A methods of ensuring consistency between UML Diagrams

طرق ضمان الاتساق بين مخططات النمذجة (UML)

Ihab L.Hussein Alsammak , Assistant lecturer
Ministry of Education, Directorate General of Education of Karbala
ehablaith@Gmail.com
Humam M. Abdul Sahib, Assistant lecturer
Department of Electrical and Electronics Engineering
College of Engineering, University of Kerbala, Iraq
humam.alkaabi@uokerbala.edu.iq
Wasan H.Itwee
Ministry of Construction Housing and Public Municipalities
General Directorate of Sewerage/ Sewerage Karbala Directorate
Wassan_iraq@yahoo.com

Abstract

The Unified Modelling Language (UML), is the first step of developing an object-oriented design method and a standard notation for the modelling of real-world objects and it consists of fourteen various diagram types. The complex relationships between UML diagrams will lead to the inconsistencies among the unified modelling language (UML) diagrams therefore, it is so important to verify model in the early phase before implement it because early error detection will be easier to fix than later phase.

This paper focuses on in what the way to determine and identify inconsistencies between unified modelling language diagrams. Eleven of the consistency rules will be applied to check the consistency models between mostly used categories of unified modelling language diagrams in the field of systems design and systems analysis. As follows state of these diagrams: (Use Case Diagram, Class Diagram, Communication Diagram, and Sequence Diagram). To check inconsistencies between unified modelling language diagrams there are four ways: dynamic check, automatic maintenance, manual check, and compulsory restriction. Using these methods and rules are useful for developers to model material systems. In the consistency rules each diagram has elements are described by a logical approach.

Keywords: UML; Consistency rules; Use Case Diagrams, Class Diagrams; Sequence Diagrams; Logical approach.

المستخلص :

لغة النمذجة الموحدة (UML)، هي الخطوة الأولى للتطوير و التصميم الموجه نحو الكائن والتدوين القياسي لنمذجة الكائنات في العالم الحقيقي، ويتألف من أربعة عشر نوع مختلف من الرسم التخطيطي. العلاقات المعقدة بين مخططات النمذجة (UML) ستؤدي إلى التناقضات بين المخططات لذلك فمن المهم جدا التحقق من النماذج في المرحلة المبكرة من تصميم النظام قبل تنفيذه لأن الكشف عن الأخطاء في وقت مبكر سيكون من الأسهل إصلاحه عن ما إذا تم كشفه في المرحلة اللاحقة.

تركز هذه الورقة على كيفية اكتشاف وتحديد التناقضات بين مخططات النمذجة (UML) . سيتم تطبيق أحد عشر (11) من قواعد الاتساق للتحقق من اتساق النماذج الأكثر شيوعا بين المخططات النمذجة (UML) في مجال تحليل النظم وتصميمها . هذه المخططات هي كما يلي: (Use Case Diagram ، Class Diagram ، Communication Diagram ، Sequence Diagram). هناك أربع طرق للتحقق من التناقضات بين المخططات النمذجة (UML)على النحو التالي: الاختيار الديناميكي، والصيانة التلقائية، والتقييد الإجباري، والتحقق اليدوي. هذه القواعد والأساليب مفيدة للمطورين لنمذجة نظم المعلومات. يتم وصف عناصر كل مخطط مشارك في قواعد الاتساق باستخدام نهج منطقي.

الكلمات المفتاحية : UML ، مخططات الطبقات، قواعد الاتساق، مخططات التسلسل، استخدام مخططات الحالة، النهج المنطقي

1. Introduction

Software processes whether if software specification, software designs and implementation, software validation and software evolution are used four activities as software development's criteria [1]. These activities are sequential, incremental or repetitive and functions are restricted to programs it must be identified. One of the most important techniques used to document the specifications of systems and software model [1]. The software model is a method of an abstract representation, graphical functions and software limitations and used to understand the program before building or modifying it.

There are many researchers involved in the given official definition of UML diagrams [2]. Unified Modelling Language (UML) is used to capture and document all program requirements in the object-based system. Until now, UML is represented by fourteen graphs that are used to explain various views of the system. Using Object Constraint Language (OCL) [3] which is supported by natural language description the abstract syntax and semantics that is one of UML standard. However, the reduction of OCL for the expression of UML elements [4] and as not all the elements in the OCL [5] forces the research to give a more accurate definition of UML elements. The exact meaning of the elements is very important in order to arrive at a common understanding of their meanings and make UML can be a reason in terms of the consistency of the graph.

Consistency is a situation in which two or more elements of different programs that describe aspects of the system are satisfied, the meaning of a dependency relationship is that class depends on another class. In the program code, the dependency relationship is often represented by an object being used in another object process [6]. One of the most important benefits of consistency between the models is to make sure the implementation of the system properly and without problems [7]. Usually, there are three major activities in managing model consistency and these are inconsistency detection, consistency specification and inconsistency handling [8]. In the consistency specification, consistency rules that should be respected by various schemes are described in order to be consistent. If the consistency rules are adhered to, inconsistencies will reduce and can be detected and dealt with. Although there are a lot of researches on consistency between UML diagrams, there is still a lack of research of consistency driven by the use case.

The remainder of the paper is structured as follows. In **section 2** presents some of previous work and methods to illustrate of finding the inconsistency together with discussions relating it to other similar work and methods. In **section 3** the scope of work is shown. **In section 4** how discovery of inconsistencies between UML Diagrams are shown with some rules and examples. The outline the conclusions and future work in **section 5.**

2. RELATED WORK

This part provides a review of some of the related work that has been done from some of researcher in the field of consistency of UML designs for software systems. Boris Litvak and Yehudai used an algorithmic method to check consistency between UML diagrams Sequence and State to check the contradictions the UML modeling tools was used to help us to detect inconsistencies is shown in [9] while Küster, J. M. and Stehr, J. [10] they proposed an advertising approach using the CSP algebra process to verify consistency between sequence and state graphs. Egyed, A. [11] has approach to examining the automated consistency named VIEWINTEGRA was developed.

Chanda et al. [12] ,Shinkawa [13] and Sapna et al. [14] are proposed consistency between use case and activity diagram. Using Colored Petri Net (CPN) ,Shinkawa [13] describe the consistency between sequence ,use case, state chart and activity diagram. He is suggested that the use case should have at least one activity diagram.

He determines the use case, action and implementation events as transitions. while Sapna and other. [14] recognize employment case elements, sequencing and action graphs using the schema table. But Noted the definitions are limited only to elements of the use case, activity, actor, and aim of the writing. It is possible to be A use case it may be an activity diagram, and every actor in use case diagram should is agree to a class in activity diagram. here found They define that all object

and its messages in sequence diagram agree to a class and its style in class diagram. That should notes They suggest two (2) consistency rules between sequence diagram and use case.

OCL is applied to fast the consistency rules. in use case maybe there are different flows of scenarios or activities. and after that the scenario is described through a sequence diagram. Chanda et al. [12] broad elements of use case, sequence diagram and activity such as CFG. They locate an action/ activity in activity diagram such as a case of a use case in use case diagram. after that notes The state syntax of all diagram is then used to cause the rules by using CFG.

All most study on consistency between class diagram and use case are most important to researchers [14; 15]. Fryz et al. [15] see a use case diagram as Requirements for users and they described the diagram as a graphic. by using graphs Was selected defined consistency between class diagram and use case. Depending on the Elements assign in consistency rules by means of Chanda et al. [12] and Sapna et al. [14] and not put up with abstract syntax standardize through Object Management Group (OMG) [16]. It is advantageous from use the norm with regard Application of any proposed approach to industrial software development.

3. SCOPE OF WORK

There are many researchers involved in the given official definition of UML diagrams because they focused on several areas and use different technology. In this part of this paper, the explanation of the formalization the elements of Use Case, Sequence, Class and Communication diagrams are shown. Further, the consistency rules are shown.

Definition 1. UML model is defined as a set

UML model = {UCD = Use Case diagrams, SEQD= Sequence diagrams, CDs= communication diagrams, CD= Class diagrams},

Where:

UCD = {ucdi|$1 \leq i \leq n$} is limited set of Use Case diagrams.

SEQD = {SeqDi|$1 \leq i \leq n$} is limited set of Sequence diagrams for Use Case.

CDs= {commDi|$1 \leq i \leq n$} is limited set of communication diagrams for Sequence.

CD = {CDi|$1 \leq i \leq n$} is limited set of Class diagrams for Sequence.

Definition 1 describes a UML model that consists of at least one Use Case diagram, one Sequence diagram, one Communication diagrams, one Activity diagrams and one Class diagrams.

3.1 Formalization of UCD

Definition 2. Use Case Diagram (UCD) is defined as a set

UCD = {A, UC, R, CO},

Where

A is referring to limited set of actors when A = {ai|$1 \leq i \leq n$},

UC is referring to limited set of Use Cases where UC = {uCi|$1 \leq i \leq n$},

R is referring to limited set of relationships where R = {Assoc, Include, Extend, GenUC, GenAc},

CO is referring to limited set of constraints for UCD where CO = {COi|$1 \leq i \leq n$},

COi is referring to limited set of constraints for a Use Case uCi where COi = {COij|$1 \leq i \leq n$ and $1 \leq j \leq n$}.

3.2 Formalization of SEQD

Definition 3: SeqDuCi is defined as a finite set of Sequence diagrams corresponding to a Use Case uCi.

SeqDuCi = {SeqDuCi1, SeqDuCi2, ..., SeqDuCin| uC ∈ UC}

where SeqDuCi ∈ SEQD

Definition 4: SeqDuCin is defined as a tuple

SeqDuCin = {Ps, E, V, l, O, C, S}

where

- Ps is a set of lifelines denoting involved in relations where Ps = {psi| $1 \leq i \leq n$}.
- E is a set of events, each event match to receiving or sending a message where E = {ei |$1 \leq i \leq n$}.

- V is a limited set of edges. V is referring to a link between two Ps. Therefore, V can be representing as
  {(e,e') | e,e' ∈ E and e'≠ e}. V = {vi| 1≤ i ≤ n}.
- l is a refer to labeling function which assign each v ∈ V a message name m with m = l (v).
- O is a function which maps each e ∈ E to the member it belongs to.
- C is a set of Boolean of form t(e) – t(e') ≤ d which represents the timing constraints enforced on SeqDuCin.
- S is a limited set of states to which participant goes where S = {si |1≤ i ≤ n}.

3.3 Formalization of CDs

Definition 5: (CDs definition)

let assumed that the types of personal elements of communications diagram as:

Elements = {Lifelines, Links, Messages, Conds, Msgs,Vals, Prms }

A communication diagram has an 8-tuple:

CDs = (CDname, Elements, α_{cond} , α_{msg}, α_{val}, α_{prm} , α_{in} ,α_{out})

Where:

- **CDname** refers to the communication diagram name.
- **Lifelines** is referring to finite the set of lifelines
- **Links** is referring to the finite set of links
- **Messages** is referring to finite the set of messages
- **Conds** is referring to the finite set of conditions.
- **Msgs** is referring to the finite set of messages names.
- **Vals** is referring to the finite set of return values.
- **Prms** is referring to the finite set of parameters.

3.4 Formalization of Class diagram

A class is a one of the most major block of UML class diagram that is used to store and manage information. The main structure of class consists of " name at the top, attributes in the middle, and operation at the bottom " by these three components with a class used to each class is graphically depicted (see figure 1). A UML class C is representing by atomic theory C. The above explanation describe that a class is usually a set of operations and attributes denote shorten as:

$C = (C_n, a_i, a_j\ O_i)$ where:

- C_n refer to the class name
- a_i is a finite set of attribute name of the class, for i= l . .n .
- a_1· is a finite set of attribute type of the class, for j= l . . . n.
- O_i is a finite set of operations of a class, for i= 1 . . n

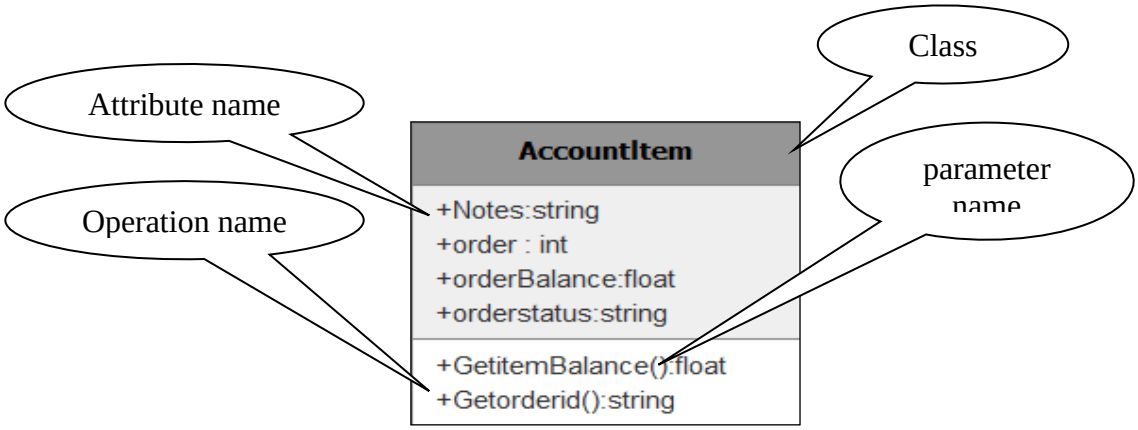

Figure 1. Common Properties of a Class in UML Class Diagram

4. discovery of Inconsistencies between UML Diagrams

To ensure a system execution is correctly and the system is consistent or not, rules must be used as explained below.

4.1. Consistency Rules between Class Diagrams and Sequence Diagrams

Class diagram includes an interfaces and relationship between other classes. A sequence diagram is an interactive diagram that shows how objects works together and in which order and it is object interactions appear arranged in the time sequence. It depicts the objects and classes involved in the script and the sequence of messages exchanged between the requirements to perform scenario functions. For example, the diagram as shown in figure 1 to explain the rules of consistency between the two figures.

1) **Rule 01:** The number of objects in the sequence diagram must be equal to their number in the class of the class diagram.

   Usually the object created by the class, it is impossible created by itself. In Figure 1, object "x" is a corresponds of class "X", object "y" is a corresponds of class "Y", but in the example it is very clear that object "z" unmatched in the class and the class that belong to doesn't exist. This indicates that the existence of object " z " is a violation of Rule 1. If the object "z" is very important, it can not be deleted, nor does it belong to the class "X" and the class "Y" so it must be adding new named class "Z" to the Class Diagram. Conversely, if the addition class of object "c" is not possible, It must be removed in the Sequence Diagram.
2) **Rule 02:** The name of the sequence Diagrams must be updated synchronously when the class name is updated in corresponding diagrams in the sequence Diagrams.

   In all Sequence Diagrams and Class diagrams, must give an appropriate object name as well as an appropriate class type. That means the rule 02 is obviously right. The possibility of using the class in multiple Sequence Diagrams, so update the class in Sequence Diagrams must be wholly.
3) **Rule 03:** There must be a dependency relationship between two or more classes to which the objects belong respectively in Sequence Diagrams if the object sends a message to another object in the sequence Diagrams.

   In contrast, there should be at least one message interaction between the corresponding objects if there is a dependency relationship between two or more class.

The meaning of a dependency relationship is that class depends on another class. In the program code, the dependency relationship is often represented by an object being used in another object process. Specifically, an object is used as a parameter or local variable for another object process. Let's analyze the dependency relationship in Figure 1 and as shown in the program code part below. Operation "operation7" of class "Y" is called operation "2" of type "X", which determines that the object "x" must send a message called "operation7" to intercept "y" in the sequence diagram. So that means the Rule 03 is correct.

```
Public class X
{
…….
……
Public void operation2()
{
Y y=new Y();
y.operation7();
}
……
}
```

Licong Tian and Besheng Zhou said 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 researchers agree with this view. According to rule 03, it is obviously wrong. Dependency is the best choice for the correct relationship.

4) Rule 04: Compatibility must be achieved between the message of sequence Diagrams and operation of the receiver (an object), And the process must be visible to the sender (an object).
In the sequence Diagrams, the message means the object sends a message to another object. One of the features of the order where the receiver should be able to complete it. The procedure is ultimately represented as a process for the future. As shown in **Figure 1**, the message named "operation7" in sequence diagram competitions the process named "operation7" of class "B" in class diagram, but any operation corresponds the message named "operation A" in class "B" can't be seen. There is clearly a no consistency between the sequence diagram and the class diagram. When the talking about vision, the class "X" must have the authority to call the class "Y", Which depend on the vision of class "Y" and to which the class "Y". The same is true for the object "x" and the operation "operation7".
5) Rule 05: If any deletion or addition of a class is done in the Class Diagram, the changes must be done synchronously on the corresponding objects and message in the Sequence Diagram.
Through the class in the Class Diagram are derived messages and objects into the Sequence Diagram.

4.2. Consistency rules between Communication Diagrams and Sequence Diagrams

Communication Diagrams it is shown the relations during an architectural view where the arc between the communicate Lifelines are graced with description of the pass into Messages and their sequencing. Communication Diagrams and Sequence Diagrams are ones of the types of variables of the interaction diagram. Communication Diagrams and Sequence Diagrams are matched especially with simple that uses none of the structure mechanisms like Combined Fragments and Interaction Uses. Communication Diagrams and Sequence Diagrams are on par with each other as well as for variables can be transformed from one variable to another. This means that the consistency rule related that previously described in Sequence Diagrams can be also applied to Communication Diagrams. So it is not very necessary to discuss all the consistency rules between Communication Diagrams and other diagrams. Let's take an example in Figure 1 to explain and show some consistency rule between Communication Diagrams and Sequence Diagrams**.**

6) Rule 06: The objects of both the Communication Diagrams and the Sequence Diagrams of the system must belong to same class in the Class Diagrams.
The object can only be created through a special class. This means that when the object appears in one or more interactions, it must be done in the same class. It is clear in the example in Figure 1, the object "z" is a corresponding in class "Z" in the Sequence Diagram, but it is a corresponding in class "D" in the Communication Diagram. That is means detection inconsistency between the 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 significative when the two diagrams are used to describe the same use case diagram or other objects. But do not benefit us to use at one time. Objects and messages become different when a Communication Diagram and a Sequence Diagram describe different use case or other objects, so this rule do not match by Licong Tian are useless.

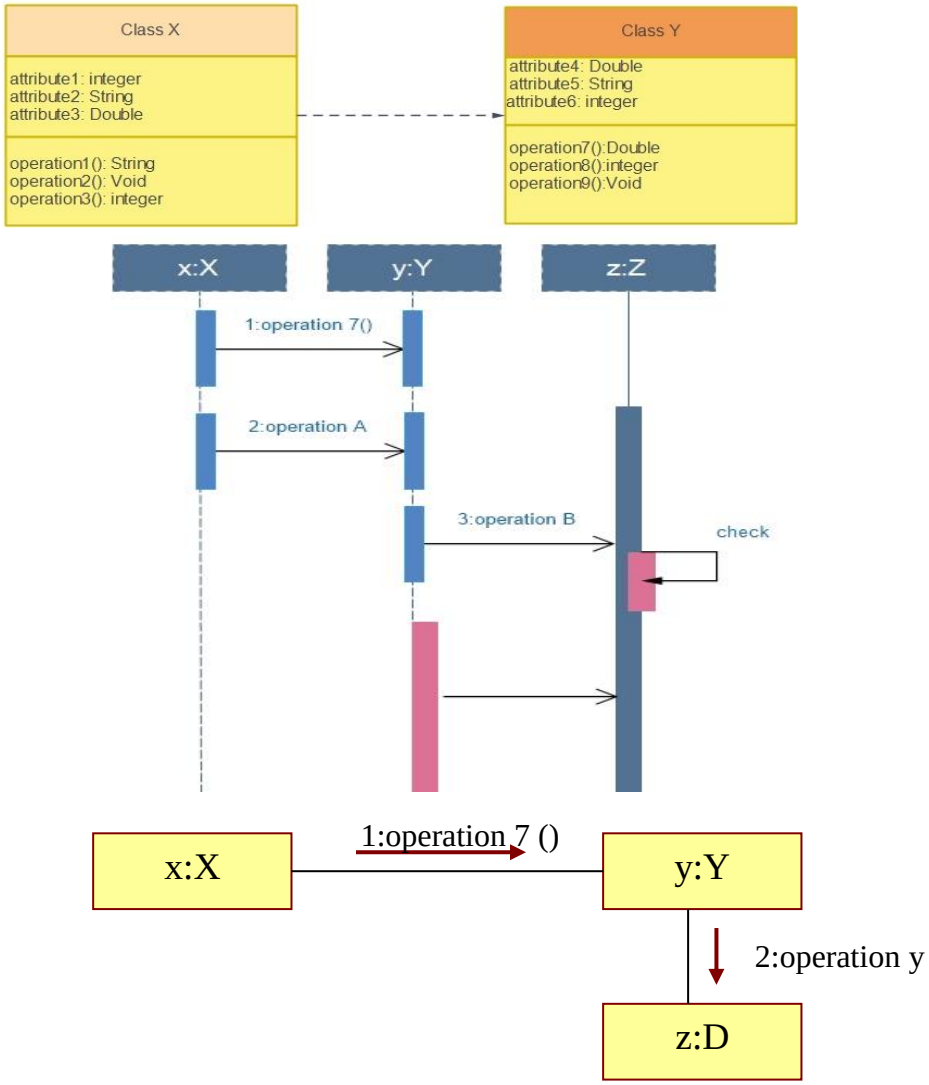

Figure 2. An example of Class Diagrams, Sequence Diagrams and Communication Diagrams

4.3 Consistency Rules between Class Diagrams and Use Case Diagrams

Use Case Diagrams explain actors, Use cases and relationships between them. The developers use the Use Case Diagrams to helps him to represent and capture the behaviors of information systems. Since these behaviors of information systems will be implemented later in Class Diagrams. For this reason, it is very important that there be consistency rules between Diagrams such as Use Case Diagrams and Class Diagrams.

7) Rule 7: Each use cases in Use Case Diagrams must be correspond the operations of classes in Class Diagrams.

Because the functions and services are explained by Use cases. The Operations in the class are the final representation of the functions and services available in Use cases. Therefore, each Operations in the class Diagrams must have the corresponding use cases in Use Case Diagrams. Because the using of cases is not fixed, this use cases the possibility of mismatch one operation.

8) Rule 8: Each use case must have a noun and a verb.

The noun of use case in Use Case Diagrams should correspond to the name of one class in the Class Diagram. That means, the use case in the class diagram and class in the class diagram must belonging between them, so that Use case.name equal class. name. Also, the verb must match to an operation of a class in the class diagram.

4.4 Consistency Rules between Use Case and Sequence Diagrams

The ATM system will examine as example to show how the rules can be applied.

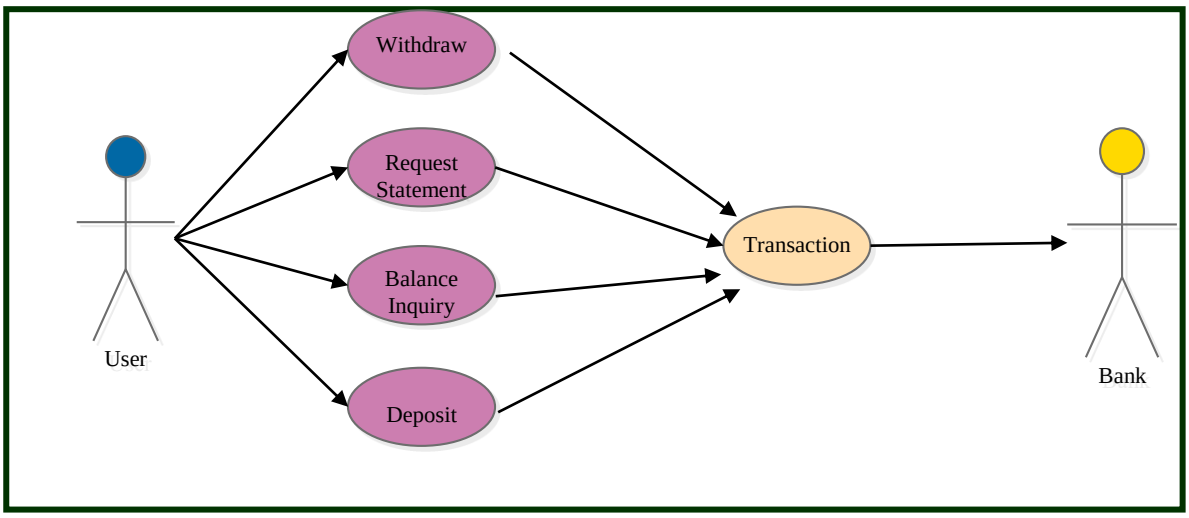

Figure 3: Use Case Diagram for ATM System.

9) Rule 9: : in use case diagram for every Use Case, Sequence diagram will have occurred at minimum one.

In figure 3 with Withdrawal Use Case and in figure 4 there is a one Sequence diagram is appearing. So the figure 3 and figure 4 validated the Rule 1

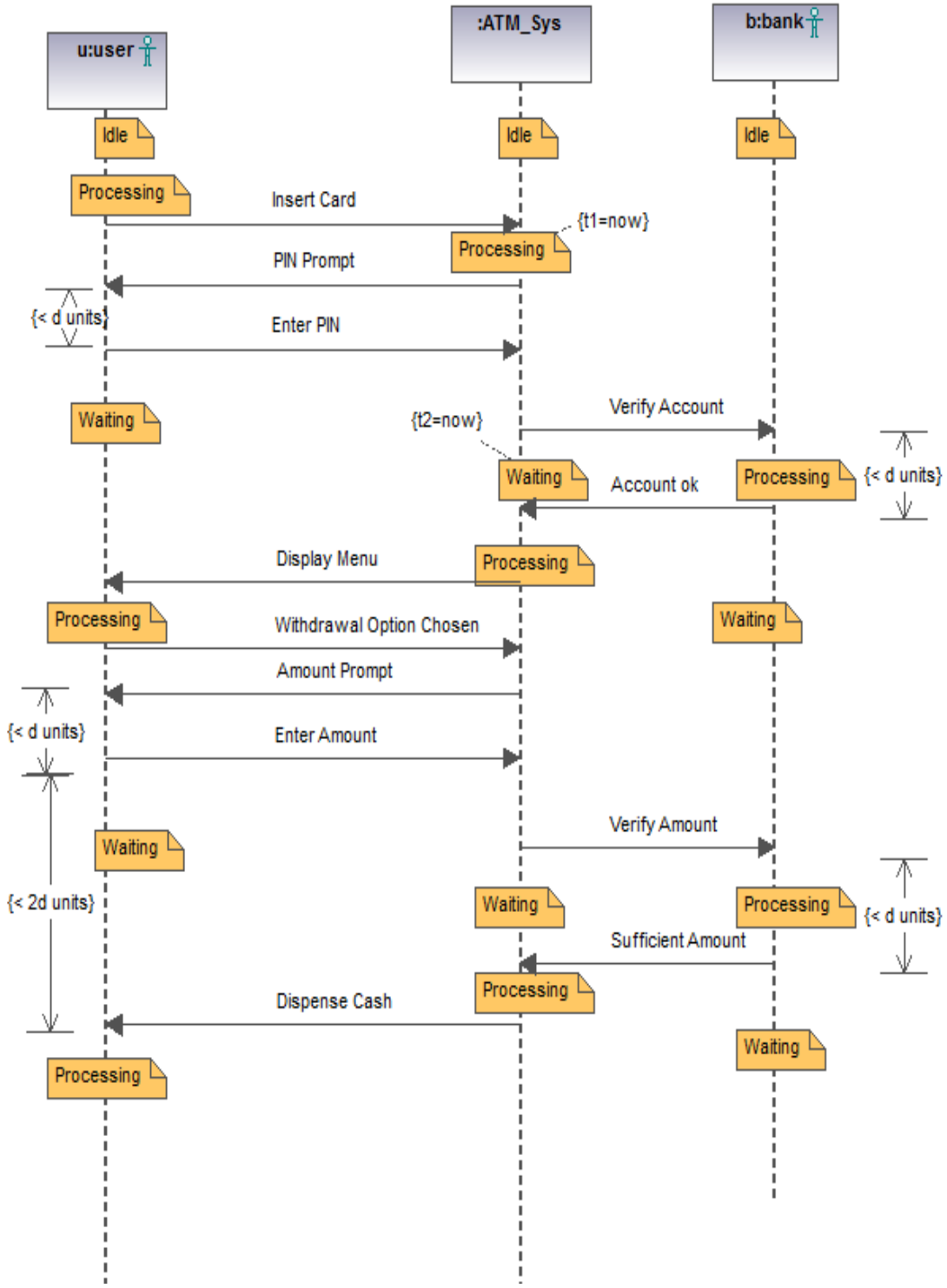

Figure 4: Sequence Diagram for Withdrawal Use Case

10) Rule 10: in the corresponding Sequence diagram with a Use Case, each actor associated will appear as a participant.

In figure 3 the use case of the bank and user of use case diagram, according to Rule 10 seem as participants in Sequence diagram in Figure 4

11) Rule 11: Let the Use Case Diagram are a set of {Actor, use case, Relationship, constraints} and sequence diagram. = {lifelines, Events, edges, labelling, maps, constraints, states} be a sequence diagram matching to set Use Case Belongs Use Case. For any set of Use Case

Belongs Use Case, the set of constraints Belongs constraints, associated with set of Use Case is the subset of the set of constraints in sequence diagram of set Use Case. constraints C.
In figure 3 the Withdrawal Use Case is connected with some bands which are reflected in Figure 4 .

Table 1. Check methods of inconsistencies between UML diagrams.

consistency rules	manual check	compulsory restriction	automatic maintenance	dynamic check
Rule 1		√		
Rule 2			√	
Rule 3		√		
Rule 4		√	√	
Rule 5			√	
Rule 6		√		√
Rule 7	√			
Rule 8		√		
Rule 9	√		√	
Rule 10		√		
Rule 11	√			

When any element is modified that means Automatic maintenance, the UML modeling tools update corresponding elements as automatically can define the Dynamic check as UML modeling tools capture user procedure by this way the users know the contradictions and how they should be done. By using the UML modeling tools and using the three methods unfortunately not possible find all the inconsistencies, this is proof of the manual check as method is indispensable, by using the hand some of inconsistencies can find it easily. for each consistency rule as in Table 1 optimum way are given to check it , but only one or two optimum methods are listed in the table 1. It must be taken into consideration by using diverse methods can be checking the consistency rule in same time should notes in the table at maximum two best approaches are listed.

5. CONCLUSION

It is very important to find the error from making sure user model in the early phase, when those errors are detected early in the design process it is more helpful and easier to fix than if they were detected at a later phase. In other words, the inconsistencies between UML diagrams will be the cause of increasing defects, errors, and problems in the software systems, so the diagrams describing a software must be consistent.

In this paper, three methods were examined: first verification was done by hand and the second verification was done using modeling tools (UML) and the third verification using the representation method. As is known, it is difficult to convert UML schemes or diagrams into official language, so at present, the two previous approaches can be considered as feasible approaches, to check the contradictions the UML modeling tools was used to help us to detect inconsistencies, this is achieved by relying on three methods: method 1 compulsory restriction, method 2 automatic maintenance and method 3 dynamic check. from the compulsory restriction the UML modeling tools provided to the user a correct information to choose, parry inputting incorrect information.

References

[1] Sommerville, I. (2004). Software Engineering 7 (Seventh ed) (Second ed.). Harlow: Pearson Addison Wesley.

[2] Hubaux, A. Cleve, P.-Y. Schobbens, A. Keller, O. Muliawan, S. Castro, et al. (2009). Towards a Unifying Conceptual Framework for Inconsistency Management A F. J. Lucas, F. Molina, A. Toval.: A Systematic Review of UML Model Consistency Management, Information and Software Technology, 51, 2009, 1631-1645.

[3] Object Management Group (OMG) (2010). OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.3. Needham, MA 02494, U.S.A. : Object Management Group (OMG).

[4] Lucas, F. J., Molina, F., & Toval, A. (2009). A Systematic Review of UML Model Consistency Management. Information and Software Technology, 51, 1631-1645.

[5] Ibrahim, N., Ibrahim, R., Saringat, M., Mansor, D., & Herawan, T. (2010). On Well-Formedness Rules for UML Use Case Diagram. In F. Wang, Z. Gong, X. Luo & J. Lei (Eds.), Web Information Systems and Mining (Vol. 6318, pp. 432-439): Springer Berlin / Heidelberg.

[6] F. J. Lucas, F. Molina, A. Toval. A Systematic Review of UML Model Consistency Management, Information and Software Technology, 51, 2009, 1631-1645.

[7] Nugroho, A., & Chaudron, M. R. V. (2008). A survey into the rigor of UML use and its perceived impact on quality and productivity. Paper presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, Kaiserslautern, Germany.

[8] Hubaux, A., Cleve, A., Schobbens, P.-Y., Keller, A., Muliawan, O., Castro, S., et al. (2009). Towards a Unifying Conceptual Framework for Inconsistency Management Approaches: University of Namur Rue Grandgagnage.

[9] B. Dobing, J. Parsons. Dimensions of UML Diagram Use: A Survey of practitioners, Journal of Database Management, 19, (1), 2008, 18.

[10] [Küster, J. M. ,Stehr, J. Towards explicit behavioral consistency concepts in the uml. Second International Workshop on Scenarios and State Machines: Models, Algorithms and Tools, Portland,Oregon, USA, May 3 2003.

[11] Egyed, A. Scalable consistency checking between diagrams-the view integra approach. 16th IEEE International Conference on Automated Software Engineering (ASE'01), San Diego, California, November 26-29 2001.

[12] J. Chanda, A. Kanjilal, S. Sengupta, S. Bhattacharya.Traceability of Requirements and Consistency Verification of UML UseCase, Activity and Class diagram: A Formal Approach, in: International Conference on Methods and Models in Computer Science 2009 (ICM2CS), 2009, pp. 1-4.

[13] Y. Shinkawa.: Inter-Model Consistency in UML Based on CPN Formalism, in: 13th Asia Pacific Software Engineering Conference (APSEC '06) 2006, pp. 414-418.

[14] P. G. Sapna, H. Mohanty. Ensuring Consistency in Relational Repository of UML Models, in: 10th International Conference on Information Technology (ICIT 2007), 2007, pp. 217-222.

[15] L. Fryz, & L. Kotulski. Assurance of System Consistency during Independent Creation of UML Diagrams, in: 2nd International Conference on Dependability of Computer Systems, 2007 (DepCoS-RELCOMEX '07), 2007, pp. 51-58.

[16] Object Management Group (OMG), OMG Unified Modeling LanguageTM (OMG UML), Superstructure Version 2.3. Retrieved from: <http://www.omg.org/spec/UML/2.3>, 2010.