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
[3] Microsoft Malaysia.
email : noraini@uthm.edu.my [1], rosziati@uthm.edu.my[2], dmansor@microsoft.com[3]

Abstract – *Unified Modeling Language (UML) is used for specifying and documenting the artifacts of the systems. Similar to programming language, UML also consists of syntax and semantics. The 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 where precise meaning to the diagrams is given. This is to ensure that system developers have a common understanding towards the meaning of the diagrams. In order to achieve that, the well-formedness rules of system models must be correct before the dynamic semantics are fulfilled. If they can be achieved, the ambiguities and inconsistencies of UML diagrams can be improved. Therefore, a more precise meaning of well-formedness rules of UML diagrams is very important. This paper discusses the well-formedness rules of UML use case diagram and provides an approach to formalize these rules.*

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

1. Introduction

Modeling of a system is an essential process in software development lifecycle. It will produce a system artifact called a system model.

A system model can provide structure for problem solving; experiment to explore multiple solutions and abstractions to manage complexity. 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.

In object-oriented based software development, a system model can be developed by using Unified Modeling Language (UML). UML is a language for specifying, constructing, visualizing, and documenting the software system and its components. It is an object modeling and specification language used in software engineering. It 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.

Similar to programming languages, such as C++ and Java, UML consists of syntax and semantics. According to an authority that organized UML, Object Management Group (OMG, 1997), 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*. The meaning of a description written in the language is defined only if the description is well formed. It means that the description must fulfill the rules defined in the static semantics.

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

In this paper, we will focus to UML use case diagram as it is used as driving force in software development. As other UML diagrams, it has description of their well-formedness and dynamic semantics using combination of graphical notation, formal and natural language. As a mean of communication between user and developer, it must be precise. It is to ensure the diagram is interpreted correctly.

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, Section 5 on review on well-formedness rules handling by modeling tools, Section 6 on formalization of well-formedness rules of use case diagram and Section 7 concludes the paper.

2. Related Works

Even though use case diagram is valuable in requirement analysis process (Siau & Lee, 2004), there are few research works about use case diagram and its formalization. Liu et al. in (Liu, He, & Li, 2001) and (Li, Liu, & He, 2001) show the formalization of use case model in terms of dynamic semantic. Overgaard and Palmkvist (Övergaard & Palmkvist, 2004) also formalize

the dynamic semantics of use cases and their relationships using operational semantic. Pons et al. (Pons *et al.*, 2003) 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 lacks of researches in formalization of the well-formedness rules.

Addition to that, a research conducted by Labiche (Labiche, 2009) notifies that awareness of well-formedness is low among practitioners and students. He also claims that tools do not necessarily enforce well-formedness rules. So, there is a need to develop a tool that insist on the rules.

Ha et al. (Ha & Kang, 2008) 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.(Mostafa *et al.*, 2007) 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 (Sengupta & Bhattacharya, 2008) also suggest a methodology for formalizing use case diagram using Z notation. They focus on dynamic semantic of use case diagram.

While Eichelberger (Eichelberger, 2008) 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 (Bahrami, 1999). An example of use case diagram is shown in Figure 1.

The figure shows that there are two (2) actors and four (4) use cases. Actors are `User` and `Customer` and use cases are Withdraw, Perform ATM Transaction, Card Identification and Transfer Funds.

Figure 1 also shows that Customer is generalized to User. User has association to Withdraw. While Withdraw and Transfer Funds are include Card Identification. Withdraw and Transfer Funds are generalized to Perform ATM Transaction.

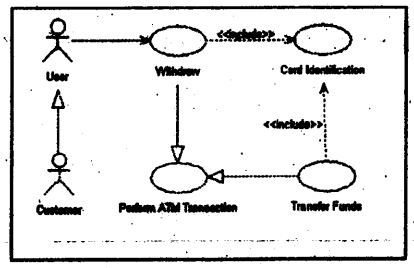

Figure 1. Use Case Diagram

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

a. 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.

b. 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 has a name consists of verb and noun.

c.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, 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

A use case diagram consists of:

- a finite set of actors;
- a finite set of use cases;
- a communication consists of
  - association between actors and use cases;
  - generalization between two actors;
  - generalization between two use cases;
  - include relationship
  - extend relationship

a. Association

- Each actor must be associated/ involved with at least one use case.
- Every use case is involved with at least one actor.
- Actors are not allowed to interact (associate) with other actors.
- Association can only happen between actors and use cases.

b. Generalization

- Generalization can only happen between actors or between use cases.
- It is an acyclic relationship.
- Once the actor a_j is generalized to a_i, actor a_i cannot be generalized to a_j.
- Once the use case c_m is generalized to c_n, use case c_n cannot be generalized to c_m.

c. <<include>> relationship

- include relationship denoted by an arrow from a base use case to an included use case.

d. <<extend>> relationship

- extend relationship denoted by a new extension use case to a base use case.

5. Review on Well-formedness Rules Handling by Modeling Tools

This section is purposely to overview on how the two most popular modeling tools, Rational Rose and Visio handle the well-formedness rules.

Figure 2 shows the use case diagram drawn using Rational Rose. It depicts that modeler is freely draw the association between actors (`User` and `Customer`) on the use case diagram even it is known that association in use case diagram is only happen between actor (s) and use case (s). But when generalization is drawn between `User` and `Withdraw`, Rational Rose prompts an error message that noticing generalization must connect class to class or use case to use case.

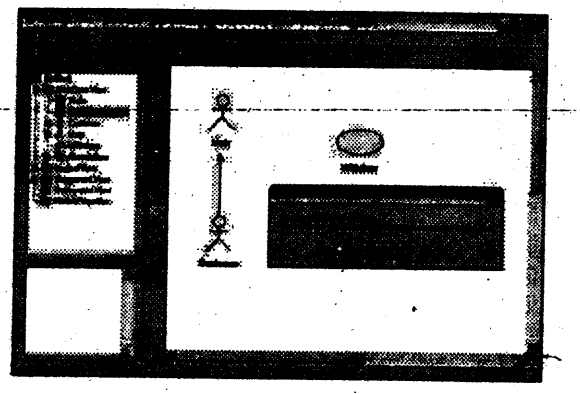

Figure 2. Use Case Diagram drawn using Rational Rose

Then, we draw use case diagram using Visio. The diagram is shown as Figure 3. It depicts that the well-formedness rules for association and generalization is not handled by the tool. It permits an association between actors (`User` and `Customer`) and generalization between actor and use case (`User` and `Perform ATM Transaction`).

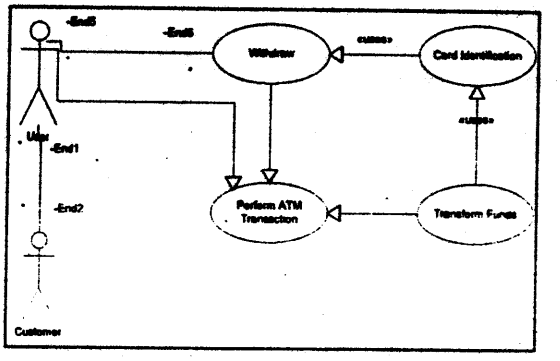

Figure 3. Use Case Diagram drawn using Visio

From above scenario, we can see that the well-formedness rules do not handle well by modeling tools. It left to modelers' expertise to decide on the rules. So, it is good to have a tool that enforce the well-formedness rules.

6. Formalization on 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 four elements A, C, S and G where

$A = \{a_1, a_2, a_3..., a_m\}$ is finite set of actors;
$C = \{c_1, c_2, c_3..., c_n\}$ is finite set of use cases;
$S = \{s_1, s_2, s_3..., s_m\}$ is finite set of associations;
G is generalization.

Following are a part of the rules:

a. 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.

$$\forall a \in A, \exists c \in C, S(a) = c \qquad (1)$$

Definition 3. Actors are not allowed to interact (associate) with other actors.

$$\forall a_i, a_j \in A, S(a_i) \neq a_j, 1 \leq i, j \leq m \qquad (2)$$

Definition 4. Association can only happen between actors and use cases, where

$$S \subset A \times C \qquad (3)$$

b. Generalization

Definition 5. Generalization between actors. The generalization is acyclic.

G is a relation on A, where aGb *iff* G is irreflexive and asymmetric, $\forall a, b \in A$.

$$G \subset A \times A \qquad (4)$$

If in $G, a_i \rightarrow a_j, i \neq j$, then

$$a_j \not\rightarrow a_i$$

Let $A = \{a_1, a_2, a_3\}$, then
$G = \{(a_1, a_2), (a_1, a_3), (a_1, a_3)\}$

Definition 6. Generalization between use cases. The generalization is acyclic.

G is a relation on C, where cGd *iff* G is irreflexive and asymmetric, $\forall c, d \in A$.

$$G \subset C \times C \qquad (5)$$

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
$G = \{(c_1, c_2), (c_1, c_3), (c_1, c_3)\}$

c.Include relationship

Definition 7. <<include>> relationship denoted by ----->. It is identified by

$$-----> \subset C \times C, \text{ as} \qquad (6)$$
$$c_m \text{ -----> } c_n$$

where c_m is base use case and c_n is included use case

d. Extend relationship

Definition 8. <<extend>> relationship denoted by <-----. It is identified by

$$<---- \subset C \times C, \text{ as} \qquad (7)$$
$$c_k <---- c_p$$

where c_k is base use case and c_p is new extension use case.

7. Conclusions

UML is a popular modeling technique especially in object-oriented based software development. Unfortunately, it still lacks of precise meaning of their diagram. Existing research on UML semantics focuses on dynamic semantics. It is important to formalize the UML well-formedness rules in order to help in improving consistence understanding of UML models between interested parties such as software modelers and developers. In this paper, we describe the UML well-formedness rules for use case diagram and formalize them. We intend to formalize 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

[1] Bahrami, A. (1999). *Object Oriented Systems Development*: Mc Graw Hill.

[2] Eichelberger, H. (2008). *Automatic Layout of UML Use Case Diagrams.* Paper presented at the Proceedings of the 4th ACM symposium on Software visualization Germany.

[3] Ha, I., & Kang, B. (2008). Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram. In *Intelligent Data Engineering and Automated Learning – IDEAL 2008* (Vol. Volume 5326/2008, pp. 436-443). Germany: Springer Berlin / Heidelberg.

[4] Labiche, Y. (2009). The UML Is More Than Boxes and Lines. In *Models in Software Engineering* (Vol. 5421/2009, pp. 375-386). Germany: Springer Berlin / Heidelberg.

[5] Li, X., Liu, Z., & He, J. (2001). *Formal and Use-Case Driven Requirement Analysis in UML.* Paper presented at the Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development.

[6] Liu, Z., He, J., & Li, X. (2001). *Formalizing the Use of UML in Requirement Analysis*: International Institute for Software Technology.

[7] Mostafa, A. M., Ismail, M. A., El-Bolok, H., & Saad, E. M. (2007). *Toward a Formalization of UML2.0 Metamodel using Z Specifications.* Paper presented at the Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on.

[8] OMG. (1997). *UML Semantics*: Object Management Group.

[9] Övergaard, G., & Palmkvist, K. (2004). A Formal Approach to Use Cases and Their Relationships. In *The Unified Modeling Language. «UML»'98: Beyond the Notation* (Vol. 1618, pp. 406-418). Heidelberg: Springer Verlag.

[10] Pons, C., Giandini, R., Baum, G., Garbi, J. L., & Mercado, P. (2003). Specification and Checking of Dependency Relations between UML Models In *UML and the unified process* (pp. 237-253). Hershey, USA: IGI Publishing.

[11] Sengupta, S., & Bhattacharya, S. (2008, 2008). *Formalization of UML Diagrams and Their Consistency Verification- A Z Notation Based Approach.* Paper presented at the Proceedings of the 1st conference on India software engineering conference, Hyderabad, India.

[12] Siau, K., & Lee, L. (2004). Are a use case and class diagrams complementary in requirements analysis? An experimental study on use case and class diagrams in UML. *Requirements Engineering, Volume 9,* (Number 4 / November, 2004), 229-237.