Chapter I

A Framework for Managing Consistency of Evolving UML Models

Tom Mens
Université de Mons-Hainaut, Belgium

Ragnhild Van Der Straeten
Vrije Universiteit Brussel, Belgium

Jocelyn Simmonds
Universidad de Chile, Chile

Abstract

As the standard for object-oriented analysis and design, the UML (Unified Modeling Language) metamodel, as well as contemporary CASE (Computer-Aided Software Engineering) tools, must provide adequate and integrated support for all essential aspects of software evolution. This includes version control, traceability, impact analysis, change propagation, inconsistency management, and model refactorings. This chapter focuses on the latter two aspects, and shows how tool support for these aspects can be provided. First, we extend the UML metamodel with support for versioning. Second, we make a classification of the possible inconsistencies of UML design models. Finally, we use the formalism of description logics, a decidable fragment of first-order predicate logic, to express logic rules that can detect and resolve
these inconsistencies. We also show how the logic rules are used to propose model refactorings. As a proof of concept, we report on the results of initial experiments with a prototype tool we developed for this approach.

A Framework for Managing Consistency of Evolving UML Models

Introduction

Any software system that is deployed in a real-world setting is subject to evolution (Lehman, Ramil, Wernick, Perry, & Turski, 1997). Because of this, it is crucial for any software development process to provide support for software evolution during all development phases. This includes support for version control (Conradi & Westfechtel, 1998), traceability management and change impact analysis (Bohner & Arnold, 1996), change propagation (Rajlich, 1997), inconsistency management (Finkelstein, Gabbay, Hunter, Kramer, & Nuseibeh, 1993; Grundy, Hosking, & Mugridge, 1998; Spanoudakis & Zisman, 2001), and software restructuring (Opdyke, 1992; Fowler, 1999).

Since design forms an essential part of the software development process, the activities mentioned above should not be restricted to source code, but should affect design models as well. Since UML (Unified Modeling Language) is the generally accepted object-oriented design standard, it ought to play an essential role in software evolution. Unfortunately, contemporary tools and environments for UML do not provide adequate and integrated support for many of the essential aspects of software evolution indicated above. The main underlying cause of all the problems mentioned is that the UML metamodel itself provides very poor support for software evolution. Hence, we do not only need to address the problem from a technical point of view (tool support), but from a conceptual point of view as well (metamodel support). If we have such an “evolution-aware” UML metamodel, we should be able to come up with an “evolution framework” that allows us to address the various evolution aspects in a uniform and integrated way. This is summarised in Figure 1, which illustrates the different kinds of evolution activities that an ideal UML tool should be able to perform.

Such tool support is crucial because of the inherent complexity of UML design models, which are typically expressed as a (large) collection of interdependent and partially overlapping UML diagrams. Different aspects of the software system are covered by different types of UML diagrams. Because of the wide variety of different types of UML diagrams, and the many relationships between them, managing all these diagrams is a very complex task. To make it even more complex, as the software evolves, those diagrams need to undergo changes to correct errors, accommodate new requirements, and so on. Any of those changes may lead to inconsistencies within or across UML diagrams, and may in turn require subsequent changes to other elements in the UML diagrams. An additional problem is that changes to the design may necessitate changes in the source code as well, and vice versa. All of this contributes to the complexity of the problem, making tool support crucial.
Related Content

Introducing Non-functional Requirements in UML
www.igi-global.com/chapter/introducing-non-functional-requirements-uml/30540?camid=4v1a

SDRule Markup Language: Towards Modeling and Interchanging Ontological Commitments for Semantic Decision Making
www.igi-global.com/chapter/sdrule-markup-language/35856?camid=4v1a

Agile Development of Rule Systems
www.igi-global.com/chapter/agile-development-rule-systems/35862?camid=4v1a

Native XML Programming: Make Your Tags Active
www.igi-global.com/chapter/native-xml-programming/27781?camid=4v1a