Journal of Software Engineering and Applications, **2013, 6, 73-77**
doi:10.4236/jsea.2013.63b016 Published Online March 2013 (http://www.scirp.org/journal/jsea)

Identification and Check of Inconsistencies between UML Diagrams

Xianhong Liu

School of Management, Henan University of Science and Technology, Luoyang, China.
Email: Lxh2072@163.com

Received 2013

ABSTRACT

Relationships between Unified Modeling Language (UML) diagrams are complex. The complexity leads to inconsistencies between UML diagrams easily. This paper focus on how to identify and check inconsistencies between UML diagrams. 13 consistency rules are given to identify inconsistencies between the most frequent 6 types of UML diagrams in the domain of information systems analysis and design. These diagrams are as follows: Use Case Diagrams, Class Diagrams, Activity Diagrams, State Machine Diagrams, Sequence Diagrams and Communication Diagrams. 4 methods are given to check inconsistencies between UML diagrams as follows: manual check, compulsory restriction, automatic maintenance, dynamic check. These rules and methods are helpful for developers to model information systems.

Keywords: UML; Inconsistency; Identify; Check

1. Introduction

Unified Modeling Language (UML) defines 13 types of diagrams which support developers to model information systems from different angles and levels. This kind of multi-view modeling way, on the one hand is useful to reduce complexity of models, on the other hand leads to inconsistencies between diagrams easily. Even though there are some researches on inconsistencies between UML diagrams, some of them are not perfect. Firstly, some researches don't discuses this issue completely. For example, Licong Tian argues that "Activity Diagrams is a kind of State Machine Diagrams. Consistency rules between Activity Diagrams and other Diagrams are the same as State Machine Diagrams. So I will don't discuses it in detail." [1]. In fact, there are some consistency rules between Activity Diagrams and other Diagrams. In addition, inconsistencies between Use Case Diagrams and other Diagrams are discussed rarely. Secondly, some conclusions are inaccurate. Xiaojian Liu argues that "if an operation of a class can call an operation of another class, there must be an association relationship between the two classes." [2]. This thought actually misunderstands the association relationship. This paper discuses how to identify and check inconsistencies between UML diagrams. But only 6 types of diagrams used frequently in the domain of information systems analysis and design are discussed as follows: Use Case Diagrams, Class Diagrams, Activity Diagrams, State Machine Diagrams, Sequence Diagrams, and Communication Diagrams.

2. Identification of Inconsistencies between UML Diagrams

How can we judge UML diagrams are consistent or not? Rules are needed. Some consistency rules that help developers to find out inconsistencies are illustrated bellow.

2.1. Consistency Rules between Class Diagrams and Sequence Diagrams

Class Diagrams describe classes, interfaces and relationships between them. A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Let's take the diagrams in **Figure 1** as an example to explain consistency rules between the two diagrams.

Rule 01: an object in Sequence Diagrams must be an instance of a normal class in Class Diagrams.

An object can not be created by itself, only be created by a class. In **Figure 1**, object "a" is an instance of class "A", object "b" is an instance of class "B", but the class that object "c' belongs to doesn't exist. It is obvious that the existence of object "c" breaks Rule 01. How to solve this problem? If object 'c" can not be deleted and it doesn't belong to neither class "A" nor class 'B", class

"C" must be added to the Class Diagram. Conversely, we need to adjust the class that object "c" belongs to or delete it in the Sequence Diagram. In addition, because we can't create an instance from an abstract class, the class an object belongs to must be a normal class.

Rule 02: when the name of a class is modified in Class Diagrams, the name of the corresponding class must be updated synchronously in Sequence Diagrams.

In Sequence Diagrams, an object should be given an appropriate name, and set an appropriate type which is a class from Class Diagrams. So rule 02 is obviously right. A class may be used in multiple Sequence Diagrams, therefore please note that the class should be updated in all Sequence Diagrams.

Rule 03: if an object sends a message to another object in Sequence Diagrams, there must be a dependency relationship between the two classes that the two objects belong to respectively. Contrariwise, if there is a dependency relationship between two classes, there must be at least one message interaction between the corresponding objects.

A dependency relationship means that a class depends on another one. In program code, a dependency relationship often represents that an object is used in an operation of another object. Specifically, an object is used as a parameter or a local variable of an operation of another object. Let's simulate the dependency relationship in **Figure 1** by a section of program code below. The operation "operation7" of class "B" is called by the operation "operation2" of class "A", which decides that object "a" must send a message named "operation7" to object "b" in the Sequence Diagram. So rule 03 is right.

```
public class A
{
    ……
    public void operation2( )
{
        B b=new B( );
        b.operation7( );
}
    ……
}
```

Licong Tian argues that "if there is a message interaction between two objects in Sequence Diagrams, there must be an association relationship between the corresponding two classes." [1]. Many scholars also agree with this viewpoint. According to rule 03, it is obvious wrong. The correct relationship should be dependency.

Rule 04: a message of Sequence Diagrams must correspond to an operation of the receiver (an object), and the operation is visible to the sender (an object).

A message in Sequence Diagrams is an order that an object sends to another object. The order must be an action that the receiver can complete. The action ultimately is represented as an operation of the receiver. As shown in **Figure 1**, the message "operation7" matches the operation "operation7" of class "B". But there is not an operation matching the message "operationx" in class "B". It is obvious that an inconsistency exists between the Class Diagram and the Sequence Diagram. In terms of the visibility, class "A" must had the authority to call "B", which depends on the visibility of class "B" and the package (namespace) that class "B" belongs to. In addition, object "b" must have the authority to call "operation7", which depends on the visibility of "operation7".

Rule 05: if a class is deleted in Class Diagrams, the corresponding objects and messages (corresponding to operations) of the class should be deleted synchronously in Sequence Diagrams.

Objects and messages in Sequence Diagrams derive from classes in Class Diagrams. Therefore, correlative objects and messages should be deleted when a class is deleted.

2.2. Consistency rules between Sequence Diagrams and Communication Diagrams

Communication Diagrams show interactions through an architectural view where the arc between the communicating Lifelines are decorated with description of the passed Messages and their sequencing. Sequence Diagrams and Communication Diagrams are kinds of variants of Interaction Diagram. Communication Diagrams correspond to simple Sequence Diagrams that use none of the structuring mechanisms such as Interaction Uses and Combined Fragments. They are on a par with each other, and can be transformed from one variant to anther one. So the consistency rules related to Sequence Diagrams mentioned above apply to Communication Diagrams. It is needless to discuses the consistency rules

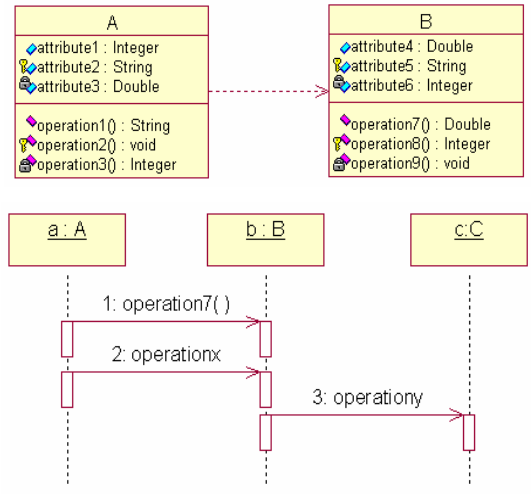

Figure 1. An example of Class Diagrams and Sequence Diagrams.

between Communication Diagrams and other diagrams. Let's take the diagrams in **Figure 2** as an example to explain consistency rules between Sequence Diagrams and Communication Diagrams.

Rule 06: the same object in Sequence Diagrams and Communication Diagrams must belong to the same class in Class Diagrams.

An object can only be created by a unique class. Therefore, when an object appears in more than one Interaction Diagrams, it should belong to the same class. As shown in **Figure 2**, object "c" is an instance of class "C" in the Sequence Diagram, but it is a instance of class "D" in the Communication Diagram. It is obvious that inconsistency exists between the two interaction diagrams.

Licong Tian argues one rule as follow: "If an object is created, deleted or modified in Sequence Diagrams, it should be created, deleted or modified in Communication Diagrams" [1]. This rule is meaningful when the two diagrams are used to describe the same use case or other object. But it is worthless to use them at one time. When a Sequence Diagram and a Communication Diagram describe different use case or other objects, objects and messages are different, so this rule argued by Licong Tian are worthless.

2.3. Consistency Rules between Class Diagrams and State Machine Diagrams

A State Machine diagram can show the different states of an object also how an object responds to various events by changing from one state to another. The elements of objects and states in State Machine Diagrams have relation to Class Diagrams. Therefore, there are some consistency rules between Class Diagrams and State Machine Diagrams.

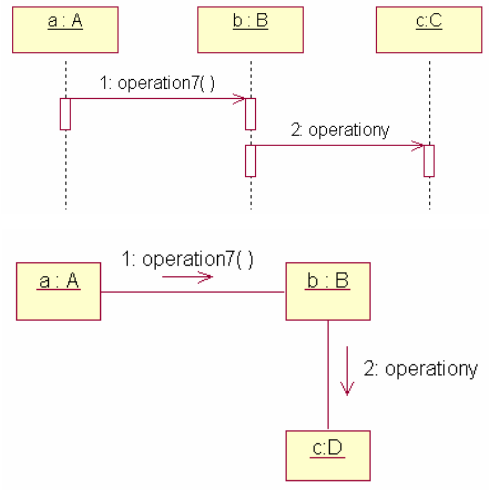

Figure 2. An example of Sequence Diagrams and Communication Diagrams.

Rule 07: an object that State Machine Diagrams describes must correspond to an instance of a normal class in Class Diagrams.

An object is created by a class. If an object that a State Machine Diagram describes doesn't belong to any class in a Class Diagram, the State Machine Diagram is worthless. In this situation, it is need to delete the State Machine Diagram, or add a new class to the Class Diagram.

Rule 08: if a class is deleted in Class Diagram completely (not hid only), all the corresponding State Machine Diagrams should be deleted.

An object in State Machine Diagram depends on a class in Class Diagrams. If a class is deleted, the states of its object lose its significance, and corresponding State Machine diagrams are not need any more.

Rule 09: a state in State Machine Diagrams must be a legitimate value of one or more attribute of a class in Class Diagrams.

A state is a situation or status that an object is doing something or waiting for something to happen [3]. States are essentially represented by the values of the attributes of an object. A change of states is just the change of attribute values. If a state doesn't accord with the range of attributes, it is obvious that State Machine Diagrams contradict Class Diagrams.

Rule 10: if an action in State Machine Diagrams is to call an operation of a class, the operation should keep consistent with the definition of the operation in Class Diagrams, including the name, parameters, etc.

An action represents what an object should do in the course of changing from one state to another. It is divisible or indivisible. It may be a simple calculation, or call an operation of an object. If an action calls an operation, the operation deserves to keep consistent with the definition of the operation in Class Diagrams.

Yun Wang argues that "an action of State Machine Diagrams should be defined in the class that a State Machine Diagram belongs to." [4]. This viewpoint is not accurate. It equates an action with an operation. Actually, operations are only one kind variety of actions. Weizhong Shao argues that "as for a transition of states of an object, a trigger is just to call an operation the object." [5]. This thought is also not accurate. A trigger is the reason to cause the transition. There is no inevitable relation between the two things.

2.4. Consistency Rules between Sequence Diagrams and State Machine Diagrams

When an object sends a message to another, the state of the receiver may be changed. Therefore, there are some consistency rules between Sequence Diagrams and State Machine Diagrams.

Rule 11: if an activity in State Machine Diagrams is to call an operation, the operation should be a message in Sequence Diagrams.

As mentioned above, a message in Sequence Diagrams corresponds to an operation of a class in Class Diagrams. If an activity in a State Machine Diagram is to call an operation, the operation must correspond to a message in Sequence Diagrams.

2.5. Consistency Rules between Use Case Diagrams and Class Diagrams

Use Case Diagrams describe Use cases, actors and relationships between them. It helps developers to capture and represent behaviors of information systems. Because behaviors of information systems finally are undertook by operations of classes. Therefore, there are some consistency rules between Use Case Diagrams and Class Diagrams.

Rule 12: use cases in Use Case Diagrams must be assigned to operations of classes in Class Diagrams.

Use cases represent functions or services of information systems. Operations of Classes are ultimate undertakers of these functions or services. So use cases in Use Case Diagrams must be assigned to operations of classes. Because the granularity of a use case is not of uniform size, it is possible that one use case doesn't just correspond to one operation.

2.6. Consistency Rules between Activity Diagrams and Class Diagrams

Activity Diagrams are often used to describe use cases in Use Case Diagrams or operations in Class Diagrams. The main elements of Activity Diagrams include activities, decisions, synchronizations, swimlanes and so on. Activity Diagrams are often used to describe use cases which must be assigned to operations of classes, so there are some consistency rules between Activity Diagrams and Class Diagrams.

Rule 13: if an Activity Diagram is used to describe a use case, activities and swimlanes in the Activity Diagram correspond to operations and classes in Class Diagrams respectively.

When an Activity Diagram is used to describe a use case, an activity is concrete behavior which corresponds to operations of classes. Because swimlanes represent activities are assigned to which person or organization, it should corresponds to classes. Because the granularity of an activity or swimlane is not of uniform size, it is possible that a use case doesn't just correspond to an operation, a swimlane doesn't just correspond to a class.

3. Check of Inconsistencies between UML Diagrams

Table 1. Check methods of inconsistencies between UML diagrams.

consistency rules	*manual check*	*compulsory restriction*	*automatic maintenance*	*dynamic check*
rule 01		√		
rule 02			√	
rule 03				
rule 04		√	√	
rule 05			√	
rule 06		√		√
rule 07	√			
rule 08			√	
rule 09	√			
rule 10		√		√
rule 11		√		√
rule 12	√			
rule 13	√			

How to check the consistency rules mentioned above are obeyed or violated? There are three approaches: checking by hand, checking by UML modeling tools, checking by formalization method [6]. It is difficult to transform UML diagrams into formalization language completely, so the feasible approaches are the former two at present. UML modeling tools can help us to check inconsistencies through three methods: compulsory restriction, automatic maintenance，dynamic check. Compulsory restriction means that UML modeling tools provide correct information for users to choose, avoid inputting wrong information. Automatic maintenance means when an element is modified, UML modeling tools automatically update corresponding elements. Dynamic check means that UML modeling tools capture users' operations that may cause inconsistencies and tell users what the reasons are and how they should do. Unfortunately, UML modeling tools can not find all the inconsistencies using the three methods. So the method of manual check is indispensable. Some inconsistencies can find only by hand. In **Table 1**, optimum methods are given to check each consistency rule. Please note that each consistency rule can be checked using multiple methods, but only one or two optimum methods are listed in the table.

REFERENCES

[1] Licong Tian,Besheng Zhou, "Research on model consis-

tency checking mechanism in UML visual modeling tools", Computer Applications and Software, vol. 22, Jan.2005, pp.24-26. (in Chinese)

[2] Xiaojian Liu, Zhanhuai Li, "Checking consistency of UML class diagram with relational model", Computer Engineering and Applications, vol.26, Sep.2006, pp.13-16. (in Chinese)

[3] Grady Booch,James Rumbaugh,Ivar Jacobson, The Unified Modeling Language User Guide. New Jersey: Addison-Wesley Professional,2005

[4] Yun Wang, Youcheng, Liu, "Model consistency checking mechanism in UML visual system", Journal of Computer Research & Development, Vol.37, Jan.2000, pp.1-8. (in Chinese)

[5] Weizhong Shao, Fuqing Yang, Object-Oriented System Analysis and Design. beijing: Tsinghua University Press,2006

[6] Xi Chen, QingChun Wang. "Research on UML consistency", SoftWare Guide, vol.8, Apr.2009, pp.26-27. (in Chinese).

