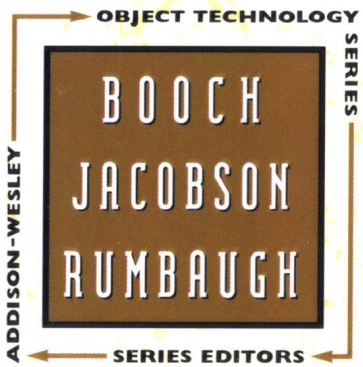
UML与面向对象设计影印丛书

UML与并行分布式实时应用程序设计

DESIGNING CONCURRENT, DISTRIBUTED, AND REAL-TIME APPLICATIONS WITH UML

(美) HASSAN GOMAA 编著

Foreword by Peter Freeman and Bran Selic



科学出版社
www.sciencep.com

UML 与面向对象设计影印丛书

UML 与并行分布式实时应用程序设计

Designing Concurrent, Distributed, and Real-Time Applications with UML

（美） Hassan Gomaa 编著

科学出版社
北京

图字：01-2003-7658 号

内 容 简 介

本书对 UML 在并行分布式实时系统开发中的应用作了详细、全面的介绍，尤其对面向对象方法解决此类系统特有的问题作了有针对性的讲解。主要内容包括 OO 软件工程的生命周期、用案建模技术在嵌入式系统中的应用、静态和动态分析建模、对象和类的构建、有限状态机和状态表、分布式对象技术、并行系统的架构设计，等等。

本书适用于并行分布式实时系统开发人员和自学者。

图书在版编目(CIP)数据

UML 与并行分布式实时应用程序设计=Designing Concurrent,Distributed,and Real-Time Applications with UML/（美）Hassan Gomaa 编著. —影印本.—北京：科学出版社，2004
（UML 与面向对象设计影印丛书）
ISBN 7-03-012492-8

Ⅰ.U… Ⅱ.G… Ⅲ.面向对象语言，UML—程序设计—英文 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 103055 号

策划编辑：李佩乾/责任编辑：舒 立
责任印制：吕春珉/封面制作：东方人华平面设计室

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com
双青印刷厂 印刷
科学出版社发行 各地新华书店经销
*
2004 年 1 月第 一 版 开本：787×960 1/16
2004 年 1 月第 一次印刷 印张：50 3/4
印数：1—3 000 字数：972 000

定价：82.00 元

(如有印装质量问题,我社负责调换<环伟>)

影印前言

随着计算机硬件性能的迅速提高和价格的持续下降，其应用范围也在不断扩大。交给计算机解决的问题也越来越难，越来越复杂。这就使得计算机软件变得越来越复杂和庞大。20 世纪 60 年代的软件危机使人们清醒地认识到按照工程化的方法组织软件开发的必要性。于是软件开发方法从 60 年代毫无工程性可言的手工作坊式开发，过渡到 70 年代结构化的分析设计方法、80 年代初的实体关系开发方法，直到面向对象的开发方法。

面向对象的软件开发方法是在结构化开发范型和实体关系开发范型的基础上发展而来的，它运用分类、封装、继承、消息等人类自然的思维机制，允许软件开发者处理更为复杂的问题域和其支持技术，在很大程度上缓解了软件危机。面向对象技术发端于程序设计语言，以后又向软件开发的早期阶段延伸，形成了面向对象的分析和设计。

20 世纪 80 年代末 90 年代初，先后出现了几十种面向对象的分析设计方法。其中，Booch, Coad/Yourdon、OMT 和 Jacobson 等方法得到了面向对象软件开发界的广泛认可。各种方法对许多面向对象的概念的理解不尽相同，即便概念相同，各自技术上的表示法也不同。通过 90 年代不同方法流派之间的争论，人们逐渐认识到不同的方法既有其容易解决的问题，又有其不容易解决的问题，彼此之间需要进行融合和借鉴；并且各种方法的表示也有很大的差异，不利于进一步的交流与协作。在这种情况下，统一建模语言(UML)于 90 年代中期应运而生。

UML 的产生离不开三位面向对象的方法论专家 G. Booch、J. Rumbaugh 和 I. Jacobson 的通力合作。他们从多种方法中吸收了大量有用的建模概念，使 UML 的概念和表示法在规模上超过了以往任何一种方法，并且提供了允许用户对语言做进一步扩展的机制。UML 使不同厂商开发的系统模型能够基于共同的概念，使用相同的表示法，呈现彼此一致的模型风格。1997 年 11 月 UML 被 OMG 组织正式采纳为标准的建模语言，并在随后的几年中迅速地发展为事实上的建模语言国际标准。

UML 在语法和语义的定义方面也做了大量的工作。以往各种关于面向对象方法的著作通常是以比较简单的方式定义其建模概念，而以主要篇幅给出过程指导，论述如何运用这些概念来进行开发。UML 则以一种建模语言的姿态出现，使用语言学中的一些技术来定义。尽管真正从语言学的角度看它还有许多缺陷，但它在这方面所做的努力却是以往的各种建模方法无法比拟的。

从 UML 的早期版本开始，便受到了计算机产业界的重视，OMG 的采纳和大公司的支持把它推上了实际上的工业标准的地位，使它拥有越来越多的用户。它被广泛地用

于应用领域和多种类型的系统建模，如管理信息系统、通信与控制系统、嵌入式实时系统、分布式系统、系统软件等。近几年还被运用于软件再工程、质量管理、过程管理、配置管理等方面。而且它的应用不仅仅限于计算机软件，还可用于非软件系统，例如硬件设计、业务处理流程、企业或事业单位的结构与行为建模，等等。

在 UML 陆续发布的几个版本中，逐步修正了前一个版本中的缺陷和错误。即将发布的 UML2.0 版本将是对 UML 的又一次重大的改进。将来的 UML 将向着语言家族化、可执行化、精确化等理念迈进，为软件产业的工程化提供更有力的支撑。

本丛书收录了与面向对象技术和 UML 有关的十几本书，反映了面向对象技术最新的发展趋势以及 UML 的新的研究动态。其中涉及对面向对象建模理论研究与实践的有这样几本书：《面向对象系统架构及设计》主要讨论了面向对象的基本概念、静态设计、永久对象、动态设计、设计模式以及体系结构等近几年来面向对象技术领域中的新的理论知识与方法；《用 UML 进行用况对象建模》主要介绍了面向对象的需求阶段、分析阶段、设计阶段中用况模型的建立方法与技术；《高级用况建模》介绍了在建立用况模型中需要注意的高级的问题与技术；《UML 面向对象设计基础》则侧重于经典的面向对象理论知识的阐述；《UML 参考手册》列出了 UML 的所有术语和标准元素，从语义、表示法和用途等方面详尽地介绍了 UML 的构成和概念。

涉及 UML 在特定领域的运用的有这样几本：《UML 实时系统开发》讨论了进行实时系统开发时需要对 UML 进行扩展的技术；《用 UML 构建 Web 应用程序》讨论了运用 UML 进行 Web 应用建模所应该注意的技术与方法；《面向对象系统测试：模型、视图与工具》介绍了将 UML 应用于面向对象的测试领域所应掌握的方法与工具；《对象、构件、框架与 UML 应用》讨论了如何运用 UML 对面向对象的新技术——构件-框架技术建模的方法策略；《UML 与 Visual Basic 应用程序开发》主要讨论了从 UML 模型到 Visual Basic 程序的建模与映射方法；《XML 程序的 UML 建模》讲解了如何将 XML 与 UML 结合，创建动态的 Web 应用程序，实现最优的 B2B 应用集成；《构建可扩展数据库应用程序》介绍了商务模式和数据库模式的建模方法以及集成系统的程序实现；《UML 与并行分布式实时应用程序设计》对 UML 在并行分布式实时系统开发中的应用作了全面而详细的介绍，尤其对面向对象方法解决此类系统特有的问题作了有针对性的讲解；《UML 与 J2EE 企业应用程序开发》系统介绍了使用 J2EE 开发企业级软件工程时，将 UML 建模技术应用到软件开发各个阶段的方法。

介绍面向对象编程技术的有两本书：《COM 高手心经》和《ATL 技术内幕》，深入探讨了面向对象的编程新技术——COM 和 ATL 技术的使用技巧与技术内幕。

还有一本《Executable UML 技术内幕》，这本书介绍了可执行 UML 的理念与其支持技术，使得模型的验证与模拟以及代码的自动生成成为可能，也代表着将来软件开发的一种新的模式。

总之，这套书所涉及的内容包含了对软件生命周期的全过程建模的方法与技术，同时也对近年来的热点领域建模技术、新型编程技术作了深入的介绍，有些内容已经涉及到了前沿领域。可以说，每一本都很经典。

有鉴于此，特向软件领域中不同程度的读者推荐这套书，供大家阅读、学习和研究。

北京大学计算机系　蒋严冰 博士

Foreword by Peter Freeman

The recent and rapid advances in hardware and communications have led to an explosion of concurrent, real-time, and distributed applications. This, in turn, is changing forever the nature of the demands on practical software development. The widespread advent of object-oriented approaches and now the use of UML are changing practice, but as usual, at a pace that lags behind the needs.

One reason for this lag has been the absence of good, authoritative, practical guides to the object-oriented analysis and design of concurrent applications, especially those that are distributed and/or real-time. This book goes a long way toward fulfilling that need.

I cannot think of a better person to write a definitive text on this topic than Hassan Gomaa. For more than 20 years, Hassan has contributed to a deeper understanding of concurrent, distributed, and real-time applications through his work in industry as a designer, his research into new real-time design methods, and his teaching as a university professor. This book shows the results of his experience.

It is superbly organized and illustrated, as only an experienced teacher could have done. It shows the depth of understanding of the technology that comes from long and deep research focused on the subject matter. It has the illustrations and practical knowledge that come from long and direct contact with practical software design.

I hope that you enjoy this book as much as I did, and that you will be able to use it for many years to come.

Peter A. Freeman
John P. Imlay, Jr., Dean and Professor
Georgia Institute of Technology
May 2000

Foreword by Bran Selic

Errors using inadequate data are much less than those using no data at all.
—Charles Babbage

A recent search of a popular bookseller's Web site revealed a total of 1,188 titles (and growing daily) that are classified as dealing with "software engineering." Despite this apparent glut, there is precious little engineering in most of them and, correspondingly, little or no engineering in much of the software being developed these days. This book aims to change that.

Traditional engineering disciplines invariably involved the use of science and mathematics to ensure that a design would meet its objectives at an acceptable cost. Thus, one could proceed with high confidence to construct a bridge based on predictions made from a mathematical model of the design. In the case of software, however, design is primarily an informal process too often devoid of formal predictive models or techniques. From this perspective, it is highly instructive to contrast how software and hardware have evolved over the last several decades. Whereas hardware has become smaller, faster, cheaper, and more reliable, software has become larger, slower, more expensive, and less reliable over the same period. Significantly, modern hardware design relies heavily on constructing predictive models.

The absence of engineering fundamentals from software practice can be attributed in part to the fickle, almost chaotic, nature of software, which makes it notoriously difficult to model mathematically. Despite this inherent difficulty, however, a number of very useful analytical techniques have been developed. In particular, such techniques have evolved in the embedded real-time domain, where it is often critical to predict the temporal properties of software with a high

degree of certainty because human lives may depend on it. Yet, regardless of their proven effectiveness, these methods are not used very often. In fact, many real-time developers are not even aware of their existence.

The issue here is one of culture, or, more appropriately, the lack of one. Writing software is primarily an intellectual exercise, unhampered by physical limitations such as the need to cut and shape material or to expend large amounts of energy. Seduced by this apparent lack of resistance and the common perception that when all is said and done, only the code matters, far too many practitioners still equate software design with the process of writing software. Strangely enough, the same individuals have no trouble understanding the difference between designing a jumbo jet and assembling it.

Another incidental hurdle to the introduction of these techniques into software practice is that for historical reasons, some of them are defined in the context of the traditional procedural programming model. Although the techniques are not fundamentally dependent on that model, there remains the problem of mapping them to the newer object-oriented programming model for those who want to exploit the many advantages of that paradigm.

Hassan Gomaa's book is the first one I have seen that addresses these issues in a systematic and comprehensive manner. Much more than a mere compilation of unconnected "patterns" and point techniques, it explains clearly and in detail a way of reconciling specific traditional engineering techniques with the industry-standard Unified Modeling Language. Furthermore, it shows how such techniques fit into a fully defined development process, one that is specifically oriented toward developing concurrent, distributed, real-time systems. (Experienced software developers recognize fundamentally hard problems behind each of these terms individually—systems that combine all three typically belong to the category of the most challenging engineering problems.)

Based on the well-known dictum that we learn best by doing, fully worked-out, non-trivial examples take up a major portion of this book. The reader will benefit greatly from working through one or more of these examples to gain an intuitive feel for the approach, and, on a higher plane, for what software engineering should ultimately look like.

Bran Selic
May 2000

Preface

The UML Notation and Software Design Methods

This book describes the object-oriented analysis and design of concurrent applications, in particular distributed and real-time applications. Object-oriented concepts are crucial in software analysis and design because they address fundamental issues of adaptation and evolution. With the proliferation of notations and methods for the object-oriented analysis and design of software systems, the Unified Modeling Language (UML) has emerged to provide a standardized notation for describing object-oriented models. However, for the UML notation to be effectively applied, it needs to be used in conjunction with an object-oriented analysis and design method.

Most books on object-oriented analysis and design only address the design of sequential systems or omit the important design issues that need to be addressed when designing distributed and real-time applications. Blending object-oriented concepts with the concepts of concurrent processing is essential to the successful designing of these applications. Because the UML is now the standardized notation for describing object-oriented models, this book uses the UML notation throughout.

The COMET Concurrent Object Modeling and Architectural Design Method

COMET is a Concurrent Object Modeling and Architectural Design Method for the development of concurrent applications—in particular, distributed and real-time applications. The COMET Object-Oriented Software Life Cycle is a highly

iterative software life cycle, based around the use case concept. The Requirements Modeling phase views the system as a black box. A use case model is developed, which defines the functional requirements of the system in terms of actors and use cases.

In the Analysis Modeling phase, static and dynamic models of the system are developed. The static model defines the structural relationships among problem domain classes. Object structuring criteria are used to determine the objects to be considered for the analysis model. A dynamic model is then developed, in which the use cases from the requirements model are refined to show the objects that participate in each use case and their interactions with each other. In the dynamic model, state-dependent objects are defined by using statecharts.

In the Design Modeling phase, the software architecture of the system is designed, in which the analysis model is mapped to an operational environment. The analysis model, with its emphasis on the problem domain, is mapped to the design model, with its emphasis on the solution domain. Subsystem structuring criteria are provided to structure the system into subsystems. For distributed applications, the emphasis is on the division of responsibility between clients and servers, including issues concerning the centralization versus distribution of data and control. In addition, the design of message communication interfaces is considered, including synchronous, asynchronous, brokered, and group communication. Each subsystem is then designed. For the design of concurrent applications, including real-time applications, the emphasis is on object-oriented and concurrent tasking concepts. Task communication and synchronization interfaces are designed. The performance of the real-time design is analyzed by using the Software Engineering Institute's rate monotonic analysis approach.

What This Book Provides

Several textbooks on the market describe object-oriented concepts and methods, intended for all kinds of applications. However, distributed and real-time applications have special needs, which are treated only superficially in most of these books. This book provides a comprehensive treatment of the application of fundamental object-oriented concepts to the analysis and design of distributed (including client/server) and real-time applications. In addition to the object-oriented concepts of information hiding, classes, and inheritance, this book also describes the concepts of finite state machines, concurrent tasks, distributed object technology, and real-time scheduling. It then describes in considerable detail the COMET

method, which is a UML based object-oriented analysis and design method for concurrent, distributed and real-time applications. To show how COMET is applied in practice, this book also describes several comprehensive case studies, presented by application area: real-time software design, client/server software design, and distributed application design.

The following are distinguishing features of this book:

- Emphasis on structuring criteria to assist the designer at various stages of the analysis and design process: subsystems, objects, and concurrent tasks
- Emphasis on dynamic modeling, in the form of both object interaction modeling and finite state machine modeling, describing in detail how object collaborations and statecharts work together
- Emphasis on concurrency, describing the characteristics of active and passive objects
- Emphasis on distributed application design and the ways in which distributed components can communicate with each other
- Emphasis on performance analysis of real-time designs, using real-time scheduling
- Comprehensive case studies of various applications to illustrate in detail the application of concepts and methods

Organization of Book

The book is divided into three parts. Part I of the book provides a broad overview by describing concepts, technology, life cycles and methods for designing concurrent, distributed, and real-time applications. Chapter 1 starts with a brief description of the difference between a method and a notation, followed by a discussion of the characteristics of real-time and distributed applications. Chapter 2 presents a brief overview of the aspects of the UML notation used by the COMET method. Next, there is a description of the important design concepts (Chapter 3) and necessary technology support (Chapter 4) for concurrent and distributed systems. This is followed in Chapter 5 by a brief survey of software life cycles and design methods.

Part II of the book describes the COMET method (Concurrent Object Modeling and architectural design mEThod). In Chapter 6, there is an overview of the object-oriented software life cycle used by COMET. Chapter 7 describes the requirements modeling phase of COMET, in particular, use case modeling, and

Chapters 8 through 11 describe the analysis modeling phases of COMET. Chapters 12–16 describe the design modeling phase of COMET. Chapter 17 describes the performance analysis of real-time designs using real-time scheduling—in particular, rate monotonic analysis.

Finally, in Part III, the COMET method is illustrated through five detailed case studies of concurrent application design: two real-time design case studies, one client/server case study, and two distributed application case studies. The real-time Elevator Control System case study is described in Chapter 18, with both non-distributed and distributed solutions presented. The client/server Banking System case study is described in Chapter 19. The real-time Cruise Control System case study is described in Chapter 20. The distributed Factory Automation case study is described in Chapter 21, and the distributed Electronic Commerce case study is described in Chapter 22.

Ways to Read This Book

This book may be read in various ways. Reading it in the order it is presented, Chapters 1–5 provide introductory concepts and technology, Chapter 6 provides an overview of COMET, Chapters 7–17 provide an in-depth treatment of designing applications with COMET, and Chapters 18–22 provide detailed case studies.

Part I is introductory and may be skipped by experienced readers, who will want to proceed directly to the description of COMET in Part II. Readers familiar with the UML may skip Chapter 2. Readers familiar with software design concepts may skip Chapter 3. Readers familiar with concurrent and distributed system technology may skip Chapter 4. Readers familiar with software life cycles and methods may skip the survey in Chapter 5. Readers particularly interested in COMET may proceed directly to Parts II and III. Readers particularly interested in distributed application design should read Chapters 4, 12, and 13, the additional information on concurrent subsystem design in Chapters 14–16, as well as the distributed application case studies in Chapters 18, 19, 21, and 22. Readers particularly interested in real-time design and scheduling should read Chapters 4, 14–17, and the hard real-time design case studies in Chapters 18 and 20.

Experienced designers may also use this book as a reference, referring to various chapters as their projects reach that stage of the analysis or design process. Each chapter is relatively self-contained. For example, at different times, you might refer to Chapter 7 for a concise description of use cases, Chapter 10 when designing statecharts, Chapter 11 for developing the dynamic model, Chapter 13

for distributed component design, Chapter 14 when designing concurrent tasks, or Chapter 17 for real-time scheduling. You can also understand how to use the COMET method by reading the case studies, because each case study explains the decisions made at each step of the design process.

Acknowledgments

The author gratefully acknowledges the reviewers of earlier drafts of the manuscript. Of the reviewers, he is particularly grateful to Jeff Magee, Larry McAlister, Kevin Mills, Robert G. Pettit IV, and Maria Ericsson for their insightful reviews. I would also like to thank Anhtuan Q. Dinh, Ghulam Ahmad Farrukh, Johan Galle, Kelli Houston, Jishnu Mukerji, Leslee Probasco, Sanjeev Setia, and Duminda Wijesekera for their helpful reviews.

Additional thanks are due to Kevin Mills for his contributions on the use of stereotypes in COMET, Shigeru Otsuki for his assistance with the section on design patterns, Roger Alexander for his help with one of the examples in Chapter 15, and Larry McAlister, who contributed Figure 21.1. Particular thanks are due to Tyrrell Albaugh for her hard work coordinating the lengthy production process, to Kristin Erickson who coordinated the editorial process, and to Malinda McCain for her meticulous copyediting of the manuscript.

The author is also grateful to his students in his Software Design and Software Project Lab courses at George Mason University for their enthusiasm, dedication, and valuable feedback. The author gratefully acknowledges the Software Engineering Institute (SEI) for the material provided on real-time scheduling, on which parts of Chapter 17 are based. The author also gratefully acknowledges the Software Productivity Consortium's sponsorship of the development of an earlier version of the material described in Chapters 9–11 of this book. He also thanks Arman Anwar, Hua Lin, and Michael Shin for their hard work producing earlier versions of the figures.

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

Contents

Foreword by Peter Freeman xiii
Foreword by Bran Selic xv
Preface xvii

PART I UML NOTATION, DESIGN CONCEPTS, TECHNOLOGY, LIFE CYCLES, AND METHODS 1

1 Introduction 3
1.1 Object-Oriented Methods and the Unified Modeling Language 4
1.2 Method and Notation 5
1.3 Concurrent Applications 6
1.4 Real-Time Systems and Applications 8
1.5 Distributed Systems and Applications 10
1.6 Summary 11

2 Overview of UML Notation 13
2.1 UML Diagrams 13
2.2 Use Case Diagrams 14
2.3 UML Notation for Classes and Objects 14
2.4 Class Diagrams 15
2.5 Interaction Diagrams 17
2.6 Statechart Diagrams 19
2.7 Packages 20
2.8 Concurrent Collaboration Diagrams 21

2.9 Deployment Diagrams 23
2.10 UML Extension Mechanisms 24
2.11 The UML as a Standard 25
2.12 Summary 26

3 Software Design and Architecture Concepts 27
3.1 Object-Oriented Concepts 27
3.2 Information Hiding 30
3.3 Inheritance 36
3.4 Active and Passive Objects 37
3.5 Concurrent Processing 38
3.6 Cooperation between Concurrent Tasks 40
3.7 Information Hiding Applied to Access Synchronization . . 49
3.8 Monitors 51
3.9 Design Patterns 53
3.10 Software Architecture and Component-Based Systems . . . 55
3.11 Summary 56

4 Concurrent and Distributed System Technology 57
4.1 Environments for Concurrent Processing 57
4.2 Runtime Support for Multiprogramming and Multiprocessing Environments 60
4.3 Task Scheduling 63
4.4 Operating System Input/Output Considerations 65
4.5 Client/Server and Distributed System Technology 68
4.6 World Wide Web Technology 73
4.7 Distributed Operating System Services 75
4.8 Middleware 78
4.9 Common Object Request Broker Architecture (CORBA) . . 81
4.10 Other Component Technologies 85
4.11 Transaction Processing Systems 86
4.12 Summary 88

5 Software Life Cycles and Methods 91
5.1 Software Life Cycle Approaches 91
5.2 Design Verification and Validation 98
5.3 Software Testing 99
5.4 Evolution of Software Design Methods 101

5.5 Evolution of Object-Oriented Analysis and Design Methods 103
5.6 Survey of Concurrent and Real-Time Design Methods ... 105
5.7 Summary 106

PART II COMET: CONCURRENT OBJECT MODELING AND ARCHITECTURAL DESIGN WITH UML 107

6 Overview of COMET 109
6.1 COMET Object-Oriented Software Life Cycle 109
6.2 Comparison of the COMET Life Cycle with Other Software Processes 112
6.3 Requirements, Analysis, and Design Models 113
6.4 The COMET in a Nutshell 115
6.5 Summary 118

7 Use Case Modeling 119
7.1 Use Cases 119
7.2 Actors 120
7.3 Actors, Roles, and Users 123
7.4 Identifying Use Cases 123
7.5 Documenting Use Cases in the Use Case Model 124
7.6 Examples of Use Cases 125
7.7 Use Case Relationships 130
7.8 Use Case Packages 134
7.9 Summary 135

8 Static Modeling 137
8.1 Associations between Classes 137
8.2 Composition and Aggregation Hierarchies 145
8.3 Generalization/Specialization Hierarchy 147
8.4 Constraints 149
8.5 Static Modeling and the UML 149
8.6 Static Modeling of the System Context 152
8.7 Static Modeling of Entity Classes 155
8.8 Summary 157

9 Object and Class Structuring 159
9.1 Object Structuring Criteria 160
9.2 Categorization of Application Classes 160