Software Modeling and Design

This book provides all you need to know for modeling and design of software applications, from use cases to software architectures in UML. It shows you how to apply the COMET UML-based modeling and design method to real-world problems. The author describes architectural patterns for various architectures, such as broker, discovery, and transaction patterns for service-oriented architectures, and layered patterns for software product line architectures, and addresses software quality attributes, including maintainability, modifiability, testability, traceability, scalability, reusability, performance, availability, and security.

Complete case studies illustrate design issues for different software architectures: a banking system for client/server architectures, an online shopping system for service-oriented architectures, an emergency monitoring system for component-based software architectures, and an automated guided vehicle system for real-time software architectures.

Organized as an introduction followed by several self-contained chapters, the book is perfect for senior undergraduate or graduate courses in software engineering and for experienced software engineers who want a quick reference at each stage of the analysis, design, and development of large-scale software systems.

Hassan Gomaa is Professor of Computer Science and Software Engineering at George Mason University. Gomaa has more than thirty years' experience in software engineering, in both industry and academia. He has published more than 170 technical papers and is the author of three books: *Designing Software Product Lines with UML; Designing Concurrent, Distributed, and Real-Time Applications with UML*; and *Software Design Methods for Concurrent and Real-Time Systems*.

SOFTWARE MODELING AND DESIGN

UML, Use Cases, Patterns, and Software Architectures

Hassan Gomaa
George Mason University, Fairfax, Virginia

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town,
Singapore, São Paulo, Delhi, Tokyo, Mexico City

Cambridge University Press
32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org
Information on this title: www.cambridge.org/9780521764148

First published 2011

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication data

Gomaa, Hassan.
Software modeling and design : UML, use cases, patterns, and software architectures /
Hassan Gomaa.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-521-76414-8 (hardback)
1. Computer software – Development. 2. Software architecture. 3. Computer simulation. I. Title.
QA76.76.D47G6522 2011
003′.3–dc22 2010049584

ISBN 978-0-521-76414-8 Hardback

To Gill, William and Neela, Alex,

Amanda and Neil, and Edward

Contents

Preface

OVERVIEW

This book describes a use case–driven UML-based method for the modeling and design of software architectures, including object-oriented software architectures, client/server software architectures, service-oriented architectures, component-based software architectures, concurrent and real-time software architectures, and software product line architectures. The book provides a unified approach to designing software architectures and describes the special considerations for each category of software architecture. In addition, there are four case studies, a client/server banking system, a service-oriented architecture for an online shopping system, a distributed component-based emergency monitoring system, and a real-time automated guided vehicle system.

This book describes a UML-based software modeling and design method called COMET (Collaborative Object Modeling and Architectural Design Method). COMET is a highly iterative object-oriented software development method that addresses the requirements, analysis, and design modeling phases of the object-oriented development life cycle.

The book is intended to appeal to readers who wish to design software architectures using a systematic UML-based method that starts from requirements modeling with use cases, through static and dynamic modeling, to software design based on architectural design patterns.

WHAT THIS BOOK PROVIDES

Various textbooks on the market describe object-oriented analysis and design concepts and methods. This book addresses the specific needs of designing software architectures. It addresses UML-based design of software architectures, starting with use cases for requirements modeling, static modeling with class diagrams, and dynamic modeling with object interaction analysis and state machine modeling, through software design with architectural design patterns. All examples are

described using the UML 2 notation, the latest version of the standard. In particular, this book:

- Provides a comprehensive treatment of the application of the UML-based object-oriented concepts to requirements modeling, analysis modeling, and design modeling. Requirements modeling addresses use case modeling to describe functional requirements with extensions to describe nonfunctional requirements. Analysis modeling addresses static modeling and dynamic modeling (both interaction and state machine modeling). Design modeling addresses important architectural issues, including a systematic approach for integrating use case–based interaction diagrams into an initial software architecture and applying architectural and design patterns for designing software architectures.
- Provides a common approach for requirements and analysis modeling and then addresses specific design issues (in a separate chapter for each category of software architecture) for designing the software architecture for object-oriented software systems, client/server systems, service-oriented systems, component-based systems, real-time systems, and software product lines.
- Describes how software architectures are designed by first considering software architectural patterns relevant for that category of software architecture, such as client/service patterns for client/server and component-based software architecture; brokering, discovery, and transaction patterns for service-oriented architectures; real-time control patterns for real-time software architecture; and layered patterns for software product line architectures.
- Describes software quality attributes, which can have a profound effect on the quality of a software product. Many of these attributes can be addressed and evaluated at the time the software architecture is developed. The software quality attributes covered include maintainability, modifiability, testability, traceability, scalability, reusability, performance, availability, and security.
- Presents four detailed case studies. Case studies are presented by software architecture area, including a banking system for client/server architectures, an online shopping system for service-oriented architecture, an emergency monitoring system for component-based software architecture, and an automated guided vehicle system for the real-time software architecture.
- Appendices include a glossary, a bibliography, and a catalog of architectural design patterns. There is also be an appendix on teaching considerations for teaching academic and industrial courses based on this book. Exercises follow most chapters.

INTENDED AUDIENCE

This book is intended for both academic and professional audiences. The academic audience includes senior undergraduate- and graduate-level students in computer science and software engineering, as well as researchers in the field. The professional audience includes analysts, software architects, software designers, programmers, project leaders, technical managers, program managers, and quality-assurance specialists who are involved in the analysis, design, and development of large-scale software systems in industry and government.

WAYS TO READ THIS BOOK

This book may be read in various ways. It can be read in the order in which it is presented, in which case Chapters 1 through 4 provide introductory concepts; Chapter 5 provides an overview of the COMET/UML software modeling and design method; Chapters 6 through 20 provide an in-depth treatment of software modeling and design; and Chapters 21 through 24 provide detailed case studies.

Alternatively, some readers may wish to skip some chapters, depending on their level of familiarity with the topics discussed. Chapters 1 through 4 are introductory and may be skipped by experienced readers. Readers familiar with software design concepts may skip Chapter 4. Readers particularly interested in software modeling and design can proceed directly to the description of COMET/UML, starting in Chapter 5. Readers who are not familiar with UML, or who are interested in finding out about the changes introduced by UML 2, can read Chapter 2 in conjunction with Chapters 5 through 20.

Experienced software designers may also use this book as a reference, referring to various chapters as their projects reach a particular stage of the requirements, analysis, or design process. Each chapter is relatively self-contained. For example, at different times one might refer to Chapter 6 for a description of use cases, to Chapter 7 for a discussion of static modeling, and to Chapter 9 for a description of dynamic interaction modeling. Chapter 10 can be referenced for designing state machines; Chapter 12 and Appendix A for software architectural patterns; Chapter 14 for object-oriented software architectures; and Chapter 15 for designing a relational database from a static model. Chapter 16 can be consulted for service-oriented architectures; Chapter 17 for distributed component-based software design; Chapter 18 for real-time design; and Chapter 19 for software product line design. One can also improve one's understanding of how to use the COMET/UML method by reading the case studies, because each case study explains the decisions made at each step of the requirements, analysis, and design modeling processes in the design of a real-world application.

Hassan Gomaa
George Mason University
December 2010
Email: hgomaa@gmu.edu
Web: http://mason.gmu.edu/~hgomaa

Annotated Table of Contents

PART I: OVERVIEW

Chapter 1: Introduction

This chapter presents an introduction to software modeling and design, a discussion of software design issues, an introduction to software architecture, and an overview of object-oriented analysis and design with UML.

Chapter 2: Overview of the UML Notation

This chapter presents an introduction to the UML notation, including use case diagrams, class diagrams, interaction diagrams, statechart diagrams, packages, concurrent communication diagrams, and deployment diagrams. The chapter also covers UML extension mechanisms and the evolution of UML into a standard.

Chapter 3: Software Life Cycle Models and Processes

This chapter introduces the software life cycles used for developing software, including the waterfall, prototyping, iterative, spiral, and unified process. It compares and contrasts them.

Chapter 4: Software Design and Architecture Concepts

This chapter discusses and presents an overview of key software design concepts, including object-oriented design concepts of classes, objects, information hiding and inheritance, and concurrent processing with concurrent objects. An introduction is given to software architecture and components, software design patterns, and software quality attributes.

Chapter 5: Overview of Software Modeling and Design Method

This chapter provides an overview of the software modeling and design method, including requirements modeling, analysis modeling, and design modeling. An overview of the different kinds of software architectures addressed in this textbook is given.

PART II: SOFTWARE MODELING

Chapter 6: Use Case Modeling

This chapter starts with an overview of requirements analysis and specification. It then goes on to describe the use case modeling approach to developing requirements. This is followed by an approach for developing use cases. The chapter covers use cases, actors, identifying use cases, documenting use cases, and use case relationships. An introduction is given to activity diagrams for precise modeling of individual use cases. Use cases are extended to document nonfunctional requirements.

Chapter 7: Static Modeling

This chapter describes static modeling concepts, including associations, whole/part relationships (composition and aggregation), and generalization/specialization relationships. Special topics include modeling the boundary of the system and modeling entity classes, which are information-intensive classes.

Chapter 8: Object and Class Structuring

This chapter describes the categorization of application classes, or the role the class plays in the application. The major categories covered are boundary objects, entity objects, control objects, and application logic objects. This chapter also describes the corresponding behavior pattern for each category of object.

Chapter 9: Dynamic Interaction Modeling

This chapter describes dynamic interaction modeling concepts. Interaction (sequence or communication) diagrams are developed for each use case, including the main scenario and alternative scenarios. It also describes how to develop an interaction model starting from the use case.

Chapter 10: Finite State Machines

This chapter describes finite state machine modeling concepts. In particular, a state-dependent control class needs to be modeled with a finite state machine and depicted as a statechart. This chapter covers events, states, conditions, actions, entry and exit actions, composite states, and sequential and orthogonal states.

Chapter 11: State-Dependent Dynamic Interaction Modeling

This chapter describes dynamic interaction modeling for state-dependent object interactions. It describes how state machines and interaction diagrams relate to each other and how to make them consistent with each other.

PART III: ARCHITECTURAL DESIGN

Chapter 12: Overview of Software Architectures

This chapter introduces software architecture concepts. Multiple views of a software architecture and an overview of software architectural patterns (architectural structure and communication patterns) are presented. A template for software architectural patterns is provided, and interface design is introduced and discussed.

Chapter 13: Software Subsystem Architectural Design

This chapter presents issues in software architectural design, including the transition from analysis to architectural design, separation of concerns in subsystem design, subsystem structuring criteria, and the design of subsystem message communication interfaces.

Chapter 14: Designing Object-Oriented Software Architectures

This chapter describes object-oriented design of sequential software architectures, particularly design using the concepts of information hiding, classes, and inheritance. In class interface design, the designer of the class needs to decide what information should be hidden and what information should be revealed in the class interface, which consists of the operations provided by the class. This chapter also discusses design by contract and sequential class design, which includes the design of data abstraction classes, state machine classes, graphical user interface classes, and business logic classes. Detailed design of classes is also considered.

Chapter 15: Designing Client/Server Software Architectures

The design of clients and servers is described in this chapter. It also includes a discussion of client/service patterns (structural and behavioral), sequential and concurrent services, and mapping a static model to a relational database, which includes the design of database wrappers and logical relational database design.

Chapter 16: Designing Service-Oriented Architectures

This chapter describes the characteristics of service-oriented architectures. It discusses Web services and service patterns, including registration, brokering, and discovery patterns. It then describes transaction patterns and transaction design, including atomic transactions, two-phase commit protocol, compound transactions,

and long-living transactions. This chapter also presents information on how to design services for reuse, how to build applications that reuse services, and service coordination.

Chapter 17: Designing Component-Based Software Architectures

This chapter describes distributed component-based software architectural design. The design of component interfaces (provided and required) is described. The chapter also discusses how component-based software architectures can be depicted with the structured class and composite structure diagram notation introduced in UML 2, which allows components, ports, connectors, and provided and required interfaces to be depicted.

Chapter 18: Designing Concurrent and Real-Time Software Architectures

This chapter considers the characteristics of embedded real-time systems. It discusses concurrency and control; control patterns for real-time systems; concurrent task structuring, including event-driven tasks, periodic tasks, and demand-driven tasks; and design of task interfaces, including message communication, event synchronization, and communication through passive objects.

Chapter 19: Designing Software Product Line Architectures

This chapter presents characteristics of software product lines – modeling commonality and variability for a family of systems. Also discussed are feature modeling, variability modeling, software product line architectures, and application engineering. Variability modeling in use cases, static and dynamic models, and software architectures is also considered.

Chapter 20: Software Quality Attributes

This chapter describes software quality attributes and how they are used to evaluate the quality of the software architecture. Software quality attributes include maintainability, modifiability, traceability, usability, reusability, testability, performance, and security. The chapter also presents a discussion of how the architectural design method supports the software quality attributes.

PART IV: CASE STUDIES

Each case study provides a detailed description of how to apply the concepts and methods described so far to the design of different kinds of software architecture: client/server software architecture, service-oriented architecture, component-based software architecture, and real-time software architecture. In each case study, the rationale for the modeling and design decisions is discussed.

Chapter 21: Client/Server Software Architecture Case Study: Banking System

This chapter describes how the software modeling and design method is applied to the design of a client/server system that consists of a bank server and several ATM clients. The design of the ATM client is also an example of concurrent software design. The design of the banking service is an example of a sequential object-oriented design.

Chapter 22: Service-Oriented Architecture Case Study: Online Shopping System

This chapter describes how the software modeling and design method is applied to the design of a service-oriented architecture for an online shopping system, which consists of multiple services invoked by multiple clients and needs brokering, discovery, and service coordination.

Chapter 23: Component-Based Software Architecture Case Study: Emergency Monitoring System

This chapter describes how the software modeling and design method is applied to the design of a component-based software architecture, an emergency monitoring system, in which software components can be assigned to the hardware configuration at deployment time.

Chapter 24: Real-Time Software Architecture Case Study: Automated Guided Vehicle System

This chapter describes how the software modeling and design method is applied to the design of a real-time automated guided vehicle system (consisting of several concurrent tasks), which is part of a factory automation system of systems.

Appendix A: Catalog of Software Architectural Patterns

The software architectural structure, communication, and transaction patterns used in this textbook are documented alphabetically in a common template for easy reference.

Appendix B: Teaching Considerations

This appendix describes approaches for teaching academic and industrial courses based on this textbook.

Acknowledgments

I gratefully acknowledge the reviewers of earlier drafts of the manuscript for their constructive comments, including Rob Pettit, Kevin Mills, Bran Selic, and the anonymous reviewers. I am very grateful to the students in my software design and reusable software architecture courses at George Mason University for their enthusiasm, dedication, and valuable feedback. Many thanks are also due to Koji Hashimoto, Erika Olimpiew, Mohammad Abu-Matar, Upsorn Praphamontripong, and Sylvia Henshaw for their hard work and careful attention producing the figures. I am also very grateful to the Cambridge University Press editorial and production staff, including Heather Bergman, Lauren Cowles, David Jou, Diane Lamsback, and the production staff at Aptara, without whom this book would not have seen the light of day.

I gratefully acknowledge the permission given to me by Pearson Education, Inc., to use material from my earlier textbooks, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, © 2000 Hassan Gomaa, Reproduced by permission of Pearson Education, Inc., and *Designing Software Product Lines with UML*, © 2005 Hassan Gomaa, Reproduced by permission of Pearson Education, Inc.

Last, but not least, I would like to thank my wife, Gill, for her encouragement, understanding, and support.