IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033-1117, USA ITB7419
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

Chapter XVII

Formal Approaches to Systems Analysis Using UML: An Overview

Jonathan Whittle
NASA Ames Research Center, USA

Formal methods, whereby a system is described and/or analyzed using precise mathematical techniques, is a well-established and yet, under-used approach for developing software systems. One of the reasons for this is that project deadlines often impose an unsatisfactory development strategy in which code is produced on an ad hoc basis without proper thought about the requirements and design of the piece of software in mind. The result is a large, often poorly documented and un-modular monolith of code that does not lend itself to formal analysis. Because of their complexity, formal methods work best when code is well structured, e.g., when they are applied at the modeling level. UML is a modeling language that is easily learned by system developers and, more importantly, an industry standard, which supports communication between the various project stakeholders. The increased popularity of UML provides a real opportunity for formal methods to be used on a daily basis within the software lifecycle. Unfortunately, the lack of precision of UML means that many formal techniques cannot be applied directly. If formal methods are to be given the place they deserve within UML, a more precise description of UML must be developed. This chapter surveys recent attempts to provide such a description, as well as techniques for analyzing UML models formally.

The Unified Modeling Language (UML) (Object Management Group, 1999; Booch, Jacobson, & Rumbaugh, 1998) provides a collection of standard notations for modeling almost any kind of computer artifact. UML supports a highly iterative, distributed software development process, in which each stage of the software lifecycle (e.g., requirements capture/analysis, initial and detailed design) can be specified using a combination of particular UML notations. The fact that UML is an industry standard promotes communication and understanding between different project stakeholders. When used within a commercial tool (e.g., Rhapsody [I-Logix Inc, 2001], Rational Rose [Rational Software Corporation, 2001]) that supports stub code generation from models, UML can alleviate many

This chapter appears in the book, Advanced Topics in Database Research by Keng Siau.

of the traditional problems with organizing a complex software development project. Although a powerful and flexible approach, there currently exist a number of gaps in the support provided by UML and commercial tools. First and foremost, the consistency checks provided by current tools are limited to very simple syntactic checks, such as consistency of naming across models. A greatly improved process would be obtained if tools were augmented with deeper semantic analyses of UML models. Unfortunately, although many of these techniques already exist, having been developed under the banner of Formal Methods, they cannot be applied directly to UML. UML is, in fact, grossly imprecise. There is as yet no standard formal semantics for any part of UML, and this makes the development of semantic tool support an onerous task.

This chapter gives an overview of current attempts to provide an additional degree of formality to UML and also of attempts to apply existing Formal Methods analyses to UML models. Space prevents the presentation of too much detail, so the description is at a more introductory level. Our starting point is the UML definition document itself (Object Management Group, 2000), which actually includes a section on UML semantics. Unfortunately, this semantics is by no means formal but provides merely a collection of rules or English text describing a subset of the necessary semantics.

UML is mainly a diagrammatic modeling language, consisting of various graphical notations for modeling all aspects of an object-oriented system and its design. In addition, UML contains a textual language, Object Constraint Language (OCL) that can be used to specify additional constraints on a UML model. A UML *class diagram* is a notation for modeling the static structure of a system. It describes the classes in a system and the relationships between them.

Figure 1 shows an example of a class diagram for part of an Automated Teller Machine (ATM) example. In object-oriented fashion, the main class (here "ATM toplevel") is broken down into sub-classes. The aggregation relation (—◇) shows when one class is *part of* another one. If the diamond shape is filled, the relation is a composition, which means that the *part of* relationship is strict, in the sense that the part cannot exist without the whole. The generalization relation (—▷) shows when one class is *an instance of* another. Classes can also be related by simple associations, which are drawn as lines between classes. For further details, see Object Management Group (2000). *Statecharts*, introduced originally by David

Figure 1: A UML class diagram