FORMALIZATION OF WELL-FORMEDNESS RULES FOR UML USE CASE DIAGRAM

Noraini Ibrahim 1, Rosziati Ibrahim 2, and Dzahar Mansor 3

1,2 Department of Software Engineering, Faculty of Information Technology and Multimedia,
Universiti Tun Hussein Onn Malaysia
Parit Raja, 86400 Batu Pahat, Johor
3 Microsoft Malaysia,
Level 30, Tower 2, Petronas Twin Towers,
Kuala Lumpur City Centre, 50088 Kuala Lumpur.
email: {noraini 1, rosziati 2} @uthm.edu.my, dmansor@microsoft.com 3

ABSTRACT
Unified Modeling Language (UML) is a modeling language for specifying, constructing and documenting the artifacts of the systems. Similar to programming language such as C++ and Java, UML also consists of syntax and semantics. The UML syntax is the graphical notations to draw the UML diagrams and the semantics is the meaning of the notations. The semantics is grouped into two, static or well-formedness and dynamic. Currently, UML offers 13 diagrams for modeling any systems. The semantics of these 13 diagrams must give precise meaning. This is to ensure that the software modeler and system developer have improved common understanding towards the meaning of the diagrams. This can be achieved by employing well-formedness rules to address ambiguities and inconsistencies in UML representation. This paper discusses the well-formedness rules of one particular UML diagram, which is a use case diagram and provides an approach to formalize these rules.

Keywords: UML, well-formedness rules, use case diagram, semantics, formal method.

1 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.

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).

UML is a language for specifying, constructing, visualizing, and documenting the software system and its components. It includes a set of graphical notation techniques to create abstract models of specific systems.

Similar to programming languages, such as C++ and Java, UML also consists of syntax and semantics. According to an authorities that organized UML, Object Management Group [14], syntax is what constructs exists in the language and how the constructs are built up in terms of other constructs. While, semantics is categorized into two static (well known as well-formedness) and dynamic semantics. The static semantics in general is defined as how an instance of a construct should be connected to other instances to be meaningful, and the dynamic semantics is defined as the meaning of a well-formed construct.

Currently, UML specifies 13 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.

The huge complexity of UML that contains multi diagram notations 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 meanings of the diagrams are defined only if they are well-formed. It means that the UML diagrams must be at first fulfill the rules defined in the static semantics. The document for well-formedness rules of UML specification can be found in [15].

There has been some studies in formalizing UML semantics over the years such as on State
Machine diagrams (for instance in [9], [17] and [7]), Class Diagram (for example in [2], [3], [21] and [4]) and Sequence Diagram [21] and [12]. However, most of these address dynamic semantics without much regard on the formalization of the well-formedness rules of UML specification.

In this paper, we will focus on formalization of well-formedness rules of UML use case diagram. Our work is based on the set theory. We believe that work in this direction is essential towards to enhance UML usage to building consistent software.

The rest of this paper is organized as follows. The discussion on related works is in Section 2 and review of UML use case diagram is in Section 3. Section 4 presents the well-formedness rules of use case diagram and Section 5 concludes the paper.

2 RELATED WORKS

Even though use case diagram is valuable in requirement analysis process [20], there are few research works about use case diagram and its formalization. Liu et al. in [11] and [10] show the formalization of use case model in terms of dynamic semantic. Overgaard and Palmkvist [16] also formalize the dynamic semantics of use cases and their relationships using operational semantic. Pons et al. [18] use operational semantics to formalize the vertical relationships between use case diagram and collaboration diagrams and horizontal relationship of use cases.

Even though well-formedness is very important in order to achieve the rightness of UML diagrams, there are still lack of researches in formalization of the well-formedness rules. Addition to that, a research conducted by Labiche [8] notifies that awareness of well-formedness is low among practitioners and students. He also claims that tools do not necessarily enforce well-formedness rules. However, there is a need to develop a tool that insist on the rules.

Ha et al. [6] in their work propose several checking rules for consistency checking of well-formedness rules of nine UML diagrams. They demonstrate the consistency checking between sequence diagram and object diagram.

Mostafa et al. [13] provide formal specification for use case diagram using Z specification language. They also provide formal syntax for class diagram and state machine diagram.

Sengupta et al. [19] also suggest a methodology for formalizing use case diagram using Z notation. They focus on dynamic semantic of use case diagram.

While in [5], Eichelberger has proposed a set of layout and drawing guidelines to describe structural, semantic and drawing rules for layout of use case diagrams.

3 UML USE CASE DIAGRAM

A use case diagram is an UML diagram used to provide an overview of all or part of the system requirements. It is also used to communicate the scope of a development project. It shows the relationships among actors and use cases within a system.

A use-case diagram is a kind of behavioral diagram. It 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 [1]. An example of use case diagram is shown in Figure 1.

The figure shows that there are three (3) actors and four (4) use cases. Actors are Band Manager, Record Manager and Billboard Reporting Service and use cases are View Sales Statistics For My Band’s CDs, View Billboard 200 Report, View Sales Statistics For Specific CD and Retrieve Latest Billboard 200 Report.

Figure 1 also shows that Band Manager has association to View Sales Statistics For My Band’s CDs and View Billboard 200 Report, Record Manager has association to View Billboard 200 Report, View Sales Statistics For Specific CD and Billboard Reporting Service associate to Retrieve Latest Billboard 200 Report.

Details of each elements of use case diagram are described in the following sub-sections (Figure 1).

3.1 Actors

Actor is a person, organization or system that plays a role in one or more interaction with the system. The primary actor of the system is placed in the top left corner of the diagram. Actors must be drawn outside edges of a use case diagram. Actors must be named with singular nouns that accurately reflect its role. <<system>> stereotype is used to indicate system actors.
3.2 Use Case

A use case describes a sequence of actions that provide a measurable value to an actor. A use case is drawn as horizontal ellipse on a UML use case diagram. It must have a name consists of verb and noun.

3.3 Communication

There are several types of communication that may appear on a use case diagram. They are an association between an actor and a use case, association between two use cases, a generalization between two actors and a generalization between two use cases.

Associations are depicted as lines connecting two modeling elements with an optional open-headed arrowhead on one end of the line, indicating the direction of the initial invocation of relationship. Associations do not present information. They indicate that an actor is somehow involved with a use case.

<<include>> relationship is just like calling a function or invoking an operation within source code. It is modeled as dependencies between use cases and therefore a dashed line is used. The association end is at the included use case(s). <<include>> relationship points from base use case to the included use case.

<<extend>> relationship depict important alternate flows graphically by making them into their own use cases. The direction of the <<extend>> arrow points from the new extension use case to the base.

Generalizations are depicted as closed-headed arrows pointing toward the more general modeling elements. Potential actors may also be generalized. But when produce use case diagram, do not connect the included use case directly to the generalized actor. Primary actors of an included use case are implicitly the actors of base use case(s). Adding a connection to the generalized actor just adds another actor to the use case – one actor too many. This common diagramming error indicates that another actor instances is required to execute the use case, when it is not.

4 WELL-FORMEDNESS RULES OF USE CASE DIAGRAM

This section summarizes the formal syntax of modeling notations of use case diagram.

Definition 1. A use case diagram, \( U \) consists of six elements \( A, C, S, G, \text{------> and <-----} \) where
\[
\begin{align*}
A &= \{a_1, a_2, a_3, \ldots, a_m\} \text{ is finite set of actors;} \\
C &= \{c_1, c_2, c_3, \ldots, c_m\} \text{ is finite set of use cases;} \\
S &= \{s_1, s_2, s_3, \ldots, s_m\} \text{ is finite set of associations;} \\
G &= \text{generalization;} \\
\text{------> is <<include>> relationship;} \\
\text{ <----- is <<extend>> relationship.}
\end{align*}
\]

Following are a part of the rules:

4.1 Association

- **Definition 2.** Each actor must be associated/involved with at least one use case and every use case is involved with at least one actor.
\[
\forall a \in A, \exists c \in C, S(a) = c \quad (1)
\]

- **Definition 3.** Actors are not allowed to interact (associate) with other actors.
\[
\forall a_i, a_j \in A, S(a_i) \neq a_j, 1 \leq i, j \leq m \quad (2)
\]

- **Definition 4.** Association can only happen between actors and use cases, where \( S \subset A \times S \) (3)

4.2 Generalization

Generalization can only happen between actors or between use cases.

- **Definition 5.** Generalization between actors. The generalization is acyclic.
Definition 6. Generalization between use cases. The generalization is acyclic.

G is a relation on C, where cGd iff G is irreflexive and asymmetric, ∀c,d ∈ C.

\[ G \subseteq C \times C \]  

(5)

If in \( G, c_i \rightarrow c_j, i \neq j \), then

\[ c_j \not\rightarrow c_i \]

Let \( C = \{c_1, c_2, c_3\} \), then

\[ G = \{(c_1,c_2),(c_1,c_3),(c_2,c_3)\} \]

4.3 <<include>> relationship

Definition 7. <<include>> relationship denoted by \( \rightarrow \). It is identified by

\( \rightarrow \subseteq C \times C \), as

\( c_m \rightarrow c_n \)

where \( c_m \) is base use case and \( c_n \) is included use case

4.4 <<extend>> relationship

Definition 8. <<extend>> relationship denoted by \( \leftarrow \). It is identified by

\( \leftarrow \subseteq C \times C \), as

\( c_k \leftarrow c_p \)

where \( c_i \) is base case use and \( c_j \) is new extension use case

4.5 Example

Based on Figure 2, there are some errors. According to (1), \( \forall a \in A, \exists c \in C, S(a) = c \) where each A must be associated to at least one C, but in Figure 2 A named Customer do not associate to any use cases. While, the figure shows that A named User and Customer associated together. It contrast to (2) whereby \( S(a_i) \neq a_j \). The other error is the <<include>> relationship between Withdraw and Card Identification use cases. The arrow should point from C of Withdraw and Card Identification as stated in (6).

Figure 2. Example of Wrong Use Case Diagram

Figure 2 then being modified according to the definitions in Section 5. The corrected use case diagram is showed in Figure 3. Based on (4) where \( G \subseteq A \times A \), the A named Customer is generalized as a User. While for Transfer Funds and Withdraw generalized to Perform ATM Transaction use cases, it is also fulfilled (5), \( G \subseteq C \times C \).

Figure 3. Modified Use Case Diagram

5 CONCLUSION AND FUTURE WORKS

Existing research on UML semantics focuses on dynamic semantics. Formalization of UML well-formedness rules for this diagram will help in improve consistence understanding of UML models between interested parties. In this paper, we present
an approach to formalize UML well-formedness rules for use case diagram. We intend to formalize all other UML diagrams including the formalization of consistency check among UML diagrams and then to develop an automated tool to support the rules as many as possible.

REFERENCES

study on use case and class diagrams in UML. Requirements Engineering, Volume 9, (Number 4 / November, 2004), 229-237.