Proceedings of MUCEET2009
Malaysian Technical Universities Conference on Engineering and Technology
June 20-22, 2009, MS Garden,Kuantan, Pahang, Malaysia

MUCEET2009

Semantic Rules of UML Specification

Noraini Ibrahim and Rosziati Ibrahim

***Abstract*— Modeling of a system is an essential process in software development lifecycle (SDLC). It will produce a system artifact called a system model. In object–oriented based software development, a system model can be developed by using Unified Modeling Language (UML). UML is a modeling language for specifying, constructing, and documenting the artifacts of systems. It consists of 13 diagrams that can be used to describe the different views of a system. Each diagram has its own syntax and semantics. The syntax or abstract syntax is the notations for each element of the diagrams, whereas the semantics is the meaning of the notations. The huge complexity of UML specification that content multi diagrams and notations, and lack of formal semantics decrease the quality of system models produced. It will lead to wrong interpretations and inconsistency between models. Therefore, a precise meaning of UML diagrams is very important in order to have a common understanding of their meaning. Formalization of the semantics of UML specification is important in order to provide the consistency of the system models. This paper provides an overview of the semantics rules of UML specification and suggests an approach to formalize these semantics rules.**

***Keywords*:** UML semantics, formalization, consistency and rules

I. INTRODUCTION

Process of software development is iterative and incremental. It makes up the life cycle of a system. Each cycle will produce a number of different artifacts. One of them is model.

Noraini Ibrahim is doing PhD at Universiti Tun Hussein Onn Malaysia (UTHM), Parit Raja, 86400 Batu Pahat, Johor (e-mail:noraini@uthm.edu.my)

Assoc. Prof. Dr. Rosziati Ibrahim is a lecturer at Universiti Tun Hussein Onn Malaysia (UTHM), Parit Raja, 86400 Batu Pahat, Johor. She is now with the Department of Software Engineering, Faculty of Information Technology and Multimedia (e-mail:rosziati@uthm.edu.my)

A model is an abstract representation of a system, constructed to understand the system prior to building or modifying it. Most of the modeling techniques involve graphical languages. One example of modeling techniques is Unified Modeling Language (UML).

A system model can provide structure for problem solving; experiment to explore multiple solutions and abstractions to manage complexity [12]. Models make it easier to express complex ideas, for example, an architect builds a model to communicate ideas more easily to clients. Models reduce complexity by separating those aspects that are unimportant from those that are important. The models should have traceability links or in other words they must be consistent.

UML is a diagramming language. For each elements or constructs have its own notation or abstract syntax and semantics. Abstract syntax is rules by which language elements (for example, words) are assembled into expressions (for example, clauses). While semantics is rules by which syntactic expressions are assigned meanings. This is where ambiguity comes out.

The huge complexity of UML that contents multi diagram notation and lack of formal semantics decrease the quality of system models produced. Precise meaning of UML diagrams is very important in order to have a common understanding of their meaning.

The rest of this paper is organized as follows: the review of UML is in Section 2, Section 3 review on UML Semantics. Section 4 discusses related work and Section 5 concludes the paper.

II. REVIEW OF UML

Unified Modeling Language (UML) is a language for specifying, constructing, visualizing, and documenting the software system and its components. UML is an object modeling and specification language used in software engineering. UML includes a set of graphical notation techniques to create abstract models of specific systems.

Currently, UML specifies 13 UML diagrams. They are divided by two categories: Structure and Behavior Diagram. Structure diagram consists of class diagram, composite structure diagram, component diagram, deployment diagram, object diagram and package diagram. While behavior diagram consists of activity diagram, sequence diagram, communication diagram, interaction overview

diagram, timing diagram, use case diagram and state machine diagram.

A system model does not need to have the whole collection of UML diagrams. The most popular UML diagrams used in industry are use case diagram, sequence diagram, class diagram and statechart diagram.

A use-case diagram is a graph of actors, a set of use cases, communication (participation) associations between the actors and the use cases, and generalization among the use cases. Each use case shows flow of events through the system, whereas an actor is a user playing a role with respect to the system [15]. An example of use case diagram is shown in figure 1.

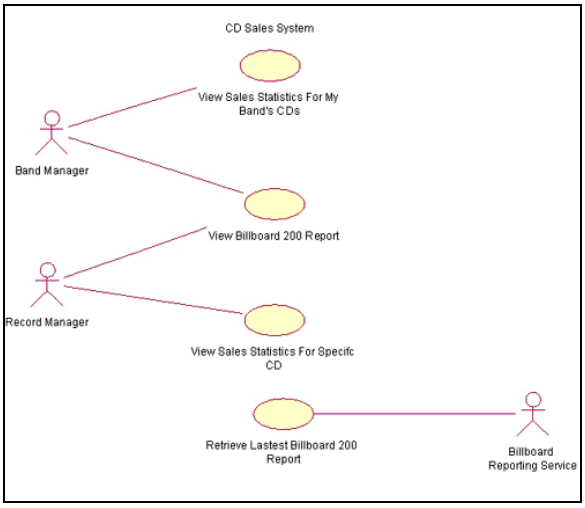

Figure 1: Use Case Diagram.

Based on figure 1, there are 3 actors, which are `Band Manager`, `Record Manager` and `Billboard Reporting Service`. Actor `Band Manager` has association (communicate) with use cases named `View Sales Statistics For My Band's CDs` and `View Billboard 200 Report`. While actor `Record Manager` interact with `View Billboard 200 Report` and `View Sales Statistics For Specific CD`. Whereas, actor `Billboard Reporting Service` associate with `Retrieve Lastest Billboard 200 Report`.

In a normal practice, after we draw a use case diagram, for each use case we detail out with sequence diagrams. A sequence diagram shows how objects communicate with each other in terms of a sequence of messages. It also indicates the lifespan of objects relative to those messages [15]. Figure 2 shows the example of sequence diagram.

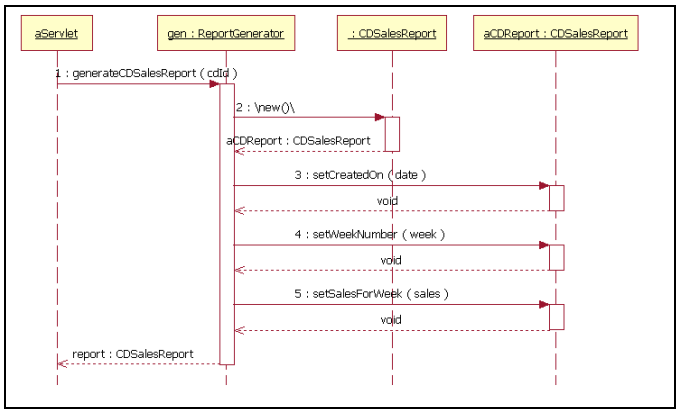

Figure 2: Sequence Diagram.

Based on figure 2, `aServlet` object sends a message labeled as `generateCDSalesReport` to the `ReportGenerator` class instance named `gen`. It means that the `ReportGenerator` object implements this message handler. On the `generateCDSalesReport` message label containing `cdId` in parentheses, which means that `aServlet` is passing a variable named `cdId` with the message. When `gen` instance receives a `generateCDSalesReport` message, it then makes subsequent calls to the `CDSalesReport` class, and an actual instance of a `CDSalesReport` called `aCDReport` gets returned. The `gen` instance then makes calls to the returned `aCDReport` instance, passing it parameters on each message call. At the end of the sequence, the `gen` instance returns `aCDReport` to its caller aServlet.

From sequence diagrams, all the classes are gathered in class diagram. Class diagram is a collection of static modeling elements, such as classes and their relationships [15]. In figure 3, the inheritance relationship and two association relationships can be seen. The `CDSalesReport` class inherits from the `Report` class. A `CDSalesReport` is associated with one `CD`, but the `CD` class doesn't know anything about the `CDSalesReport` class. The `CD` and the `Band` classes both know about each other, and both classes can be associated to one or more of each other.

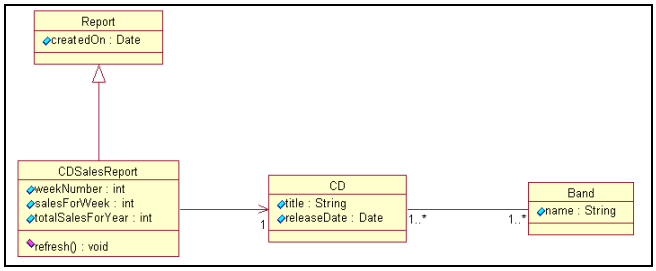

Figure 3: Class Diagram.

A state machine diagram on the other hand is used to show life for all objects belong to one class. When building state machine diagram, all behavior of all possible objects of a class must be considered. A single state machine diagram is developed for a class [16]. The example of statechart diagram is shown in figure 4.

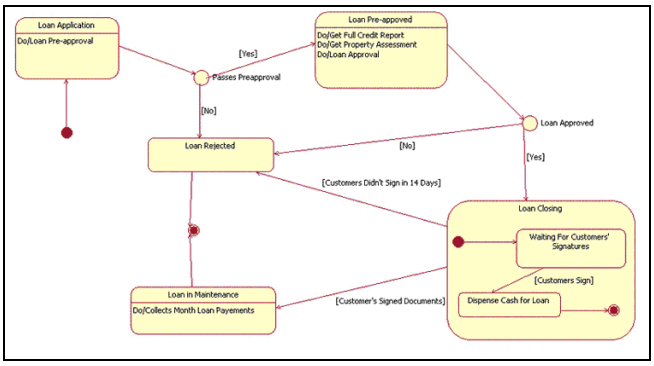

Figure 4: Statechart Diagram.

The notation set of the statechart diagram has five basic elements: the initial starting point, which is drawn using a solid circle; a transition between states, which is drawn using a line with an open arrowhead; a state, which is drawn using a rectangle with rounded corners; a decision point, which is drawn as an open circle; and one or more termination points, which are drawn using a circle with a solid circle inside it [14]. To draw a statechart diagram, begin with a starting point and a transition line pointing to the initial state of the class.

Based on figure 4, a loan processing begins in the `Loan Application` state. After `pre-approval` process is done, the transition will go either to `Loan Pre-approved` state or the `Loan Rejected` state. The decision is noted by the empty circle in the transition line. If the `loan approved`, the transition will go to `Loan Closing` state before going through the `Loan in Maintenance` state. The above examples of CD Sales are adapted from tutorial at the website [14].

III. REVIEW OF UML SEMANTICS

The current UML specification consists of two interrelated parts. They are UML Semantics and UML Notation. UML Semantics specifies the abstract syntax and semantics of UML object modeling concepts, since UML Notation is a graphic notation for the visual representation of the UML semantics. The abstract syntax for the UML Semantics is expressed using a small subset of the UML Notation. In addition, the UML Notation describes the mapping of the graphic notation to the underlying semantics. The two parts complement each other without duplicating functionality [17].

UML Semantics is divided into three layers; structural, behavioral and higher-level formalisms of UML: activities, state machines and interactions. Behavioral layer is caused by actions of structural entities from structural level. Communication between structural entities is called inter-object behavior base, whereas, the intra-object behavior base, addresses behavior occurring within structural entities. Action sub layer itself defines the semantics of individual actions.

Much effort has been given to gather UML semantics for various UML diagrams. Currently, the information relating to semantics is scattered throughout the standard document [13].

Semantics are categorized into three dimensions: internal, vertical and horizontal dimension [1], [2]. Internal deals with relationships between sub-models that coexist. For instance, an analysis model consists of an analysis class diagram, interaction diagram and collaboration diagrams. All the artifacts with a single model are related and must be compatible with each other. The vertical dimension considers relationship between models belonging to same iteration in different activities for example a design model realizing an analysis model. Whereas horizontal dimension considers relationships between artifacts belonging to the same activity in different iterations, for example a use case is extended by another use case.

IV. RELATED WORKS

There is an important number of works that giving a precise description of core concepts of UML and providing rules for analyzing their properties (for instance [3], [4], [5] and [6]). These works improve the precision of UML semantics without dealing with relationships between models. Researchers (for example [1], [7] and [8]) focus on relationship between models. While Egyed [9] is focusing to abstraction view.

Some researchers (such as [3], [4] and [6]) had gone through the semantics of statechart diagrams, while the semantics of class diagram can be found in [1], [5], [7] and [9]. Semantic of Sequence Diagram and Use Case Diagram can be seen in [2], [7] and [8].

UML is not a formal language. Many motivations are given to justify the importance of formalization. Formalization can bring clarity, consistency, correctness and enhancement to models [4]. Various ways are used by researchers to increase the precision of UML semantics.

Operational semantics is one of the ways to formalize some of the diagrams in UML Specification. In computer science, operational semantics is a way to give meaning to programs in a mathematically rigorous way [18]. It consists of a set of rules specifying how the state of an actual or hypothetical computer changes while executing a program. Each rule specifies certain preconditions on the contents of some components and their new contents after the application of the rule [19].

For example, Pons et. al. [2] use operational semantics to formalize the vertical relationship between use case diagram and collaboration diagrams and horizontal relationship of use cases. While Latella et. al. [3], using operational semantics to formalize statechart diagram. They have suggested three rules; the progress rules, the composition rule and the stuttering tools. Whereas, Lund et. al. [8] were suggesting rules for potential and mandatory choices in sequence diagram using operation semantics. It has been proved to be sound and complete with respect to a denotational semantics of UML [8]. But it is scoped only to sequence diagram. The statechart properties like the history mechanism, exit and entry actions has been formalized using operational semantics by Beeck [10].

The other way to formalize the semantics is denotation approach. This approach has three parts; a precise definition

of the syntactic domain, development of suitable semantic domain and definition of the semantic mapping. Unfortunately, Cengarle et. al. [5] said this approach was discontinued. Although, they are using this approach to formalize the semantics of UML class diagram.

Peng [4] was using algebraic specification to describe semantics of statechart. He comes out with static class translation rules, dynamic class translation rules and composite system translation rules. Whereas Zhang et. al. [6] were using graph transformation to specify behavioral semantic of statecharts.

V. CONCLUSION AND FUTURE WORKS

Even though, research on formalization of UML semantics had gone through for many years, there still need precise definitions of UML semantics. Current research more focus to partial semantics such as intra diagram consistency for statechart (such as [1] and [7]) and for sequence diagram (such as [2] and [8]). Not much effort focus on gathering complete semantics properties for the most frequent use of UML diagrams (use case, sequence, class and statechart diagram) and their traceability links. Therefore, we propose a formalization of UML semantics for these diagrams using operational semantics.

REFERENCES

[1] L. C. Briand, Y. Labiche and T. Yue, "Automated traceability analysis for UML model refinement," *Journal Information and Software Technology*, vol. 51, issue 2, pp. 512-517, February 2009.

[2] C. Pons, R. Giandini, G. Baum, J. L. Garbi, P. Mercado, "Specification and Checking Dependency Relations between UML Models," in UML and the Unified Process. Hershey: IGI Publishing, 2003, pp 237-253.

[3] D. Latella, I.Majzik and M. Massink, "Towards A Formal Operational Semantics of UML Statechart Diagrams," in Proceedings of the IFIP TC6/ WG6.1, Third International Conference on Formal Methods for Open Object-Based Distributed System (FMOODS), 1999, pp. 465.

[4] L. Peng, "Formalization of UML Using Algebraic Specifications," M.S. thesis, Ecole des Mines de Nantes, 2001.

[5] M. V. Cengarle, H. Gronniger, and B. Rumpe, "System Model Semantics of Class Diagrams," Technische Universitat Braunschweig, 2008.

[6] J. Kong, K. Zhang, J. Dong and D. Xu, "Specifying behavioural semantics of UML diagrams through graph transformations," *Journal Information and Software Technology*, vol. 82, issue 2, pp. 292-306, February 2009.

[7] Franck Xia, Gautam S. Kane, "Defining the Semantics of UML Class and Sequence Diagrams for Ensuring the Consistency and Executability of OO Software Specification," in 1st International Workshop on Automated Technology for Verification and Analysis Proceedings, 2003.

[8] M. S. Lund, K. Stolen, "A fully general operational semantics for UML sequence diagrams with potential and mandatory choice," Department of Informatics, University of Oslo, 2007.

[9] A. Egyed, "Semantic abstraction rules for class diagram," in Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE), Grenoble, France, 2000, pp. 301-304.

[10] M. v. d. Beeck, "A Structured Operational Semantics for UML-Statecharts," *Journal Software and Systems Modeling*, vol. 1, issue 2, pp. 130-141, December 2002.

[11] R. Eshuis, "Reconciling statechart semantics," *Science of Computer Programming,* vol 74, pp. 65-99, 2009.

[12] Adapted from the slide of Chris Kobryn, Co-Chair UML Revision TaskForce

[13] B. V. Selic, "On the Semantic Foundations of Standard UML 2.0," in Formal Methods for the Design of Real-Time Systems, vol. 3185/ 2004, Springer Berlin, 2004, pp. 181-199.

[14] D. Bell, "UML basics: An introduction to the Unified Modeling Language," IBM, Retrieved April 6, 2009 from http://www.ibm.com/developerworks/rational/library/769.html#fig5

[15] A. Bahrami, *Object Oriented Systems Development,* Singapore: Mc-Graw Hill, 1999.

[16] M. J. Chonoles and J. A. Schardt, UML 2 for Dummies, New York: Wiley Publishing, 2003.

[17] *UML Semantics*, Object Management Group, 1997.

[18] Wikipedia The Free Encyclopedia, Retrieved April 6, 2009 from http://en.wikipedia.org/wiki/Operational_semantics

[19] G. R. Semiotics, Dictionary.com, Retrieved April 6, 2009 from http://dictionary.reference.com/browse/operational%20semantics