A Multi-View Approach for Formalizing UML State Machine Diagrams Using Z Notation

KHADIJA EL MILOUDI, AZIZ ETTOUHAMI
LCS Laboratory, Faculty of Sciences,
University Mohammed V-Agdal
Rabat, MOROCCO
elmiloudi.khadija@gmail.com, ettouhami.aziz@gmail.com

Abstract: - Due to the missing formal foundation of UML, the semantics of a number of UML constructs is not precisely defined. Based on our previous work on formalizing class and sequence diagrams, a method for transforming a subset of UML state machine diagram into Z specification is proposed for the purpose of formally checking consistency in multi view modeling. The consistency of the resulting specification is guaranteed by providing a set of well-formedness and consistency rules. It is worth noting that our multi view approach is the first work on state machine diagram formalization based on Z notation. Our approach is illustrated using an example taken from the literature.

Key-Words: - UML State Machine Diagram, Z, Formal Methods, Consistency Checking, Multi View Modeling.

1 Introduction

Currently, UML [1, 2] is widely used during software design phases. However, when using UML in multi view modeling context, models cannot be verified and analyzed formally because of the lack of formal semantics. This requires a semantics specification which captures, in a precise way, both the structural and the dynamic features of modeled system.

Z [3] is a formal specification language based upon set theory and mathematical logic which provides formal foundation to analyze semantics and verify correctness. This paper presents our approach of formal modeling and validation for the UML State machine diagram using Z. The formalization is based on our previous work on transforming UML class and sequence diagram into Z specification in order to guarantee a multi view consistency [4, 5]. The resulting specification can then be analyzed by Z tools and hence formally prove or disprove the system safety.

This paper is organized as follows. Section 2 provides a description of related work along the lines of our motivation. Section 3 presents our approach in formalizing UML state machine diagram using Z notation. Section 4 overviews a set of well-formedness and consistency rules handled by the proposed model. Examples are offered to demonstrate the approach. Finally, Section 5 concludes the paper and outlines some future directions of our work.

2 Related Work

In fact, several studies have been undertaken to formalize UML diagrams. Especially, there has been much interest in formalizing UML state machine diagram to improve its shortfalls.

Tian and Gu [6] presented an approach of formal modeling and validation for software process, which transforms UML models based on Rational Unified Process (RUP) to Colored Petri Nets (CPN) and uses CPN tools to investigate the behavior of modeled system. A new approach for modeling state-chart Diagrams in B is proposed by Ledang and Souquières [7]. Only the modeling of UML concepts in B is considered. The problem of analyzing the derived B specification is not treated. Meng *et al.* [8] provided a formalization for UML state machine diagrams in the RAISE specification language RSL. McUmber and Cheng [9] introduced a general framework for formalizing a subset of UML diagrams in terms of different formal languages based on a mapping between metamodels describing UML and a formal language. UML is formalized in terms of Promela. Latella *et al.* [10] set the basis for the development of a formal semantics for UML state machine diagrams based on Kripke structures. A mapping of state machine diagrams to the intermediate format of extended hierarchical automata is proposed and then an operational semantics for these automata is defined. Our approach does not require an intermediate format, the diagram is directly translated into Z

specification. A formal semantics for a subset of state machine diagram is proposed by David *et al.* [11]. The subset and semantics are very close to the one supported by the tool Rhapsody. Mostafa *et al.* [12] presented a formalization of different UML diagrams using Z notation. Unlike our work, the consistency between them has not been treated. The work done by [13] presented an approach to transform up to three different UML behavioral diagrams (sequence, behavioral state machines, and activity) into a single Transition System to support model checking of software based on UML but inconsistencies between diagrams are not detected. In the paper [14], an automatic translation of UML behavioral diagrams into formal models is proposed in order to be verified by a symbolic model checker. Furthermore, the multi view verification has not been addressed by this work. A study done by [15] uses labelled transition systems as the semantic model to provide a formal semantics for UML state machines features. The paper [16] proposes a verification of UML state machine diagram by translation to UML-B using the Rodin platform and its automatic proof tools.

However, most of these works only focus on formalization of UML state machine diagrams without checking consistency and correctness in multi view modeling, and give up the advantage of unifying different models in one specification. Our approach is characterized by its clarity and conciseness. In this paper, we address the problem of modeling UML state machine diagrams using Z notation in multi view modeling, which has not been, so far, completely treated. This work forms a continuation of the previous work on formalizing UML class and sequence diagrams in Z [4, 5].

3 Z Formalization of UML State Machine Diagram

3.1 Overview of UML state machine diagrams

UML state machine diagram is an important component of UML for specifying the dynamic behavior of systems. Each state machine diagram basically consists of the states an object can occupy and the transitions which make the object change from one state to another according to a set of well-formedness rules. Each transition is characterized by an event which is an invocation of an operation in this object and contains a guard and an optional action.

In this paper, we present a formalization of the UML state machine diagrams using Z notation. We will refer to a subset of UML state machine diagrams which, nevertheless, includes all the interesting concepts. Our goal is to check the consistency of views in multi-view modeling of object-oriented systems based on UML. We focus on three different views of a system, comprising class diagram, sequence diagram and state machine diagram. The semantics of state machine diagrams is analyzed as well as its relevant problems with class and sequence diagrams that are also formalized in Z. Full details on modeling class and sequence diagrams are given respectively in [4, 5]. The video on demand system (VOD), proposed by Lopez-Herrejon and Egyed [17] is used as an example to illustrate our approach.

Given an UML class diagram of the VOD system as shown in Figure 1. The class diagram presents three classes *Service*, *Streamer* and *Program* linked by associations. Operations are defined for each class.

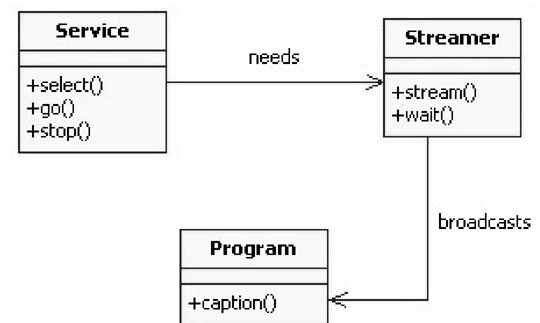

Fig. 1: Class diagram of VOD system

A sequence diagram in Figure 2 illustrates a call of method *select* in a *Service* object and a call of method *stream* from *Service* to *Streamer*.

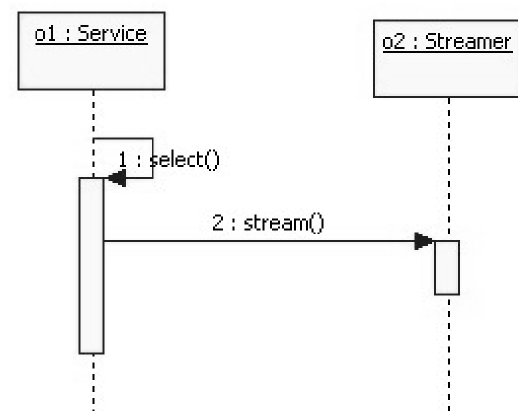

Fig. 2: Example of sequence diagram in VOD system

Figure 3 shows an example of an UML state machine diagram which may be used to specify the behavior of a *Service* object. Objects of class *Service* have two possible states: *Init* and *Streaming*. The change of the state is triggered by *select*, *go* and *stop*.

Fig. 3: State machine diagram of a Service Object

In the following, we discuss the semantics of state machine diagram by transforming it into a Z specification.

3.2 Translating UML state machine diagrams into Z specification

A state machine diagram is basically composed by a set of states and transitions among them according to a set of well-formedness rules.

We start by representation of the notion of states. The formal specification of state machine diagram could include the declaration:

$$[STATE]$$

introducing a basic type to represent the set of all states. Two variables of the type STATE are introduced by the following declaration to define the initial and final states of state machine diagram. The initial state denotes the default starting point for the state machine. The final state shows that the execution of the state machine has been terminated.

$$INITIALSTATE, FINALSTATE: STATE$$

For every object, the function statesOfObject returns the set of its states.

$$statesOfObject: OBJECT \rightarrow \mathbb{P}\ STATE$$

Another basic notion is transitions. Transitions are relationships among states. A transition designates that an object will change his current state to another. The first and second states are called source and target states. A transition can have multiple sources representing a join from multiple states as well as multiple targets in case of a fork to multiple states. A specific action is executed when an event occurs and therefore the guard is evaluated. If the guard is true, the transition may be enabled; otherwise, it is disabled. Formally, a transition is specified as follows:

$$\begin{array}{l} \textit{TRANSITION} \\ \hline event: EVENT \\ effect: EFFECT \\ guard: BOOL \\ source: \mathbb{P}\ STATE \\ target: \mathbb{P}\ STATE \\ \hline \end{array}$$

According to the current UML standard [1, 2], an event can be either a call event or a signal event. We define an event as a free type EVENT in which every element is either a SignalEvent or the result of applying the function OperationAsCallEvent to an element of type OP defined in our previous work on class diagram [4, 5]. OP is introduced as an enumerated set representing all the class operations. The signal event is not detailed in this paper; we specify it as a constant.

$$EVENT ::= OperationAsCallEvent \langle\langle OP \rangle\rangle \mid SignalEvent$$

A guard is defined as a boolean type, hence the necessity to introduce the free type BOOL.

$$BOOL ::= True \mid False$$

An effect specifies an optional behavioral, therefore we formalize it as a free type introducing a constant named Action and the constant NullEffect when the transition has no effect.

$$EFFECT ::= Action \mid NullEffect$$

The function isTargetOf, regarded as a function from a couple of state and transition to a state, is a total function: each state is related to exactly one state using a specific transition. This function will be used later in the state machine diagram definition.

$$isTargetOf: STATE \times TRANSITION \rightarrow STATE$$

Now, the semantics of the state machine diagram can be fully formalized using the previous definitions.

A state machine diagram specifies the behavior of a specific object. In Z, a schema consists of two parts:

a declaration of variables; and predicates constraining their values [18].
In the declaration part of the schema named StatechartDiagram, we introduce two variables: a variable Obj of type OBJECT is the object whose behavior is specified by the state machine diagram. The second variable named statechart specifies the components of the state machine diagram.
A state machine is defined by the set of states and the set of transitions relating between them; we distinguish between the states sources of transitions and states targets of transitions.
We introduce the variable statechart as a set of cartesian product consisting of all tuples of the form (STATE × TRANSITION × STATE) which respectively corresponds to the source state of transition, the transition and the target state.

StatechartDiagram

$Obj: OBJECT$
$statechart: \mathbb{P}\ (STATE \times TRANSITION \times STATE)$

$\forall\ Source, Target: STATE; Transition: TRANSITION$
$\bullet\ \{(Source, Transition, Target)\} \subseteq statechart$
$\wedge\ Target = isTargetOf\ (Source, Transition)$
$\wedge\ Source \in Transition.source$
$\wedge\ Target \in Transition.target$
$\wedge\ Source \in statesOfObject\ Obj \setminus \{FINALSTATE\}$
$\wedge\ Target \in statesOfObject\ Obj \setminus \{INITIALSTATE\}$
$\wedge\ (Transition.event \neq SignalEvent$
$\Rightarrow OperationAsCallEvent \sim Transition.event$
$\in\ methodsOfObject\ Obj$
$\wedge\ (\exists\ seqDiagr: SequenceDiagram;\ o: OBJECT$
$\bullet\ \langle(o, Obj,$
$OperationAsCallEvent \sim Transition.event)\rangle$
$\text{in}\ seqDiagr.Messages))$
$\wedge\ (Source = INITIALSTATE$
$\Rightarrow \#\ (outgoings\ Source) = 1$
$\wedge\ \#\ (incomings\ Source) = 0)$
$\wedge\ (Target = FINALSTATE$
$\Rightarrow \#\ (outgoings\ Target) = 0)$

The predicate part states that source and target of transitions must belong to the set of states of the object whose behavior is described by the state machine diagram. This predicate also denotes that an initial state can never be a target of a transition. Similarly, a final state cannot be a source of transition.

Applying this formalization to the example of VOD system in Figure 3, the state machine diagram of Service object will be described by the following set:
Statechart={(INITIALSTATE , select , Init),
(Init , go , Streaming),
(Streaming , stop , FINALSTATE)}

Therefore, the predicate stating that the source is never a final state and the target is never an initial state is verified.

An initialization schema is provided to define the initial value of the state machine diagram. A statechart is initially an empty set.

$$StateDiagramInit\ \widehat{=}\ [StatechartDiagram' \mid statechart' = \varnothing]$$

To check that the components of the state machine diagram are consistent, it is enough to establish that an initial state machine exists and hence also that at least one state machine exists fulfilling the requirements defined in the predicate part of the schema statechartDiagram.

theorem *InitIsOk*
$\exists\ StatechartDiagram' \bullet\ StateDiagramInit$

In Z, if a component represents an input, then its name should end with a query (?) [18]. The operation of changing a state requires two inputs: The current state of the object, and the chosen transition. We model these as two input components currentState? and transition?, of types STATE and TRANSITION, respectively.
The operation of changing state is described by:

ChangeState

$\Delta StatechartDiagram$
$currentState?: STATE$
$transition?: TRANSITION$

if $transition?.guard = True$
then $statechart'$
$= statechart$
$\cup\ \{(currentState?, transition?,$
$isTargetOf\ (currentState?, transition?))\}$
else $statechart' = statechart$

The effect of this operation is defined only when the guard of the transition is satisfied.

Once a UML state machine diagram is translated into a Z specification, the multi view consistency can be analyzed using Z tools. The predicate part of the schema StatechartDiagram will be discussed in details in Section 4 by means of a series of well-formedness rules.

4 Well-Formedness Rules in Multi View Modeling

In order to ensure the correctness of a state machine diagram and its consistency with class and sequence diagrams in multi view modeling, a set of well-formedness and consistency rules must be satisfied. We provide through the proposed model the formalization of these well-formedness rules using Z notation. We used published well-formedness rules to show the effectiveness of our model.

4.1 Intra-view well-formedness rules: states and transitions

Two additional functions that return respectively the set of transitions departing from and entering a specific state are used in the formal definition of the consistency rules.

$$incomings: STATE \longrightarrow \mathbb{P}\ TRANSITION$$
$$outgoings: STATE \longrightarrow \mathbb{P}\ TRANSITION$$

$$\forall s: STATE \bullet incomings\ s = \{ t: TRANSITION \mid t.target = s \}$$
$$\forall s: STATE \bullet outgoings\ s = \{ t: TRANSITION \mid t.source = s \}$$

Rule 1: A final state cannot have any outgoing transitions.

This rule is represented as a predicate in the schema statechartDiagram by the following Z expression using the outgoings function defined above.

$$(Target = FINALSTATE \Rightarrow \#(outgoings\ Target) = 0)$$

Rule 2: An initial state can have at most one outgoing transition and no incomings transitions. The formalization of this rule is similar to rule 1.

$$(Source = INITIALSTATE \Rightarrow \#(outgoings\ Source) = 1 \wedge \#(incomings\ Source) = 0)$$

4.2 Consistency rules between state machine diagrams and class diagrams

A state machine diagram can show the different states of an object also how an object changes from one state to another using transitions. Objects and states in state machine diagrams are related to class diagrams. Therefore, some consistency rules between class diagram and state machine diagram must be satisfied to ensure a multi view consistency.

Rule 3: an object that the state machine diagram describes must correspond to an instance of a class in class diagrams [19].

To express this consistency rule, a Z theorem is provided. The following theorem states that the object whose behavior is specified by the state machine diagram must belong to the set of objects of an existing class defined in class diagram.

theorem *consistencyStateAndClassDiagram*
$$\forall s: StatechartDiagram \bullet \exists class: CLASS \bullet s.Obj \in ObjectsOfClass\ class$$

The function ObjectsOfClass returns for each class the set of its instances. CLASS and OBJECT are defined in the class diagram formalization. More details are available in [4].

$$ObjectsOfClass: CLASS \longrightarrow \mathbb{P}\ OBJECT$$

Rule 4: if the event related to a transition in state machine diagrams is to call an operation of a class, the operation must be defined as an operation in owner's class [17]. The relation inverse of the function OperationAsCallEvent is used to reach the operation used in the event definition. This operation must belong to the set of operations corresponding to the object Obj.

$$(Transition.event \neq SignalEvent \Rightarrow OperationAsCallEvent^{\sim}\ Transition.event \in methodsOfObject\ Obj$$

The function methodsOfObject defined in our previous work on sequence diagram formalization

[5] returns the set of the class operations corresponding to each object.

4.3 Consistency rules between state machine diagrams and sequence diagrams

As one of UML behavioral diagrams, sequence diagrams illustrate object interaction. A consistency problem may occur caused by the fact that some components of the sequence diagram may be described by more than one diagram. Hence, the consistency of the system should be checked.
We define the semantics of a state machine diagram in the context of a sequence diagram that is also formalized.
Rule 5: if an event in state machine diagram is to call an operation, the operation should be a message in sequence diagram.
The sequence diagram is previously defined in [5] 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 operation invoked.

Therefore, in order to ensure the consistency between state machine and sequence diagrams, the operation invoked by the call event must appear in a message of an existing sequence diagram.

$$
\begin{aligned}
&(\mathit{Transition.event} \neq \mathit{SignalEvent}\\
&\Rightarrow (\exists\ \mathit{seqDiagr: SequenceDiagram;\ o: OBJECT}\\
&\quad \bullet \langle(\mathit{o,\ Obj.\ OperationAsCallEvent} \sim \mathit{Transition.event})\rangle\\
&\qquad\qquad \text{in } \mathit{seqDiagr.Messages}))
\end{aligned}
$$

As shown above, this predicate is included in the predicate part of the schema statechartDiagram in order to guarantee the multi view consistency.
Checking these properties for the class and sequence diagrams of VOD system introduced respectively in Figure 1 and Figure 2, the well formedness rules are satisfied. The operations invoked by the state machine diagram in Figure 3 are defined by the class Service and used as messages in sequence diagram.
The set of operations of the object o1 is as follows:

methodsOfObject o1={select, go, stop}

The sequence diagram in Figure 2 illustrates a call of method select in an object o1 of type Service. The operation select is also invoked by the transition which makes the object of type Service change from the initial state to the state Init. Therefore, the well-formedness rule 6 is satisfied. The operations go and stop invoked in the state diagram in Figure 3 are not used by the sequence diagram illustrated in Figure 2. In this case, to guarantee the multi view consistency, the existence of another sequence diagram which calls these operations must be ensured. Often when this is the case, examination of the model helps understanding the problem and therefore suggesting a correction.

5. Conclusions and Future Work

The goal of this paper is to overcome the main limitations of UML state machine diagrams semantics in multi view modeling. It provides a formal semantics of state machine diagrams according to class and sequence diagrams formal model previously published. Numerous well-formedness and consistency rules are provided to ensure the multi view consistency. The main benefits of our approach are the conciseness and clarity of the formal model providing one of the first Z formal specifications for the state machine diagram in multi view context. The resulting Z specification allows the definition of a precise and unambiguous semantics of UML state machine diagram. All the presented specifications were thoroughly type-checked using the Z/EVES system [20]. The Z/EVES immediately uncovers such inconsistencies. As future work, we aim at considering a large subset of UML state machine diagrams and checking consistency with other UML diagrams such as use case diagrams. More complex case studies in several domains are in our targets. In addition, we are currently extending the support tool for automatically translating class diagrams into Z specifications [4] to take into account UML behavioral diagrams.

References:

[1] OMG Unified Modeling Language Infrastructure 2.4.1. Available from: http://www.omg.org/spec/UML/2.4.1/Superstructure. Accessed 23March 2014.

[2] Booch, G.; Rumbaugh, J.; Jacobson, I., *The Unified Modeling Language User Guide,* Addison Wesley Longman. ISBN: 0-201-57168-4, 1998.

[3] Spivey, J.M., *The Z Notation: A Reference Manual*. 2nd Edn., England: J. M. Spivey, Oriel College, Oxford, OX1 4EW, 1998.

[4] El Miloudi, K.; El Amrani, Y.; Ettouhami, A., An Automated Translation of UML Class

Diagrams into a Formal Specification to Detect UML Inconsistencies. *Sixth International Conference on Software Engineering Advances*. Barcelona, Spain, 23-29 October 2011, pp. 432-438.

[5] El Miloudi, K.; El Amrani, Y.; Ettouhami, A., Using Z Formal Specification for Ensuring Consistency in Multi-View Modeling, *Journal of Theoretical and Applied Information Technology*. Vol. 57, 2013, pp. 407-411.

[6] Tian, B.; Gu, Y., Formal Validation for Software Modeling, *International Journal of Computer Science Issues*. Vol. 10, 2013, pp. 308-312.

[7] Ledang, H.; Souquières, J., New Approach for Modeling State-chart Diagrams in B, Technical Report A01-R-082, Laboratoire Lorrain de Recherche en Informatique et ses Applications, 2001.

[8] Meng, S.; Naixiao, Z.; Aichernig, B.K., The Formal Foundations in RSL for UML Statechart Diagrams, UNU/IIST Technical Report 299, 2004.

[9] McUmber, W.E.; Cheng, B. H. C., A General Framework for Formalizing UML with Formal Languages. *Proceedings of the 23rd International Conference on Software Engineering (ICSE01)*, Toronto, CA, May 2001, pp. 433-442.

[10] Latella, D.; Majzik, I.; Massink, M., Towards a Formal Operational Semantics of UML Statechart Diagrams. In: Ciancarini, P.; Fantechi, A.; Gorrieri, R. (eds.) *Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems*, 1999, pp. 331-347. Kluwer Academic Publishers.

[11] David, A.; Deneux, J.; d'Orso, J., A Formal Semantics for UML Statecharts, Technical Report 2003-010, Uppsala University, 2003.

[12] Mostafa, A. M.; Ismail, M. A.; EL-Bolok, H. ; Saad, E. M., Toward a Formalization of UML2.0 Metamodel using Z Specifications, *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing,* 2007.

[13] Santos, L. B. R. D.; Júnior, V. A. D. S.; & Vijaykumar, N. L., Transformation of UML Behavioral Diagrams to Support Software Model Checking. arXiv preprint arXiv:1404.0855, 2014.

[14] Fernandes, F.; Song, M., UML-Checker: An Approach for Verifying UML Behavioral Diagrams. *Journal of Software*, Vol. *9*(5), 2014, pp. 1229-1236.

[15] Liu, S.; Liu, Y.; André, E.; Choppy, C.; Sun, J.; Wadhwa, B.; Dong, J. S., A formal semantics for complete UML state machines with communications. In *Integrated Formal Methods,* 2013, pp. 331-346, Springer Berlin Heidelberg.

[16] Snook, C.; Savicks, V.; Butler, M., Verification of UML models by translation to UML-B. In *Formal Methods for Components and Objects,* 2012*,* pp. 251-266. Springer Berlin Heidelberg.

[17] Lopez-Herrejon,R.E.; Egyed, A., Detecting Inconsistencies in Multi-view Models with Variability, *6th European Conference on Modelling Foundations and Applications* (ECMFA), 2010, pp. 217-232.

[18] Woodcock, J.; Davies, J., *Using Z: Specification, Refinement, and Proof,* Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[19] Liu, X., Identification and Check of Inconsistencies between UML Diagrams. *Journal of Software Engineering and Applications,* Vol. 6, 2013, pp. 73-77.

[20] Meisels, I., Software Manual for Windows Z/EVES Version 2.3. ORA Canada Technical Report TR-97-5505-04h, 2004.