vol. 57 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

USING Z FORMAL SPECIFICATION FOR ENSURING CONSISTENCY IN MULTI-VIEW MODELING

[1]KHADIJA EL MILOUDI, [2]YOUNES EL AMRANI, [3]AZIZ ETTOUHAMI
[1] LCS laboratory, Faculty of Sciences, University Mohammed V-Agdal, Rabat, MOROCCO
[2] LRI laboratory, Faculty of Sciences, University Mohammed V-Agdal, Rabat, MOROCCO
[3]LCS laboratory, Faculty of Sciences, University Mohammed V-Agdal, Rabat, MOROCCO
E-mail: [1] elmiloudi.khadija@gmail.com, [2]elamrani@fsr.ac.ma, [3]touhami@fsr.ac.ma

ABSTRACT

Consistency between different UML diagrams is an important challenge in object oriented modeling but UML lacks any mechanism to rigorously check consistency between the models. This paper presents the first formal semantics of UML sequence diagram using Z notation. The main focus of our approach is to guarantee consistency between sequence and class diagram in multi view modeling context. By means of a representative example, we show how our approach is used for the detection of inconsistencies.

Keywords: *Z, UML, Formal Methods, Sequence Diagram, Multi-view Modeling, Consistency Checking.*

1. INTRODUCTION

During software development, models are built representing different views on a software system. We focus on design models expressed in the Unified Modeling Language (UML) [1] and more specifically on class and sequence diagrams. This paper reports our recent results on formalizing UML sequence diagram in Z notation [2]. This formal model will be used to study the multi-view consistency compared to class diagram formalization presented in a previous paper [3]. Examples are offered to demonstrate the approach. The remainder of this paper is organized as follows. In Section 2, related work is discussed. The Z formalization of sequence diagram is defined in Section 3. Section 4 overviews a set of multi-view inconsistencies handled by the proposed model. Finally, the conclusions are drawn in Section 5 as well as future work.

2. RELATED WORK

A wide range of approaches for the formalization of behavioral diagrams and checking consistency has been proposed in the literature.

Dubauskaite and Vasilecas in [4] chooses to use UML to express consistency rules among different views of UML models. The rules are defined at the metamodel level. Our approach is more thorough thanks to Z which has a precise semantics based on mathematical notations that removes ambiguities compared to UML.

In [5] a framework for deriving B specifications from UML structure and behavioral diagrams is proposed. The conformance between two aspects of UML specifications can be formally verified by analyzing the corresponding B specification. Their proposal has been applied to derive automatically B specifications from class and interaction diagrams. Our approach is similar to [5] in terms of use of formal methods. We derive automatically Z specifications from class and sequence diagrams. The difference between their approach and ours is that paper focus more on multi view consistency by providing theorems and predicates.

In [6] the authors uses an algorithmic approach to a consistency check between UML Sequence and State diagrams. The BVUML tool is implemented for automating the validation process. Our paper focuses on a consistency check between the static and the dynamic view expressed respectively by the class and the sequence diagram. The strength of our approach takes root into the simplicity and visibility of the Z notation which allow the use of the Z/EVES system to automatically process the model.

A formal semantics of UML sequence diagram is presented in [7]. The semantics captures the consistency between sequence diagram with class diagram and state diagram. The sequence diagram is represented as an ordered hierarchical tree structure. Our paper proposes a semantics of UML sequence diagram using Z notation allowing the automatic checking of consistency of UML models which is not available in [7].

Journal of Theoretical and Applied Information Technology
30th November 2013. Vol. 57 No.3
© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The paper [8] proposes safe composition as a technique for consistency checking of multi view models with variability. A representative set of UML consistency rules and a feature composition technique are used. A categorization scheme of consistency rules defined in [8] is used in our paper.

Authors of [9] defines the semantics of UML class and sequence diagrams using basic set notations. Their approach is based on an attribute grammar reflecting the semantic properties of programs. A set of axioms which reify the principles of OO programming is defined. Contrary to what has been discussed in [9], formal methods contribute to efficient software development with very low rate of defects. The formal model presented in this paper is automatically generated and can be easily proven using Z/EVES system. The simplicity and clarity of the proposed model enable easily its reuse for the automatic generation of correct programs.

In [10], thirteen consistency rules are given to identify inconsistencies between the most frequent 6 types of UML diagrams used in the information systems modeling. Four methods are provided to check inconsistencies between UML diagrams. The four methods are: manual check, compulsory restriction, automatic maintenance and dynamic check. Consistency rule identified in [10] are handled by our approach.

We present in the rest of this paper the contribution of our approach compared to existing work. Our approach handles the various rules discussed in the literature especially those corresponding to the consistency between the class and sequence diagram. It is worth noting that our approach is the first work on sequence diagram formalization and multi view consistency checking based on Z notation [2].

3. Z FORMALIZATION OF SEQUENCE DIAGRAMS

UML sequence diagrams are behavioral diagrams used to represent the interaction between different objects in the system over time in many different situations. These objects are instances of classes defined in the class diagram.

As we will define the semantics of a sequence diagram in the context of a class diagram, we briefly introduce the notation of class diagrams first. The semantics of class diagrams are detailed in a previous paper [3].

In Object-oriented modeling, a class describes the state and behavior of the class objects. The set of all object identities is introduced as the given set [OBJECT]. To model the set of all classes we introduce the given set [CLASS].

We illustrate our approach by the case study proposed in [8]. Consider the class diagram of a video on demand system (VOD) shown in Figure 1. The class diagram consists of three classes: Service, Streamer, and Program. These classes have some methods, a navigable association going from Service to Streamer, and one from Streamer to Program. Lastly a sequence diagram illustrates a call of method select in a Service object and a call of method stream from Service to Streamer [8].

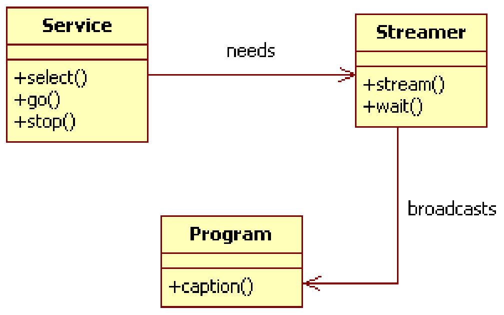

Figure 1: Class Diagram of VOD System

The following schema denotes the class Service. An attribute self represents the identifier of the current instance.

```
 ┌── ServiceCoreClass ─────────────────
 │ self: SERVICE
 └─────────────────────────────────────
```

Attributes are represented in our model as state variables with their types. In this case, the class Service does not contain attributes.

Then a free type is defined. It adds an optional nil value not available in Z to be used in initializations.

$Service ::= nilService$
$\quad | \ servicecoreclasstoservice \langle\langle ServiceCoreClass \rangle\rangle$

In our example, a schema called SService represents all instances of the class. The state variable services denotes the set of the instances of the class Service identified by the system. The state variable serviceIds is the set of their identities. A function idService binds each unique instance identifier to the corresponding class. More details are available in [3].

```
__SService ______________________________
services: ℙ Service
idService: SERVICE ⇸ Service
serviceIds: ℙ SERVICE
_________________________
idService
   = { servicecoreclass: ServiceCoreClass
          • servicecoreclass.self
            ↦ servicecoreclasstoservice
                         servicecoreclass }
serviceIds = dom idService
_________________________________________
```

The function idService guarantees that no two objects in the state have the same identity.

And finally a schema entitled System provides an overview of all the classes of the system. The system represents all classes modeling a concept.

```
__System ________________________________
SService
SStreamer
SProgram
_________________________________________
```

The class behavior is also specified in the formal model of class diagram using methods. Each method is represented by a schema defining its signature. Each operation includes a schema that indicates whether the system state will be changed (ServiceO*p* below) or remains unchanged. This schema also guarantees that the object identifier (*self*) remains unchanged. The method parameters are defined as inputs in the form (data?: type) in the schema of the operation. The method select does not have parameters.

```
__ServiceOp______________________________
ΔServiceCoreClass
_____________________________
self' = self
_________________________________________
```

```
__selectService__________________________
ServiceOp
_________________________________________
```

The complete formal model of class diagram used in this paper is shown with more detail in [3].

Consider the following example of sequence diagram specifying a particular scenario of the video on demand system. This example introduces the first formal semantics of sequence diagram in the literature based on Z notation.

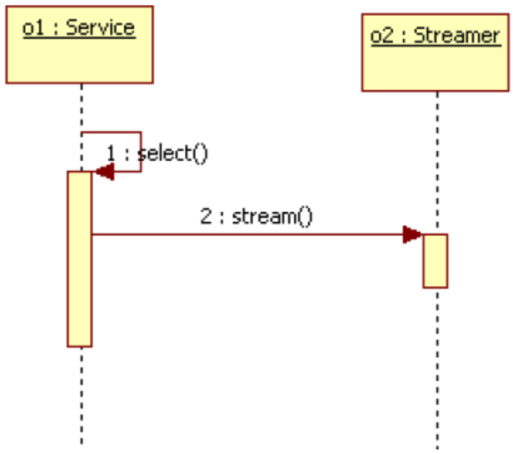

Figure 2: Example of Sequence Diagram

The objects *o1* and *o2* are respectively instances of the classes Service and Streamer. We declare o1 and o2 as OBJECT and later we precise their belonging to their respective classes.

```
| o1, o2: OBJECT
```

To represent all the class operations, we introduce an enumerated set OP containing all the methods. The methods must be defined in the class diagram formalization as explained before. The parameters of the methods are defined as inputs in the schema of each operation.

```
OP ::= select ⟨⟨selectService⟩⟩
     | go ⟨⟨goService⟩⟩
     | stop ⟨⟨stopService⟩⟩
     | stream ⟨⟨streamStreamer⟩⟩
     | wait ⟨⟨waitStreamer⟩⟩
     | caption ⟨⟨captionProgram⟩⟩
```

The messages exchanged between the objects o1 and o2 are calls to the methods previously defined. The method calls are represented by an instance of each method.

$SELECT: selectService$
$GO: goService$
$STOP: stopService$
$STREAM: streamStreamer$
$WAIT: waitStreamer$
$CAPTION: captionProgram$

Now, the semantics of the sequence diagram can be fully defined.

A sequence diagram consists of a set of objects interacting with each other by means of a sequence of messages. We choose to model the sequence diagram as a Z sequence of messages. Each message is defined by a tuple representing the sender of the message, the receiver of the message and the message.

SequenceDiagram

$Objects: \mathbb{P}\ OBJECT$
$Messages: \mathrm{seq}\ (OBJECT \times OBJECT \times OP)$

$Objects = \{o1, o2\}$
$\forall\ System$
$\bullet\ \exists\ serviceId: \mathbb{P}\ serviceIds;$
$\quad streamerId: \mathbb{P}\ streamerIds$
$\quad \bullet\ o1 \in serviceId \wedge o2 \in streamerId$
$Messages = \langle (o1, o1, select\ SELECT),$
$\qquad (o1, o2, stream\ STREAM) \rangle$

The first predicate defines the set of objects which participate in the sequence diagram. These objects must be included in the set of instances of their respective classes. This constraint is expressed through the second predicate. And finally, the sequence of messages is explicitly expressed by the third predicate.

4. CONSISTENCY CHECKING

Since the behavior of an object is described with a class diagram and its interactions with other objects are specified with different sequence diagrams, the multi view consistency must be checked.

The formal semantics of sequence diagram defined above allows checking if a sequence diagram is consistent with a class diagram.

The current OMG UML [1] specification provides well-formedness rules for the syntax of UML diagrams. The definition in UML of the semantics of these diagrams is in natural language which is ambiguous. Whereas our semantic provided in Z is precise and non-ambiguous.

The following Consistency rules expressed in Z notation [2] describe the semantic relationships that must hold between the various components of the views. These rules are defined as inter-view rules according to the classification given by [8].

Rule 1: Message name must match class method

Each message on sequence diagram must correspond to an instance of class method defined in the class diagram. The enumerated set OP contains all the methods that are previously defined in the class diagram. Therefore, each message necessarily corresponds to one of these methods. Otherwise, this rule is automatically detected using the Z/EVES System [11].

Rule 2: Each object must have a corresponding class in the class diagram

This rule is stated as predicate in the schema SequenceDiagram. The following predicate specifies that each object in sequence diagram must belong to the set of instances of its class defined in the class diagram.

$\forall\ System$
$\bullet\ \exists\ serviceId: \mathbb{P}\ serviceIds;$
$\quad streamerId: \mathbb{P}\ streamerIds$
$\quad \bullet\ o1 \in serviceId \wedge o2 \in streamerId$

Rule 3: A message of sequence diagram must correspond to an operation of the receiver object.

To illustrate this consistency rule, we propose a Z theorem.

The following theorem states that each message appearing in sequence diagram belong to the set of class operations of the receiver object.

$$\begin{array}{l}
\textbf{theorem } \mathit{methods} \\
\quad \forall\ \mathit{SequenceDiagram} \\
\quad\quad \bullet\ \exists\ \mathit{Sender, Receiver: Objects; Operation: OP} \\
\quad\quad\quad \bullet\ \langle\langle \mathit{Sender, Receiver, Operation} \rangle\rangle \text{ in } \mathit{Messages} \\
\quad\quad\quad\quad \Rightarrow \mathit{Operation} \in \mathit{methodsOfObject\ Receiver}
\end{array}$$

An additional function that return the set of the class operations corresponding to each object is used in the formal definition of the consistency rule.

$$\begin{array}{|l}
\mathit{methodsOfObject: OBJECT} \rightarrow \mathbb{P}\ \mathit{OP} \\
\hline
\mathit{methodsOfObject\ o1} = \{\mathit{select\ SELECT,\ go\ GO,\ stop\ STOP}\} \\
\mathit{methodsOfObject\ o2} = \{\mathit{stream\ STREAM,\ wait\ WAIT}\}
\end{array}$$

We first define the set of class operations of each object through the function methodsOfObjects. Then the verification is done by proving the theorem true.

5. CONCLUSIONS AND FUTURE WORK

This paper presents a formal semantics of UML sequence Diagram allowing the consistency check between the static view of a system expressed by the class diagram and the dynamic view expressed by the sequence diagram. Our approach is fully automated and checked using the Z/EVES System [11] providing one of the first Z formal specification for the sequence diagram. Our approach can be applied for checking more UML inconsistencies offered in the literature. The formal model presented in this paper can be extended to cover all UML diagrams.

REFRENCES:

[1] The Object Management Group: UML 2.3 superstructure specification. http://www.uml.org/ (last access, July 2013)

[2] J.M. Spivey, "The Z Notation: A Reference Manual", Prentice Hall International, Oxford, 1998.

[3] K. El Miloudi, Y. El Amrani and A. Ettouhami, "An Automated Translation of UML Class Diagrams into a Formal Specification to Detect UML Inconsistencies", *Sixth International Conference on Software Engineering Advances* (Barcelona), 2011, pp. 432-438.

[4] R. Dubauskaite, and O. Vasilecas, " Method on Specifying Consistency Rules among Different Aspect Models, expressed in UML", *Electronics and Electrical Engineering*, Vol. 19, No. 3, 2013, pp. 77-81.

[5] H. Ledang and J. Souquières, "Formalizing UML Behavioral Diagrams with B", *The Tenth OOPSLA Workshop on Behavioral Semantics: Back to Basics*, Tampa Bay, Florida (USA), October 15, 2001, Notheastern University Press, pp. 162–171. http://www.loria.fr/~souquier/publications/oopsla01.pdf.

[6] B.Litvak, S. Tyszberowicz, and A. Yehudai, "Behavioral Consistency Validation of UML diagrams", *Proceedings of 1st IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, IEEE Computer Society, 2003, pp. 118–125.

[7] X. Li, Z. Liu, and J. He, "A Formal Semantics of UML Sequence Diagrams", *Proceedings of Australian Software Engineering Conference (ASWEC'2004)*, April 13-16, Melbourne, Australia. IEEE Computer Society, 2004.

[8] R. E. Lopez-Herrejon, A. Egyed, "Detecting Inconsistencies in Multi-View Models with Variability", *Sixth European Conference on Modelling Foundations and Applications (ECMFA)*, 2010, pp. 217-232.

[9] F. Xia and G. S. Kane, "Defining the Semantics of UML Class and Sequence Diagrams for Ensuring the Consistency and Executability of OO Software Specification", *First International Workshop on Automated Technology for Verification and Analysis (ATVA)*, National Taiwan University, December 10-13, 2003, pp 77– 86.

[10] X. Liu, "Identification and Check of Inconsistencies between UML Diagrams", *Journal of Software Engineering and Applications (JSEA))*, vol. 6, 2013, pp. 73-77.

[11] M. Saaltink, "The Z/EVES System", *ZUM'97: The Z Formal Specification Notation — 10th International Conference of Z Users Reading*, Springer, LNCS, vol. 1212, 1997, pp. 72-85.