A Precise Approach for the Analysis of the UML Models Consistency*

Francisco Javier Lucas Martínez and Ambrosio Toval Álvarez

Software Engineering Research Group,
Department of Informatics and Systems,
University of Murcia (Spain)
{fjlucas, atoval}@um.es

Abstract. The UML notation is a well-know standard notation to describe OO systems. But the UML specification has certain imprecisions and ambiguities that, along with possible errors made by the modellers, may cause inconsistency problems in the models of the system. This paper presents a rigorous approach to improve the consistency analysis between UML diagrams.

This proposal is based on a previous formalization of the UML metamodel diagrams, [1–4], in *Maude*. The framework given by the specifications created helps to guarantee the consistency of models because all the specifications are integrated within the same formalism. This work focuses on the analysis of the inter-diagram consistency. Several examples of properties are shown that help to guarantee the consistency between UML Communication and Class Diagrams.

1 Introduction

UML [5] is a modelling language which was created as union of varied notations, and promoted by OMG. But UML specification has certain imprecisions and ambiguities that, along with possible errors made by the modellers, cause inconsistency problems in the models of the system. Within the UML-based development process, the main sources of inconsistency are, [6]:

1. The existence of multiple software artifacts or diagrams to describe the same system, which can cause inconsistencies in the information that appears in these diagrams.
2. The imprecise semantics of the UML, which means that a UML model may have multiple interpretations.

This paper presents a rigorous approach to analyze and improve the consistency between UML diagrams. This proposal is based on a previous formalization of the UML metamodel diagrams, [1–4], in *Maude* [7]. The framework given by the specifications created helps to guarantee the consistency of models, because

* Financed by the Spanish Ministry of Science and Technology, project DYNAMICA/PRESSURE TIC 2003-07804-C05-05.

J. Akoka et al. (Eds.): ER Workshops 2005, LNCS 3770, pp. 74–84, 2005.

all the specifications are integrated within the same formal technique (algebraic specifications). Furthermore, the semantic of each one of these specifications has a precise and no ambiguous interpretation due to its formalization in a formal language.

The language chosen for the realization of the formalization is *Maude*. This is a formal specification language that is based on equational logic and rewriting logic. Furthermore, Maude is a language that allows the execution of the specifications created, which allows one to animate models and create system prototypes to check the behavior of the system.

This work is based on the formalization carried out in previous work, in which the formalization of the following UML metamodel diagrams are treated: Class Diagram [1], Collaboration Diagram [2] (named Communication Diagram in UML 2.0), Statechart Diagram [3], and Sequence Diagram [4]. All theses formalizations have been updated to UML 2.0.

Thus, the integration of this formalization and the work produced that have been performed about them, such as: animating models, making transformation between models and verifying of properties can be used to improve the quality of a system.

Furthermore, all the applications of this formalization can be used in MDA, since UML language is usually used as the modelling language in MDA. We can use it to guarantee the consistency of the PIM (Platform Independent Model) models, before transforming them to PSM (Platform Specific Model) models. This formalization can also give support to the transformations that are made within the MDA (PIM→PSM, PSM→PSM, PIM→PIM).

This work focuses on the analysis of the inter-diagram consistency, and several properties are shown that help to guarantee the consistency between UML Class and Communication diagrams. This algebraic approach can be applied to any diagram which is formalized, see section 2.

After this introduction, in section 2, a general description of the algebraic formalizations of the UML diagrams used in this work is given. Section 3 shows the analysis of the consistency made for the UML Class and Communication Diagram. Section 4 identifies some related work. Finally, in section 5 conclusions and further work are given.

2 Algebraic Formalization of the UML Diagrams

As a previous step to the rigorous analysis of the consistency between the different UML diagrams of a system, it is necessary to have a rigorous representation of these models. We decided to make an algebraic formalization of part of the UML metamodel. Figure 1 presents the necessary algebraic modules to carry out an analysis of the consistency. The *Integration* module uses the available specifications of the UML diagrams and implements the equations that check the inter-diagrams consistency.

This paper focuses on the Class and Communication Diagrams. This method is generalizable to any combination of two or more diagrams, because we have

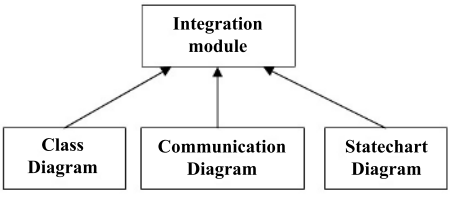

Fig. 1. Algebraic modules used in the analysis of the consistency

the corresponding formalization and integration with the rest of the diagrams, for example the UML Statechart Diagram [3].

The next sections give a description of the formalization of each diagram needed to understand the rest of the paper. For the sake of brevity, the description of the specification offers a very simplified view of the algebraic formalization. For more details, see [1, 2].

2.1 UML Class Diagram

The first diagram that will be commented is the UML Class Diagram. This diagram describes the static structure of a system and is made up of a set of elements such as classes, interfaces, and others; and relationships among these elements, such as associations and aggregations. The module that contains the formalization of the diagram is shown in Figure 2. For the sake of brevity, this is a very reduced part of the formalization (see [1] for more details).

```
(fmod CLASSDIAGRAM is sort ClassDiagram .
  ...
  op classDiagram :  ClassList ObjectList
                     AssocList LinkList -> ClassDiagram .
  op getCDClasses : ClassDiagram -> ClassList .
  ...
  var CLASSES : ClassList . var OBJECTS : ObjectList .
  var ASSOCIATIONS : AssocList . var LINKS : LinkList .
  eq getCDClasses(
        classDiagram(CLASSES, OBJECTS, ASSOCIATIONS, LINKS)) = CLASSES .
  ...
endfm)
```

Fig. 2. Module that formalizes the UML Class Diagram

This specification along with the one shown in the next section will be used to show the application of the formalization of the metamodel of the UML diagrams to guarantee the consistency between models.

In Figure 3, we can see an example of a Class Diagram despicted with a CASE tool. This diagram represents a reservation system and will be the example used in the paper to check the inter-diagrams consistency (section 3).

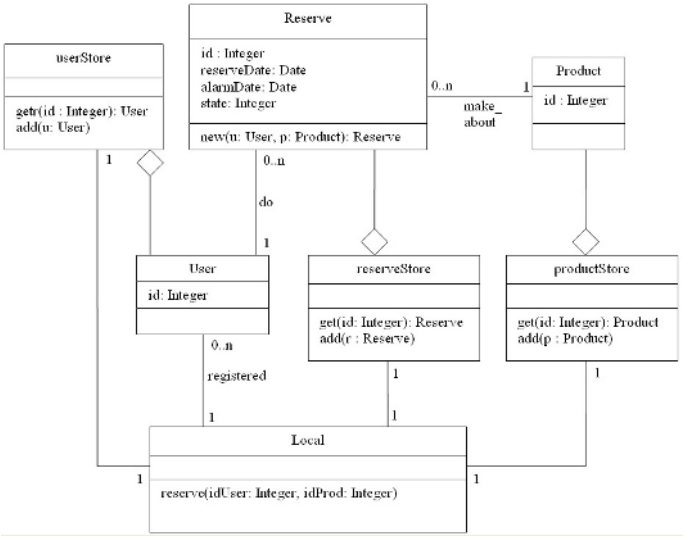

Fig. 3. Example of a UML Class Diagram

2.2 UML Communication Diagram

The next specification need is the one corresponding to the UML Communication Diagram. This type of diagram shows an interaction between objects. One of the most important aspects that is shown in this diagram is the context of the interaction. In Figure 4 appears part of this diagram formalization, the complete formalization used in this diagram is shown with more detail in [2].

```
(fmod COMMUNICATIONDIAG is sort CommunicationDiag .
  ...
  *** Constructor
  op communicationDiag : LifeLineList MessageList -> CommunicationDiag .
  ...

  op getCDLifeLines : CommunicationDiag -> LifeLineList .
  op getCDMessages : CommunicationDiag -> MessageList .

  var LIFELINES : LifeLineList .     var MESSAGES : MessageList .
  eq getCDLifeLines(communicationDiag(LIFELINES, MESSAGES)) = LIFELINES .
  eq getCDMessages(communicationDiag(LIFELINES, MESSAGES)) = MESSAGES .
  ...
endfm)
```

Fig. 4. Module that formalizes the UML Communication Diagram

3 Consistency Between UML Diagrams

Guaranteeing the inter-models consistency or the verification of inter-models properties is one of the most interesting applications of the formalization of the metamodel of UML diagram and this is what we shall see in this section.

As an application of the formalization performed, several properties have been already implemented. In this work, two of them are shown to illustrate the process of consistency verification. Other properties and operations implemented can be found in [8] such as class consistency, correct order of method invocation in a Communication Diagram,...

In the following subsections, the example of Class Diagram shown in Figure 3 will be used in the analysis of the consistency. This consistency verification has been realized between this Class Diagram and the Communication Diagrams which are shown in the Figures 6 and 9.

Although the first property verified, 3.1, is just syntactic, the second property, 3.2, also verifies richer features of the consistency between both diagrams, such as type checks of the parameter in the calls of methods.

3.1 Verification of the Consistency Regarding Associations

In this section we verify the following property:
"For each association that is defined in the Communication Diagram there must exist at least one association in the Class Diagram that allows the sender to send messages to the receiver"

This property guarantees that, for each association defined in the Communication Diagram, there exist at least one association in the Class Diagram that connects the classes that take part in the association of the interaction. In this verification we do not take into account derived associations.

We have two different alternatives to identify an association of the Communication Diagram in the Class Diagram. These are:

1. To compare the name of the association of the Class Diagram, AssocName, with the name that the association (*Connector*) of the communication diagram has.
2. To search associations in the Class Diagram which connect classes of which ClassName is the same ClassName which appears in the *Connectable Elements* (*Connector Ends*) of the association of the communication diagram.

It does not seem practical to require the same, exact, name in the association label of the Communication Diagram and in the association of the Class Diagram. Moreover, frequently this name does not appear, therefore we choose the second alternative as our criterion in the search of associations.

To implement this property, we only search that an association exists in the Class Diagram which connects the classes that are connectable elements in the association (connector) of the communication diagram. Al least one must exist in order to keep the property. Since we do not take into account the association

```
(fmod PROPERTY1 is ...
  op testProp1 : ClassDiagram CommunicationDiag -> String .
  op testProp1 : AssocList ConnectorList -> String .

  var CommunD : CommunicationDiag .
  var CLASSL : ClassList .              var OL : ObjectList .
  var AL : AssocList .              var LL : LinkList .
  var CL : ConnectorList .          var C : Connector .

  eq testProp1(classDiagram (CLASSL, OL, AL, LL), CommunD) =
         testProp1(AL, getCDConnectorList(CommunD)) .
  eq testProp1(AL, empty) = "" .
  *** C CL = ConnectorList. C = head, CL = tail.
  eq testProp1(AL, C CL) =
    if assocExists(AL, getClassName(getCE1(C)),
           getClassName(getCE2(C))) == nullAssoc then
      "ERROR. An association in the Class Diagram "
      + "between the classes " + string(getClassName(getCE1(C)))
      + " and " + string(getClassName(getCE2(C)))
      + ", which is necessary for the association "
      + string(getLabel(C)) + ", does not exist. " + testProp1(AL, CL)
    else testProp1(AL, CL) fi .
 endfm)
```

Fig. 5. Formalization of the property regarding Associations (section 3.1)

name, an association in the Communication Diagram can be "made" by several associations in the Class Diagram.

Figure 5 shows the specification that formalizes the verification of this property. Note that other semantics for the fulfilment of this property could be specified too.

In Figure 6 we see an example of Communication Diagram that does not fulfil this property, and in Figure 7 we can see the property reduction that

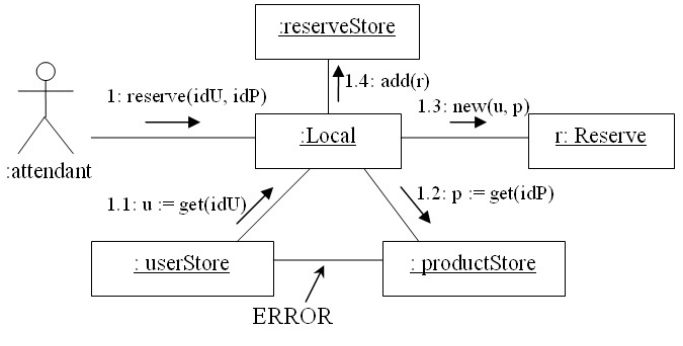

Fig. 6. Example of a UML Communication Diagram that breaks the property verified in 3.1

```
reduce in RESERVE :
  testProp1(classDiagEj,CDreserve)
result String :
  "ERROR. An association in the Class Diagram between
  the classes userStore and reserveStore, which is  necessary for
  the association userSt_resSt_label, does not exist. "
```

Fig. 7. *Maude* reduction of the example of the Figure 6

Maude produces. As we have already indicated, the Class Diagram used as the example is shown in Figure 3.

Another property implemented using this formalization, which is not shown here due to its similarity with the property of this section, is the verification of the consistency of the classes used, in other words, it guarantees that the classes used in the Communication Diagram are present in the Class Diagram to which it belongs.

3.2 Verification of the Consistency Regarding Methods

In this section, we verify that the use of the methods in the Communication Diagram is consistent with the information that appears in the Class Diagram. The property that we want to check is the following:

"The methods used in a communication diagram must be declared in the class diagram and their declaration, with regard to parameters and types, must be correct."

As we have already said, we will verify that the methods that are executed in a *ConnectableElement* (*Connector End*) exist in the class corresponding to their *ClassName*. In the case that this method exists, the property also verifies that the method has the same number of parameters, and the same types, as in the class. Furthermore, the overload of methods has been taken into account in the implementation of the property. The module that verifies this property is shown in Figure 8.

As we can see, we check that each method used by a *ConnectableElement* exists in the class to which it belongs. To do this, first we look for the class and then look for methods with the same *OpName* as in the message of the Communication Diagram.

If no method is found, an error is given as output. If one is found, we check that the number of parameters with the method invoked in the communication diagram is the same as in its definition and that these parameters have the same types. If the method has the same number of parameters but the types are different, the reduction of the property also informs us. The reasons for this error might be that the information of the types has not been included in the communication diagram or that this information has been included incorrectly.

```
(fmod PROPERTY2 is...
  op testProp2 : ClassDiagram CommunicationDiag -> String .
  op testProp2 : ClassList MessageList -> String .
  *** Params: operation list of a class, receiver and its message list.
  op testProp2 : OpList ConnectableElement Message -> String .
  *** Verify that the messsage is among the OpNames from OpList.
  op testProp2 : Message OpList ConnectableElement Int -> String .
  ...
  eq testProp2(classDiagram(CL,OL,AL, LL),
      communicationDiag(LifeLineL, ML)) = testPropInter2 (CL, ML) .

  eq testProp2(CL, empty) = "" .
  eq testProp2(CL, M ML) =
       testProp2(getOperations(getClassbyName(getClassName(
         getReceiver(M)), CL)), getReceiver(M), M)
       + testProp2(CL, ML) .

  eq testProp2(OpL, CE, M) =
    if findOps (OpL, M) =/= nullOp then
      testProp2(M, findOps(OpL, M),CE,0)
    else
      "ERROR. The message (" + string(getMsgNumber(M))
      + ". " + string(getMsgLabel(M)) + ") "
      +" doesn't exist in the class "+ string(getClassName(CE)) + ". "
    fi .
  ...
  eq testProp2(M, nullOp, CE, I) =
    if I == 0 then "ERROR. The signature of the message ("
      + string(getMsgNumber(M)) + ". " + string(getMsgLabel(M)) + ") "
      +"doesn't concur with the method of the class " +
      +string(getClassName(CE))+" neither in number of parameters nor "
      + "type of them. "
    else "ERROR. The signature of the message ("+string(getMsgNumber(M))
      + ". " + string(getMsgLabel(M)) + ") "
      +" doesn't concur with the method of the class "
      + string(getClassName(CE))
      + " because the types of the parameters are not correct. "
    fi .
  eq testProp2(M, Op OpL, CER,  I) =
    if length(getOpParamList(Op)) == length(getMsgParamList(M)) then
      if getTExpr(getOpParamList(Op)) == getTExpr(getMsgParamList(M))
             then ""
      else testProp2(M, OpL, CE, I + 1) fi
    else testProp2(M, OpL, CE, I)
    fi .
 endfm)
```

Fig. 8. Formalization of the property regarding Methods (section 3.2)

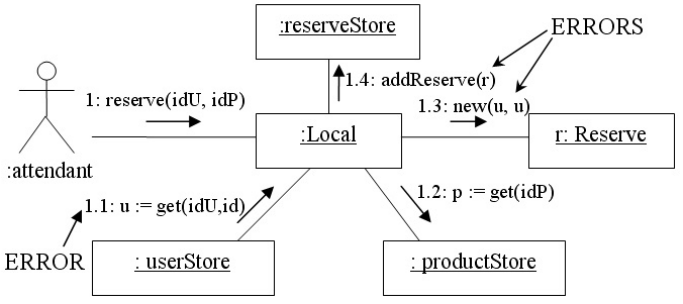

Fig. 9. Example of a UML Communication Diagram that breaks the property verified in section 3.2

```
reduce in RESERVE :
  testProp2(classDiagEj,CDreserve)
result String :
  "ERROR. The signature of the message (\001\001. get) doesn't concur
  with the method of the class userStore neither in number of
  parameters nor type of them.
  ERROR. The signature of the message (\001\003. new)  doesn't concur
  with the method of the class Reserve because the types of the
  parameters are not correct.
  ERROR. The message (\001\004. addReserve)  doesn't exist in the class
  reserveStore. "
```

Fig. 10. Reduction in *Maude* of the property verified in section 3.2 on the example of the Figure 9

Finally, in the verification of this property, we cannot take into account the syntactic identity of the name of the parameters, since the identifiers that appear in the method definition, which are called *formal parameters*, cannot be the same as the identifiers of the parameters in the invocation, called *actual parameters*, and which will replace the formal parameters in the body of the method.

Figure 9 shows an example that contains the three possible errors that this property detects. The first error is the message *1.4 addReserve*, which does not exist in the class *reserveStore*. The second error is found in the message *1.1 get(idU,id)*, this method exists in the class *userStore*, but does not have the same numbers of parameters. The last error is produced in the message *1.3 new(u,u)* (the method is declared in the class *Reserve*) also has two parameters, but the type of the second parameter is *User* instead of *Product*. The property reduction are shown in the Figure 10.

4 Related Work

In [6], a general and updated view of the consistency problems within the UML based development process are given. In this work, the use of techniques to avoid

problems of consistency is justified, because UML is considered as a standard in the development of systems.

The approach of formalizing the UML metamodel to guarantee the correct development of models has been dealt with in many papers, although the formalization of the UML Communication Diagram has not been deeply studied. Most of the approaches formalize other UML diagrams such as UML Class or Statechart diagrams.

In [9], a proposal to verify UML models using B abstract machines for UML Class Diagram is presented. Another paper [10] tackles the formalization of UML models and discusses the integrity consistency check between different models. In this approach, the formal language Object-Z is used, which allows the authors to implement each UML element as a class. None of these approaches offers the possibility of making a automatic translation from the models to the formal specification. Unlike them the formal framework presented in this paper is integrated with automatic translators that obtain the specification that represent a model from the model depicted with a CASE-tool.

Some research [11] has been done on formalization of the UML Statechart diagram. This approach uses the SPIN model checker to perform the verification. The tool verifies several properties and generates a sequence diagram that shows how to reproduce the error in the model. However, this work suffers from some problems, such as a poor efficiency of the implementation. In the Maude design, efficiency has been considered from its beginning, resulting in a fast execution of the reductions and rewrites.

5 Conclusions

This work presents a formal approach to improve the inter-diagrams consistency. The specifications created in [1, 2, 3, 4] offer a good framework to guarantee the model consistency, because all the specifications are integrated.

Furthermore, this formal framework has been revealed as a useful instrument to realize verification of properties, both intra-model and inter-model. Modifying the semantics of existing property specifications and/or adding new property specification is very easy, once the formal framework (basic sorts, operations and equations) is available. Another possible application is the realization of precise transformations that help to find better models.

As further work, we will continue to implement properties to improve the diagrams' consistency. Moreover, we are searching for real case studies to justify the use of algebraic specification within MDA. The integration of the Communication and Statechart Diagram is another of the research lines that we are working on, in order to verify that each object that takes part in the communication diagram has a consistent state when the interaction finishes. Another future work is the application of this approach on tools for the development of Web Information Systems (WIS), like MIDAS-CASE[12].

References

1. Fernández Alemán, J.L., Toval Álvarez, A.: Improving System Reliability via Rigorous Software Modeling: The UML Case. Proceedings of the 2001 IEEE Aerospace Conference (track 10: Software and Computing), Montana, USA IEEE Computer Society (2001)
2. Lucas Martínez, F.J., Toval Álvarez, A.: Formal Verification of Properties in the UML Collaboration Diagram. ICSSEA 2004: 3rd Workshop on SYSTEM TESTING AND VALIDATION. Paris (2004)
3. Fernández Alemán, J.L., Toval Álvarez, A.: Can Intuition Become Rigorous? Foundations for UML Model Verification Tools. International Symposium on Software Reliability Engineering, Published by IEEE Press (2000)
4. Whittle, J., Araújo, J., Toval Álvarez, A., Fernández Alemán, J.L.: Rigorously Automating Transformations of UML Behavior Models. Dynamic Behaviour in UML Models: Semantic Questions in conjunction with UML 2000 York, UK , ACM SIGSOFT, IEEE Computer Society (2000)
5. OMG: Object Management Group. UML Superstructure 2.0. Draft adopted Specification. Retrieved from: http://www.omg.org/uml. (2004)
6. Huzar, Z., Kuzniarz, L., Reggio, G., Sourrouille, J.L.: Consistency Problems in UML-based Software Development. In UML Modeling Languages and Applications. UML 2004 Satellite Activities Lisbon, Portugal, October 11-15, 2004 Revised Selected Papers. Jardim Nunes, N. Selic, B., Silva, A. Toval, A. (Eds.) **Springer Verlag vol. 3297 of LNCS** (2004)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcote, C.: Maude 2.0 Manual. Versión 1.0, http://maude.csl.sri.com/. (2003)
8. Lucas Martínez, F.J., Toval Álvarez, A.: Algebraic Specification of the UML Collaboration Diagram and Applications. Tech. Rep. LSI 3, Department of Informatics and Systems. University of Murcia. (2004)
9. Truong, N.T., Souquieres, J.: An approach for the verification of UML models using B. 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04) (2004)
10. Kim, S.K., Carrington, D.: A Formal Object-Oriented Approach to defining Consistency Constraints for UML Models. 2004 Australian Software Engineering Conference (ASWEC'04) (2004)
11. Litius, J., Porres Paltor, I.: vUML: a Tool for Verifying UML Models. 14th IEEE International Conference on Automated Software Engineering (1999)
12. Vara, J., de Castro, V., Cáceres, P., Marcos, E.: Arquitectura de MIDAS-CASE: una herramienta para el desarrollo de SIW basada en MDA. 4ª Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento. Madrid (2004)