Consistency Checking in UML Models

Bogumiła Hnatkowska*
Hnatkowska@ci.pwr.wroc.pl

Zbigniew Huzar*
Huzar@ci.pwr.wroc.pl

Jan Magott**
Magott@ict.pwr.wroc.pl

Abstract: The paper considers the problem of consistency among components of a UML system model. We propose OCL (Object Constraint Language) to formalize the consistency conditions that must hold between model components. The following consistencies of the diagrams are analysed: a state diagram of a class vs. a full class descriptor of the class, a full class descriptor of a class vs. state diagram of the class, a state diagram of a class vs. a class diagram, and a state diagram of a class vs. state diagrams of other classes.

Key Words: consistency checking, object-oriented system modeling, UML

1 Introduction

The UML [3], [17] is the main standard for object-oriented modelling. It is a visual modelling language that can be used in all phases of software development. It incorporates a number of influences that cater for different modelling preferences.

In research concerning the UML, at least three directions can be recognised. The first one consists in searching of deficiencies of the language, e.g., papers [1], [18]. The other field of interest is adaptations of the UML language to different application domains. For example, in the paper [10], an extension towards multicast synchronisation is presented. The main direction of the research deals with defining of formal semantics for the UML, e.g., papers [8], [9], [12], and usage of formal definitions in modelling process. Formally defined notions enable strict verification of consistencies among diagrams. In the paper [1], the problem of consistency of class diagrams has been studied.

There are many approaches of software development based on UML notation, e.g. [4], [7]. These approaches determine the phases of software development, and contents of models built within the phases. The model defined at a given phase represents a considered system from a certain point of view and consists of a set of different UML diagrams.

Software developers are faced with two main problems during a process of model construction: (a) the problem of consistency among different diagrams within a given model, and (b) the problem of consistency between two different models. We are going to investigate a con-
sistency within a model expressed in the UML. The term “consistency” of a model is understood as correctness of the model on the level of static semantics.

Consistency conditions are necessary to be defined in a software development process. Universally defined conditions, i.e., independently of a given modelling process, enable upgrading of programming tools supporting software development process. Recently, CASE packages, for example Select, or StP/UML, supporting software development with UML offer rather limited scope of consistency checking.

The paper considers the problem of consistency among selected UML diagrams: state diagrams, activity diagrams, class diagrams, and their supplement – full class descriptors. The diagrams can be considered as basic components of models that fully grasp static and dynamic aspects of a modelled system. However, the model may be incomplete.

We have chosen them because, if fully defined and extended with an initial object diagram, they may constitute an executable model of a system. Object diagrams are instances of a class diagram and consistency between them can be checked easily. Therefore, this kind of consistency is omitted in the paper.

Our aim is to identify kinds of inconsistency between model elements and express the formally. Consistencies among diagrams are expressed formally in OCL (Object Constraint Language) [15]. OCL has been used to define static semantics of the UML metamodel, also to define constraints on UML diagrams. There are programming tools supporting manipulation on OCL formulas.

In the paper we present only few examples of OCL expressions that define consistency conditions. A comprehensive set of OCL expressions that define formally conditions for all kinds of consistency presented in this paper are contained in [11].

OCL formulas defining consistency conditions can be considered as a logical specification of system properties for programming tools automatically checking consistency of a model.

Being a formal language, OCL enables automatic processing. That is, if an UML model is enriched with OCL constraints, the special software application, known as OCL Compiler, can be used to answer the following questions automatically:

1. If the model well-formedness rules are, indeed, written in a correct OCL language?
2. Do these well-formedness rules actually correspond to the model they are describing?
3. Do these well-formedness rules conflict with each other?

There are several examples of OCL tools, e.g. [19], [20], that does full type checking of OCL constraints and includes code generation from OCL to different programming languages.

The paper is organised as follows. Presentation of consistency relations between components of the model is given in Section 2. Final remarks are contained in the last Section 3.

2 Survey of consistency relations

We concentrate on inter-diagram consistency. For a given diagram, we check its consistency with respect to other diagrams. We do not consider inner-diagram consistency. We also give short comments about computational complexity of algorithms that are behind the formulas.

We consider consistency relation among three kinds of diagrams taking into account information included in full class descriptors. These diagrams are:

1. class diagram
2. state diagram
3. activity diagram
The full class descriptor contains the complete definition of a given class. The full descriptor is necessary because class diagrams may be presented on different levels of abstraction, for example, they can omit class attributes or operations.

Most of possible pairs of diagrams for inter-diagram consistency comparison are meaningless. The reasonable pairs of diagrams for comparison are grouped in the Table. 1.

<table>
<thead>
<tr>
<th></th>
<th>Class descriptor</th>
<th>Class diagram</th>
<th>State diagram</th>
<th>Activity diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class descriptor</td>
<td>X</td>
<td>X</td>
<td>3</td>
<td>X</td>
</tr>
<tr>
<td>Class diagram</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>State diagram</td>
<td>1</td>
<td>4</td>
<td>5*</td>
<td>X</td>
</tr>
<tr>
<td>Activity diagram</td>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1. Consistencies among model elements. Number in the table refers to a group (defined further) of consistency rules. Symbol X denotes that there is no need for comparison. The asterisk refers to a comparison of a given state machine diagram with respect to other state machine diagrams.

2.1 Consistency of a given state diagram of a class with respect to the full descriptor of this class

For each call and send event it should exist respective declaration of an operation or a signal in the full class descriptor.

The rule is expressed by the two following OCL invariants, where c is a class associated with a given state machine:

```ocl
context ClassDescription :: StateMachine -- [a]
inv:
let c: Class = self.class in
self.event() -> forall(e: Event | e.oclIsKindOf(CallEvent) implies (c.operations -> includes(e.operation.name)))
```

```ocl
context ClassDescription :: StateMachine -- [b]
inv:
let c: Class = self.class in
self.event() -> forall(e: Event | e.oclIsKindOf(SendEvent) implies (c.signals -> includes(e.signal.name)))
```

Let $e_i$ be the number of events in the set $event()$ of the state machine of the $i$-th class. Let $o_i$ and $sg_i$, respectively, be the number of operations and the number of signals, respectively, of the full class descriptor of the $i$-th class. Computational complexity of verification of condition [a] and [b], respectively, is equal to $O(e_i o_i)$ and $O(e_i sg_i)$, respectively.
2.2 Consistency of a given activity diagram with respect to a full descriptor of a respective class

A simplified state machine represents an activity diagram. States in the machine have not entry, exit and do compartments. An activity diagram is a specification of a method of some operation. Transitions in the machine are mainly the result of anonymous internal events, which represent completion of an activity at a state. Transitions may also be triggered by signal, time and change events. Each signal event should have respective declaration of a signal in the full class descriptor.

The state machine may send signals to or may call operations from other objects. The state machine may also call operations from the same object provided the operations are concurrent with respect to the operation represented by a given activity diagram. (The concurrency between operations should be distinguished from the property of a given operation expressed by its concurrency attribute, which allows concurrent calls to this operation.) The last stipulation implies from the fact that the call of an operation of a given object while executing another operation of the same object may bring to a deadlock. In further, we will not examine this aspect.

The considered consistency conditions of state machines for activity diagrams are the same as conditions of state machines for classes; therefore they are not presented here.

2.3 Consistency of a full descriptor of a class with respect to the state diagram of this class

Each operation, except constructor, is expected to have a respective event among triggering events, or a respective action among actions in associated state machine. Each signal is expected to have a respective event among triggering events in associated state machine.

This demand is not obligatory but it is expected in properly defined state diagram for a given class. If is not satisfied, it should be signalled at least as a warning.

2.4 Consistency of a given state diagram of a class with respect to a class diagram

Each action in a given state machine of a given class $c$ should refer to a respective class $c_1$ that is accessible from $c$. Meaning of the adjective ‘respective’ depends on a kind of action.

For each create action the following conditions must hold:

- a) The class that is pointed as a type of created instance must be associated with the class $c$.
- b) Create action results in a call of a class constructor, which is a specific class-scope operation. By convention its name is the same as the class name of created instance.
- c) The number of actual arguments of the action must match the number of formal parameters of the constructor, and the type of a given argument must conform to the type of respective formal parameter.
- c) If the type of created instance is different from $c$, then the calling constructor must have public visibility.

For each call action, which calls an operation of some object:

- a) The class of a target object must be associated with a given class $c$.
- b) The operation name must belong to operation names of the class of a target object.
- c) The number of actual action arguments must match the number of formal parameters of the operation, and the type of a given argument must conform to the type of respective formal parameter.
d) If an action results in calling an operation of a class different from \( c \), then the operation must have public visibility.

For each send action, which sends a signal to a set of objects:
   a) The class of a target object must be associated with a given class \( c \).
   b) The signal name must belong to signals of the class of a target object.
   c) The number of actual action arguments must match the number of formal parameters of the signal, and the type of a given argument must conform to the type of respective formal parameter.

For each assignment action, which assigns a value to an attribute or link.
   a) If a value is assigned to an attribute, the attribute must be declared in a given class, or must be declared as public in a class being in association with a given class.
   b) If a value is assigned to a link, the name of the link must be included in the set of names of navigable association ends of a given class \( c \).

For each return action:
   a) The class of a target object must be associated with a given class \( c \).

For each destroy action:
   a) The class of the destroyed instance must be associated with a given class \( c \).

2.5 Consistency of a given state diagram of a class with respect to state diagrams of other classes

Each event in a given state machine of a given class \( c \) should have a source (an action) in a class \( c_1 \) that has access to the class \( c \).

For example, for each call event or signal event in the state machine \( sm \), a state machine \( sm_1 \) of the class \( c_1 \) should contain a transition or a state with the action, which is a source of the event. Moreover, the class \( c \) should be accessible from the class \( c_1 \).

Each action in a given state machine of a given class \( c \) should have a target (an event) in such a class \( c_1 \) that is pointed by the action. For example, for each call action or send action in the state machine \( sm \), a state machine \( sm_1 \) of the class \( c_1 \) should contain a transition, which is triggered by an event, which is a result of the action.

Moreover, the class \( c_1 \) should be accessible from the class \( c \).

In each case, the source and target may be, in particular, in the class \( c \).

3 Conclusions

The paper deals with the problem of consistency checking among the components of a UML model. Different kinds of consistency relations were identified, and presented informally. Exemplary definitions of consistency relations using OCL language [15] were presented. The OCL is used here in the same style as it is used in standard UML documents. OCL formulas defining consistency relations may be employed as a specification for consistency checking algorithms. An analysis of presented formulas has shown that checking algorithms have polynomial computational complexity.

We suggest that our approach to consistency checking may generalise approaches that are based on type-checking techniques, e.g. [5], [16]. Checking whether OCL formulas are satisfied in a context of a given model may be solving practically by means of programming tools.
The presented approach of diagram consistency checking applied to chosen diagrams may be extended to other kinds of UML diagrams.

References


