Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 127 (2005) 17–33
www.elsevier.com/locate/entcs

From UML Models to Graph Transformation Systems

Paul Ziemann[1] Karsten Hölscher[2] Martin Gogolla[3]

Department of Computer Science
University of Bremen
Bremen, Germany

Abstract

In this paper we present an approach that allows to validate properties of UML models. The approach is based on an integrated semantics for central parts of the UML. We formally cover UML use case, class, object, statechart, collaboration, and sequence diagrams. Additionally full OCL is supported in the common UML fashion. Our semantics is based on the translation of a UML model into a graph transformation system consisting of graph transformation rules and a working graph that represents the system state. By applying the rules on the working graph, the evolution of the modeled system is simulated.

Keywords: Graph transformation, UML semantics, validation, CASE tool

1 Introduction

Today the Unified Modeling Language (UML) is widely accepted as a standard for modeling object-oriented software systems. UML is a graphical language providing different diagram types for describing particular aspects of software artifacts. The syntax of these diagrams is defined by means of a metamodel in [12], notated as class diagrams. However this approach is semi-formal, since the class diagram itself is defined in a cyclic way by the metamodel.

[1] Email: ziemann@informatik.uni-bremen.de
[2] Email: hoelscher@informatik.uni-bremen.de
[3] Email: gogolla@informatik.uni-bremen.de

doi:10.1016/j.entcs.2004.10.025

Furthermore the semantics of UML diagrams is only expressed in natural language. The graphical notation is enhanced by the Object Constraint Language (OCL), which permits to formulate constraints in a textual way that cannot be expressed by the diagrams. OCL is again semi-formally defined in [12]. A formal syntax and semantics for UML class diagrams as well as OCL has been introduced in [13], which is also included in the accepted OCL 2.0 OMG submission [1].

In this paper we present an integrated formal semantics not only for class diagrams but for further basic diagram types: use case, object, statechart and interaction diagrams. We stick to UML 1.5 but UML 2.0 likewise includes the UML concepts covered by us, albeit some details and the naming have changed in some cases. In particular, collaboration diagrams are called communication diagrams in UML 2.0. The new integrated semantics is formalized employing the concepts of graph transformation, which is a well-developed field (cf. [15], [4], [5]). We are not aware of a formal approach handling this collection of UML diagrams, in particular the formal incorporation of use cases is new (in [17] use cases are described precisely by so-called operation schemas including OCL pre- and postconditions but the connection to other UML diagrams is left open).

Our approach provides a framework for an automatic translation of a UML model into a graph transformation system. The UML model may consist of the mentioned diagram types and can include OCL expressions. The graph transformation system comprises a set of graph transformation rules and a so-called working graph, hence called system state graph. As the name may suggest, the system state graph represents the current state of the modeled system. The graph transformation rules modify this state step by step, thus simulating a run through the modeled system.

In contrast to most work on graph transformation, we employ an enhanced approach, which allows OCL expressions in rules. We combine the advantages of two worlds: the operational graph transformation world and the logic-based OCL world. On the one hand graph transformations allow to handle complex issues by depicting and modifying them using more intuitive graphical representations. On the other hand, although it is theoretically possible to represent every aspect in the graphical structure, the additional power to use OCL as a textual notation leads to the benefit of even more compact graphs in most cases. In our approach OCL expressions navigating in the current system state are used as application conditions, which decide whether a certain rule may or may not be applied. Furthermore OCL is used in attribute expressions in the right-hand side of graph transformation rules. The modeler can also utilize OCL for querying the current state of the modeled system.