Preface

This book provides a set of readings on the Unified Modeling Language (UML), currently the most popular language for modeling object-oriented software. To set the scene for these readings, this preface provides a brief historical and structural overview of UML, and then identifies the specific focus of each of the contributing chapters.

Initially based on a combination of the Booch, OMT (Object Modeling Technique) and OOSE (Object-Oriented Software Engineering) methods, UML was refined and extended by a consortium of several companies, and is subject to ongoing revisions by the Object Management Group (OMG). UML was first adopted in November 1997 by the OMG as a language for object-oriented analysis and design (UML version 1.1). Within the OMG, the UML specification is the responsibility of the Analysis and Design Task Force (ADTF). Minor changes to the UML specification that led to point releases (e.g., 1.2, 1.3, 1.4) are managed by a subgroup of the ADTF known as a UML Revision Task Force (RTF).

In late 1999, UML version 1.3 was approved and work began on version 1.4, which was expected to be ratified around the end of 2000. Following UML’s adoption by the OMG, the language has gained wide acceptance in industry for object-oriented modeling. Some preliminary work has begun on a major revision (2.0), for release some years later (possibly 2002). Though not yet an official standard, UML has been proposed for standardization by the International Standards Organization (ISO), and approval is anticipated sometime in 2001. The UML specification itself, as well as details about current work on its revisions, can be accessed online at www.omg.org/technology/uml/.

From a semiotic viewpoint, UML can be examined in relation to its syntax (notation), semantics (meaning) and pragmatics (use). The UML notation includes hundreds of symbols, from which various diagrams may be constructed to model different perspectives of an application. Different kinds of diagrams provide different views of the overall model. The UML specification includes the following canonical diagram types:

- Use case diagram
- Class diagram
- Behavior diagrams
  - Statechart diagram
  - Activity diagram
  - Interaction diagrams
    - Sequence diagram
    - Collaboration diagram
- Implementation diagrams
  - Component diagram
  - Deployment diagram

Use case diagrams are used primarily for requirements analysis to provide a high level view of how actors interact with the system. Class diagrams are used to capture the static aspects (or data model). Behavior diagrams are used to model the dynamic aspects of the system, while implementation diagrams indicate how components are packaged and de-
ployed. Although the syntax and semantics of these diagrams are described in the UML specification, very little is said about pragmatics, or how a modeler may use these notations to construct models. The only advice given as to the modeling process is that it should be “use-case driven, architecture centric and iterative.” Various companies have made their own proposals for a detailed modeling process, and the OMG itself has begun preliminary work on a “Unified Process” to provide advice on the use of UML in developing software.

In spite of its good points, UML is a large and complex language, with many features in need of refinement or clarification, and there are different views about how to use UML to develop software systems. This book sheds light on such issues, by illustrating how UML can be used successfully in practice as well as identifying some problematic aspects of UML and suggesting possible solutions. As an edited collection of insightful contributions from both industry and academia, the book should be of interest to researchers, practitioners and instructors of UML.

The book is divided into the following four sections: Applying UML, Evaluating UML, Extending UML and Formalizing UML. Although each individual chapter appears in the section most closely related to its primary focus, a chapter may include some material relevant to another section. For example, most chapters include a brief critical evaluation of some aspects of UML.

Applying UML

In “Systematic Design of Web Applications with UML,” Rolf Hennicker and Nora Koch propose a systematic design method for Web applications that addresses both navigation and presentation aspects, based on a UML profile for the Web domain. In “A Systematic Approach to Transform UML Static Models to Object-Oriented Code,” Lilian Favre and Silvia Clérici discuss a reuse-based method for mapping UML static models to object-oriented code, using Eiffel as the sample target language. In “Data Modeling and UML,” Devang Shah and Sandra Slaughter discuss how to map a data model expressed as a UML class diagram to a relational database schema, illustrating their technique with a drug dispensing application.

Evaluating UML

Extending UML


Formalizing UML

In “Linking UML with Integrated Formal Techniques,” Jing Liu, Jin Song Dong and Brendan Mahony investigate the links between static and dynamic aspects of UML and the Timed Communicating Object Z formalism, illustrating the connections with an example light control application. In “Seamless Formalizing the UML Semantics Through Metamodels,” José Luis Fernández Alemán and Ambrosio Toval Álvarez use the algebraic specification language, Maude, to provide a framework for formalizing UML that caters for evolution of the UML metamodel. In “An Interactive Viewpoint on the Role of UML,” Dina Goldin, David Keil and Peter Wegner provide a theoretical framework for modeling interactive computing, and discuss its use for strengthening the formal foundations of UML.

The editors:

Keng Siau (University of Nebraska, USA)
Terry Halpin (Microsoft Corporation, USA)

August 2000