UML MODEL VERIFICATION THROUGH DEPENDENCY RELATIONSHIPS

Mouez Ali, Hanène Ben Abdallah
Laboratoire LARIS, Faculté de Sciences Economiques et de Gestion, Sfax, B.P. 1088, 3018 Sfax, Tunisia

Faiez Gargouri
Laboratoire LARIM, Institut Supérieur d’Informatique et du Multimédia, Sfax, B.P. 1030, 3018 Sfax, Tunisia

Keywords: UML, Unified Process, Modular Verification

Abstract: The Unified Modeling Language (UML) has merged as a de-facto standard for modeling especially information systems. However, in spite of its wide spread usage, UML still lacks support for verification methods and tools. Several researches proposed verification methods for certain UML diagrams, however, none of the proposed methods covers all of the UML diagrams, which are semantically overlapping in any system model.

In this paper, we propose a modular verification method for UML models. The proposed method uses the implicit (semantic) and explicit (syntactic) relations among all the diagrams of a UML model. The implicit inter-diagram relations are deduced from the UP design process. In this paper, we overview the proposed method and illustrate its feasibility through an information system example.

1 INTRODUCTION

UML is a semi-formal language for visualizing, specifying, constructing and documenting artifacts of software systems (UML Group, 1997). It is becoming the dominant object-oriented modeling language for the design of information systems.

To specify the different aspects of a system, the UML notation proposes nine diagrams. The various diagrams of a UML model are explicitly related through the syntactic rules of UML (UML Group, 1997). For instance, each use case in the use case diagram is represented by at least one sequence diagram, and each object in the sequence diagram is an instance of a class in the class diagram. In addition, depending on the adopted specification process, a model’s diagrams are also implicitly related. For example, following the Unified Process (Jacobson et al., 1999), a use case model is specified by the analysis model which is expressed through a collaboration diagram, a set of sequence diagrams and an activity diagram.

On the other hand, with the increasing complexity of today's systems, the need for a rigorous development process is ever pressing for verification methods. In practice, two verification techniques are generally used: peer review and software testing. However, only 15% of errors can be detected during the design phase and the reparation cost of errors is 500 times greater during the maintenance phase than during the design phase (Utwente, 2002). In addition, 2/3 of the bugs come from the analysis and design activities while 1/3 of the bugs come from the implementation activity (Printz, 1997). Thus, in order to reduce the overall software development cost, formal verification tools are required to verify that the requirements can be fulfilled by the specification and to detect specification errors in an early phase of the design process.

In the case of UML-based models, verification techniques must face two obstacles. The first stems from the fact that a UML model typically includes various diagrams. The second obstacle is due to the lack of a unique formal semantics for all of the UML diagrams. These two obstacles motivated us to look for a modular analysis technique that would render the verification of a UML model into an analysis of certain diagrams of the model. This approach can
benefit from the various semantics and analysis techniques proposed for some UML diagrams, c.f., (Latella et al., 1999); (Paludetto et al., 1999).

Our modular verification exploits the implicit (i.e., semantic) and explicit relationships among the various UML diagrams of a system model. The implicit relations among the diagrams of a model are deduced from the Unified Process (Jacobson et al., 1999) where a system model is derived incrementally. The explicit relations are deduced from the UML syntax (UML group, 1997). Thus, our modular verification approach defines formally the two types of relations among a UML model’s diagrams. This formal definition eliminates several problems such as wrong model interpretations and inconsistent diagrams within the same system model (Pons et al., 2002).

In Section 2, we briefly overview proposed verification methods for UML. In section 3, we present the Unified Process (UP) (Jacobson et al., 1999) and the implicit relations among the various diagrams of an UML model generated through UP. In section 4, we first present our verification method that uses the implicit and explicit (syntactic) relations among a UML model’s diagrams. Secondly, we illustrate our verification method through an information system example. Section 5 summarizes the paper and outlines our future work.

2 RELATED WORKS

Several researchers have proposed a precise description of UML concepts and provided rules to verify when a system model satisfies a given property. Overall, most of the proposed approaches focus on models described through one type of diagram that describes either the static, dynamic or functional aspects of the system. In addition, the proposed approaches either rely on the meta-model of UML or on the translation of UML diagrams to a formal language.

2.1 Meta-model Verification

This technique combines the graphical notation, natural language and a formal language. It gives a syntactic description of the language. It’s described in (Evans et al., 1999) as following:

“The UML semantics is described using a meta-model that is presented in terms of three views: the abstract syntax, well-formedness rules, and modeling element semantics. The abstract syntax is expressed using a subset of UML static modeling notations. The abstract syntax model is supported by natural language descriptions of the syntactic structure of UML constructs. The well formedness rules are expressed in the Object Constraint Language (OCL) and the semantics of modeling elements are described in natural language. The advantage of using the meta-modeling approach is that it is accessible to anybody who understands UML.”

Thus, the meta-model of UML gives a precise notion of only the abstract syntax and does not consider the semantics. OCL is usually used to explain constraints that must hold for a model to be well-formed.

2.2 UML Verification through Diagram Formalization

In the formalization of object-oriented concepts, there are three general approaches: 1) the supplemental approach, where an informal parts of a model are expressed in natural language (c.f., Cook et al., 1994); 2) the OO-extended approach, where an existing formal notation, e.g., Z (Michael, 1992) and VDM (Lucas, 1987), are extended with Oriented-Object features, e.g., Z++(Lano, 1991), Object-Z (Duke, et al., 1991), VDM++ (Dürr et al., 1993); and 3) the integration approach which generates a formal specification from an informal object-oriented model, c.f., (Anthony, 1990; France, 1998; Roebert et al., 1995).

As for the formalization of UML, the majority of works focuses on one aspect of a model. In general, the dynamic aspect expressed by for instance the statecharts diagram. For example, the author in (Latella et al., 1999) translates UML statecharts diagram to Promela (PRoCess Meta LAnguage), the specification language of SPIN (Holzmann, 1997) tool. This language creates a communicating automaton checked by SPIN. A similar approach was adopted by (Kwon, 2000), who translates a UML statecharts to SMV (Symbolic Model Verifier) (McMillan, 1992) and checks the statecharts diagram with the SMV modeler checker. In (Paulo, et al., 2000), the authors translate a part of UML models to a LOTUS specification (ISO. LOTOS, 1985) and check the specification with the CADP (Caesar/Aldebaran Development Package) check box tool (Fernandez et al., 1992); finally, the authors in (Paludetto et al., 1999) translate the statecharts diagram to Petri Nets.

The above works define a precise syntax and semantics for isolated UML diagrams without dealing with the relations between the various diagrams in an UML model.
3 INTER-DIAGRAM RELATIONS IN UP

Several design processes have been proposed to derive a UML model, e.g., the Unified Process (UP) (Jacobson et al., 1999), Catalysis (D’Souza et al., 1999) and Rational Unified Process (RUP) (Kurchten, 1999).

UP is an iterative and incremental process. As illustrated in Figure 1, during this process, a UML model is created and refined during consecutive iterations and phases in the development process. In each development phase, a system model is derived using certain UML diagrams, depending on the focus of the phase. Table 1 lists the UML diagrams used to construct a model for each of the UP development phases.

As shown in Figure 1, the various models derived following UP are not independent. In fact, they are semantically overlapping and complementary when presenting the target system as a whole unit. Since, each model is a set of UML diagrams (see Table 1); the UP model dependencies implicitly imply dependencies among the UML diagrams used.

Figure 1: UP models and their relationships (Jacobson et al., 1999)

Table 1: UML diagrams used in UP models (Jacobson et al., 1999)

<table>
<thead>
<tr>
<th>Activities</th>
<th>UP Models</th>
<th>UML Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Capture</td>
<td>Use case model</td>
<td>Use case diagram</td>
</tr>
<tr>
<td>Analyze</td>
<td>Analysis Model</td>
<td>Collaboration Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sequence Diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Activity Diagram</td>
</tr>
<tr>
<td>Design</td>
<td>Design Model</td>
<td>Class Diagram</td>
</tr>
<tr>
<td>- System</td>
<td></td>
<td>Object Diagram</td>
</tr>
<tr>
<td>- Interface</td>
<td></td>
<td>Package Diagram</td>
</tr>
<tr>
<td>- Data Base</td>
<td></td>
<td>State-Transition Diagram</td>
</tr>
<tr>
<td>Distribution</td>
<td>Distribution Model</td>
<td>Deployment Diagram</td>
</tr>
<tr>
<td>Implementation</td>
<td>Implementation Model</td>
<td>Component Diagram</td>
</tr>
<tr>
<td>Test</td>
<td>Test Model</td>
<td></td>
</tr>
</tbody>
</table>
UP highlights the implicit relationships among a system models such as “specified by”, “realized by”, “implemented by”; etc (see Figure 1). These implicit relations will be the basis of our approach to a modular verification of UML models.

4 MODULAR VERIFICATION OF UML MODELS

To manage the complex analysis of a UML model and verify both the syntactic and semantic aspects of a system model, we propose to:

1. define a model’s inter-diagram implicit relations as proposed by UP, and
2. use the defined relations (along with the UML syntactic dependencies) to infer that a UML model satisfies a given property by verifying that certain diagrams of the model satisfy the property.

We next introduce several definitions that we use to formalize the inter-diagram relations and to provide for a modular verification of an UML model.

4.1 Definitions

Similar to UP, we suppose that a UML system model (M) is a set of UML diagrams (Jacobson et al., 1999) that represent different aspects of the system at a certain level of abstraction. A model describes the static, dynamic or functional aspects of a system. Formally, a system model \( M \) is the structure \( \langle D, R_{inter} \rangle \) where \( D \) is a set of UML diagrams and \( R_{inter} \) is a set of relationships between diagrams in \( D \). The set of inter-diagram relations \( R_{inter} \) is the object of this paper. It is detailed in the next section.

In general, a diagram \( D \) is the structure \( \langle E, R_{intra} \rangle \) where
- \( E \): a set of structural elements (e.g. use cases, actors, classes,)
- \( R_{intra} \): a set of relations between the structural elements, defined according to the UML syntax and semantics (UML Group, 1997).

For the complete set of UML structural elements, we refer the reader to the UML semantics (UML Group, 1997). In addition, the meta-model of UML details out the relationships between the structural elements of each UML diagram (UML Group, 1997). Due to space limitations, we next detail out only the three diagrams used in this paper to illustrate our verification method: the remaining UML diagrams can be defined in a similar manner.

A use case diagram is the structure
\[
U_c \triangleq \langle A \cup U, R_{intra} \rangle
\]
- \( A = \{ a_1, \ldots, a_n \} \) is a set of actors,
- \( U = \{ u_1, \ldots, u_m \} \) is a set use cases each of which is described by a sequence of actions,
- \( R_{inter} \) is a set of relations from the set \{ <<extend>>, <<include>>, <<communicate>>, <<Generalize>> \} and defined between the structural elements \( A \cup U \) according to the UML syntax (UML Group, 1997).

A collaboration diagram is the structure
\[
C_o \triangleq \langle O, L \cup M \rangle
\]
- \( O = \{ o_1, \ldots, o_u \} \) is a set of objects,
- \( L = \{ l_1, \ldots, l_v \} \) is a set of links between objects,
- \( M = \{ m_1, \ldots, m_w \} \) is a set of messages between objects.

4.2 Inter-diagram relations in a UML Model

Our verification method exploits the implicit inter-diagram relations inspired from UP. These relations are defined between the diagrams of a given UML system model. They are presented in Figure 2 as a directed, labeled graph. In the graph, vertices represent UML diagrams and edges represent the dependency relations between the diagrams.

We call the set of relations defined in the inter-diagram dependency graph of Figure 2 the implicit relations. These relations reflect the inter-model relations presented in the UP design process (see figure 1). We can therefore surcharge their definition over the set of system models as follows:

**Definition 1**

Let \( M_1 = \langle D_1, R_1 \rangle \) and \( M_2 = \langle D_2, R_2 \rangle \) be two system models. We say, “\( r \) is an implicit relation between \( M_1 \) and \( M_2 \)”, if and only if there exists \( D_I \in D_1 \) and \( D_J \in D_2 \), such that \( r \) is an implicit relation between \( D_I \) and \( D_J \).
For example, (see Figure 3), according to the UP, the use case model of a system is specified by its analysis model. Thus, we can deduce that the use case diagram (in the use case model) and the collaboration diagram (in the analysis model) are related by the \( \text{spec} \) relation.

Informally, a use case diagram \( U_c \) is specified by a collaboration diagram \( C_o \), denoted as \( U_c \xrightarrow{\text{spec}} C_o \), if and only if for each structural element \( e \) in \( U_c \), there is a set of structural elements \( \{e_i\} \) in \( C_o \) that describes the same concept represented by \( e \). The following examples represent the concept correspondence between the structural elements of \( U_c \) and those of \( C_o \):

- each actor in \( U_c \) is represented by an object in \( C_o \);
- each action in \( U_c \) textual description is represented by a message in \( U_c \).

### 4.3 UML Model Verification

The inter-diagram relations defined above can be exploited to reduce the verification of a UML model to the verification of some of its diagrams and not all of them.

A UML diagram \( D \) satisfies a logic formula \( P \) \( (D \models P) \) is defined inductively based on the type of \( D \) and the syntax of \( P \). For instance, let us define the case when \( D \) is a use case diagram.

**Definition 2**

A use case diagram \( U_c \) satisfies the prepositional logic formula \( P \) \((U_c \models P)\) if and only if:

1. there exists either:
   a. an actor \( a \in A \) that represents \( P \) and \( P \) is a predicate; or
   b. a use case \( u \in U \) “representing” \( P \) and \( P \) is a predicate.
2. if \( P \) expressed in terms of the \( p_1, p_2 \) predicates and
   a. there exists a relation \( a \xrightarrow{\text{communicate}} u \) \( e \in R_{\text{sigma}} \) such that \( a \) represents \( p_1 \) and \( u \) represents \( p_2 \) and \( P = p_1 \land p_2 \);
   b. there exists a relation \( u_1 \xrightarrow{\text{include}} u \in R_{\text{sigma}} \) such that \( u_1 \) represents \( p_1 \) and \( u_2 \) represents \( p_2 \) and \( P = p_1 \Rightarrow p_2 \);
   c. there exists a relation \( u_1 \xrightarrow{\text{extend}} u_2 \in R_{\text{sigma}} \) such that \( u_1 \) represents \( p_2 \) and \( u_2 \) represents \( p_2 \) and \( P = p_2 \Rightarrow p_1 \).
3. if \( P = \neg P' \), then \( U_c \) does not satisfy \( P' \).
4. if \( P_1, P_2 \) are two formulas, and
   a. \( P = P_1 \lor P_2 \) then either \( U_c \models P_1 \) or \( U_c \models P_2 \).
   b. \( P = P_1 \land P_2 \) then \( U_c \models P_1 \) and \( U_c \models P_2 \).

**Definition 3**

A UML model \( M \) satisfies a property \( P \) (\( M \models P \)) if and only if each diagram \( D \) in \( M \) satisfies \( P \) (\( D \models P \)).

To illustrate our modular verification approach, the following proposition states how the specification relation \(<\text{spec}>\) can be used to reduce the verification of a UML model to the verification of some of its diagrams.

**Proposition**

Let \( D_1 \) and \( D_2 \) be two UML diagrams and \( P \) be a prepositional logic formula. If \( D_1 \models P \) and \( D_2 <\text{spec}>D_1 \), then \( D_2 \models P \).

Using this proposition, if \( U_c \models P \) then for each diagram \( D \) reachable from \( U_c \) through edges labeled with \(<\text{spec}>\), we have \( D \models P \). Thus, we only need to verify the remaining diagrams of the model.

**4.4 Example**

In this section, we illustrate our UML model verification approach using the Automatic Teller Machine (ATM) example. Figure 4 shows a part of the ATM model which is expressed through a use case diagram \( A_c \) and collaboration diagram \( C_c \). The ATM use case diagram, shown in Figure 4-a, specifies the functional aspect of the ATM system. A part of the ATM dynamic system aspect is described by an activity diagram \( A_o \) and collaboration diagram \( C_o \). The ATM use case diagram, shown in Figure 4-a, specifies the functional aspect of the ATM system. A part of the ATM dynamic system aspect is described by an activity diagram \( A_o \) and collaboration diagram \( C_o \).

Suppose we want to verify the following ATM property \( P \): “every customer can withdraw money if he/she can be identified by the bank host”. To express formally \( P \), we use the following predicates and propositions:

- \( \text{customer}(c) : c \) is a customer.
- \( \text{identification}(c) : c \) is identified as a client.
- \( \text{withdraw\_money}(c) : c \) withdraws money.

Thus, the property \( P \) can be expressed formally by the following proposition:

\[ \text{withdraw\_money}(c) \rightarrow (\text{customer}(c) \land \text{identification}(c)). \]

To prove that the ATM model satisfies the property \( P \), we follow the next two steps:

1. verify that \( U_c \models P \).
2. verify that \( U_c <\text{spec}> A_c \).

Verifying that \( U_c \models P \) can be easily through a syntactic analysis of the diagram and a correspondence among the use case diagram structural elements and formula components. Let us establish the following trivial correspondence between the predicates of \( P \) and the use case diagram elements:

- the actor “customer” represents the predicate \( \text{customer}(c) \).
- the use case “withdraw\_money” represents the predicate \( \text{withdraw\_money}(c) \).
- the use case “identification” represents the predicate \( \text{identification}(c) \).

In this example, we have:

\[ P = \text{withdraw\_money}(c) \rightarrow (\text{customer}(c) \land \text{identification}(c)). \]

Thus, \( P = (\text{withdraw\_money}(c) \rightarrow \text{customer}(c)) \land (\text{withdraw\_money}(c) \rightarrow \text{identification}(c)) \).

Hence, according to definition 1, we have:

1. \( U_c \models (\text{withdraw\_money}(c) \rightarrow \text{identification}(c)) \) since the relation “customer” \(<\text{communicate}>\) withdraw\_money” is in \( U_c \), and
2. \( U_c \models (\text{withdraw\_money}(c) \rightarrow \text{identification}(c)) \) since the relation “withdraw\_money” \(<\text{include}>\) identification” is in \( U_c \).

Using this syntactic inspection of the ATM use case diagram, we easily conclude that \( U_c \models P \).

In the second step, we verify that \( U_c <\text{spec}> C_c \) and \( U_c <\text{spec}> A_c \). The verification of the \(<\text{spec}>\) relation between \( U_c \) and \( C_c \) is based on a list of UML syntactic and semantic rules that relate the two diagrams. It used detailed inspections of the structural elements of the use case diagram (e.g., the pre- and post-conditions and the guards of the activity diagram).

In this step, it must be confirmed that the collaboration diagram and activity diagram are able to cope with the use case diagram.

Finally, we have \( U_c \models P \) and \( U_c <\text{spec}> C_c \), and \( U_c <\text{spec}> A_c \).

Thus by this proposition \( C_c \models P \), and \( A_c \models P \).
According to Definition 3, we therefore conclude that the ATM model (with its presented diagrams) satisfies the property \( P \) (ATM model \( \vdash P \)).

5 CONCLUSION AND FUTURE WORKS

During the Unified Process, a variety of UML-based models of a system are developed. These models are implicitly related to one another and are semantically overlapping and complementary. The implicit relations among a UML model’s diagrams derived through UP (together with the UML syntactic dependency relations) can be exploited in a modular verification of a UML model. In this paper, we showed the feasibility of this verification approach using a simple example that contains only tree diagrams and one implicit relationship \(<\text{spec}>\).

We are currently completing the formalization of the implicit inter-diagram relations and examining the set of properties verifiable through this approach.
REFERENCES


