UML与面向对象设计影印丛书

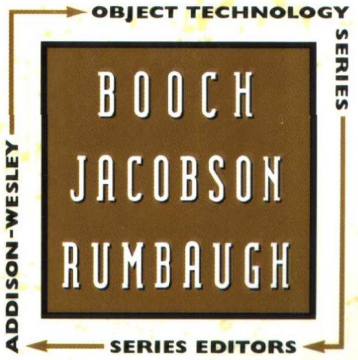
构建可扩展数据库应用程序

BUILDING SCALABLE DATABASE APPLICATIONS

OBJECT-ORIENTED DESIGN, ARCHITECTURES, AND IMPLEMENTATIONS

(美) PETER M. HEINCKIENS 编著

Foreword by Mary Loomis



科学出版社
www.sciencep.com

Pearson Education

UML 与面向对象设计影印丛书

构建可扩展数据库应用程序

Building Scalable Database Applications
Object-Oriented Design, Architectures, and Implementations

（美） Peter M. Heinckiens 编著

科学出版社

北京

图字：01-2003-7659 号

内 容 简 介

本书是介绍使用可重用商务模式建立 Web 应用程序的综合性、权威性的指导书籍。本书重点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。

本书可供数据库系统分析与设计人员阅读和参考。

图书在版编目(CIP)数据

构建可扩展数据库应用程序=Building Scalable Database Applications:Object-Oriented Design,Architectures,and Implementations/ （美） Peter M. Heinckiens 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 7-03-012493-6

Ⅰ.构… Ⅱ.H… Ⅲ.数据库系统—程序设计—英文 Ⅳ.TP311.13

中国版本图书馆 CIP 数据核字（2003）第 103054 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：21
印数：1—3 000 字数：399 000

定价：36.00 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

A fundamental trend in computing is the evolution of applications from monolithic codes to component-based systems. Evidence of increasing software modularization and distribution is clear. The typical design center for application software has evolved from a huge mainframe orientation to a two-tier, client/server perspective, in which user-interface logic resides on desktop clients and the balance of the application executes on remote servers. The server commonly supports large, well-established databases and is accessed by a variety of applications. This two-tier, client/server model is evolving further to a three-tier model, with legacy databases remaining on centrally administered servers and application logic executed by mid-tier departmental servers. The next step is even more completely distributed applications. We are creating a new understanding of what an application is. That is, an application becomes a collect of well-modularized applets that interact in a network computing environment to accomplish the work of the application.

Designing client/server and distributed applications is a complex, challenging proposition. There are so many variables to consider and variants to analyze. If you've been faced with this challenge, you may have found it difficult to leverage the experience and expertise of others in the area, partly because so little has been published. This book should help to make your job easier because it shares real experience and expertise on how to design client/server applications.

Coupled with the trend toward modularization and distribution is that of more often using object technologies for designing and implementing applications. These two trends are quite synergistic and complementary; each reinforces the other. Object technology is well-suited for distributed environments, and distributed applications are well-served by object technology.

"Object-oriented" has become one of the more popular adjectives of this period in computing. At today's computer-industry trade shows, an amazing percentage of products

are boldly labeled "object-oriented," to the point that the term is becoming a bit empty. In fact, however, object technology is exceptionally important. It encompasses object-oriented analysis and design techniques and methods, object programming languages, object database management systems, object request brokers and services, and so on. This book explains an approach for building robust applications by leveraging a stable and reusable business model that is developed using principles of object modeling.

The approach of Peter Heinckiens advocates separating not only an application's user-interface from its logic, but also separating the application's logic and its data persistence (or database) aspects. This additional cleavage plane is analogous to the extension of a two-tier client/server to a three-tier architecture. Not only are the user-interface modules separated from the application logic, but it also is easy to isolate the persistence mechanism. This approach is especially applicable in situations in which you want to build an application using an object programming language and want that application to access data that already resides in a non-object-oriented persistent store, for example, in a relational database or a conventional file system.

Peter's approach is not entirely new. For some time, others have been designing and implementing three-tier client/server and distributed applications using object modeling and object programming languages with relational database management systems. However, not much of that experience has been captured in the printed word. In contrast, Peter's book is rich with code examples that will help you understand exactly how to follow his footsteps in order to get a working system.

Building Scalable Database Applications discusses a variety of topics of importance relative to object-oriented application design. Here's a partial list of what it does:

- Presents a persistence architecture that allows for clear separation of object-oriented business models and relational database models.
- Shows how to abstract the details of database concepts and terminology so that you can concentrate on the fundamentals of persistence.
- Explains how to approach the design of reusable business objects.
- Shows how to approach the design of systems that are open and extensible.
- Emphasizes the pragmatic aspects of designing applications.
- Focuses on leveraging relational databases into object-oriented programming environments.
- Not only offers theory, but also shows how to apply that theory in practice.

Although I don't necessarily agree with everything Peter says in this book, I find it interesting and thought-provoking. It is definitely a step forward in the purveyance of practical information about how to combine notions of object-oriented programming and relational database storage. It should prove to be a valuable resource for many designers and implementers of modern applications.

Mary E. S. Loomis, PhD
Palo Alto, California

Preface

Perhaps the biggest problem the software industry has been coping with is the inability to travel in time, both forward and backward. Forward, because it would be nice to know in advance what our system specifications will end up looking like. Backward, because then we could change the mistakes we made and now have to live with. Thus our present is dictated by our past, and in its turn, our present dictates our future. So, we had better make sure that the choices we make today are well considered, or we might regret them for years to come. However, since we cannot predict the future, we must try to make our choices in such a way that our options stay open.

This situation is even more prevalent today. Because of the rapid technological advances the software industry faces, continually bigger demands are made by customers of software products. Those same customers are also experts in adding or changing "some very minor points" to the program requirements that often result in a programmer having to rewrite almost the entire program. This problem is often the cause of frustration between developers and customers. Developers get frustrated because the customer's modifications come so late in the project that major reprogramming is needed. Customers get frustrated because they had expected a more flexible collaboration.

Database Applications

The observation in the previous section particularly holds for many of today's management information system (MIS) applications. Most of these applications have very long life cycles, often spanning several technological waves in the fields of both programming and database technologies, and they often require extensions or modifications. It is impossible to rewrite the software every time the technology changes. However, at the same

time, customers want to use the latest technological evolutions (just think of the World Wide Web, for example).

Traditionally, database applications were designed using programming languages such as COBOL or, in the best case, C. The main objections to these approaches have been that writing an application takes far too long and that afterward it contains too many bugs and is not sufficiently maintainable or portable. The portability issue is very important. Often, the (potential) customer already possesses a database system and wants your application to work on it. If that system differs from the one that your library of reusable routines was developed for, you might have a problem. An even more frustrating situation occurs when you demonstrate one of your applications to potential customers, and they agree that it is just what they have been waiting for all their life—if only it had not been written for a different database system than the one that they are currently using.

To increase software development productivity, Rapid Application Development (RAD) has become very popular and has given explosive birth to fourth-generation language (4GL) systems. However, being able to develop applications fast is one thing. How do we cope with the entire software life cycle, particularly maintenance and extensibility? Recent study has shown that in several companies, more than 93% of all software efforts go to program maintenance. And this figure is increasing.

The current silver bullet to conquer the software werewolf [Brooks87] is said to be object orientation. As a result, most RAD products got an additional label stuck on them: *object-oriented.* But what does *object orientation* really mean? Does it mean we should be able to draw or inherit fancy user interfaces? Does it mean we have to program in Java, Smalltalk, or C++? Or does it perhaps mean that we should migrate to an object-oriented database system?

Database technology, too, is an area in which rapid advances are being made. As a result, we are stuck with a mixture of several technologies: network databases, relational databases, flat-file systems, object-oriented databases, and so on. How do we integrate these? If we need to buy a new system, which technology do we choose? What if this choice turns out to be wrong? Or what if, in a couple of years, we have to migrate to a new technology?

The database is playing an increasingly important role in software design. However, what is a *database application* exactly? Does the database really play such an all-important role in these applications that it justifies letting it dominate the entire software approach?

All of these questions, and plenty more, are being asked every day by application developers. These developers need a database in which to store their data, but they are finding it increasingly difficult to present a suitable solution to their customers' ever-growing software demands.

To survive the explosive technological evolution in the software industry, we need an entirely new view of the concept of databases or, rather, of persistence as a whole. The concept of persistence has to be abstracted from the actual persistence technology.

What to Expect from This Book

In this book, I sketch a picture of the issues concerning database applications and how to handle them in an object-oriented way. The concepts and views introduced here are illustrated by the use of a persistence architecture called Scoop, short for *scalable object-oriented persistence*. I describe the fundamentals of the Scoop architecture and use it to explain how to develop database applications using object-oriented techniques.

The main objective of this book is to present the concept of object storage from an application developer's point of view. Somehow we have to manipulate persistent data. How do we do this, and, more important, how do we do this without having to throw out the tools we already possess?

Various aspects of software design are covered, including

- Determining the position of the database
- Developing an object-oriented view on the database
- Designing reusable business components
- Modeling and implementing associations
- Separating the user interface from the business model
- Designing database applications in such a way that maximum reuse and openness are achieved

I emphasize how to write software that conforms to a three-tier, client/server architecture. That is, I focus on how to obtain maximum separation and independence between the user interface, the application logic, and the storage.

A case study of a real-world application illustrates the concepts and techniques presented in this book. This study is interesting in that it describes an application that has been implemented in multiple versions. One version was implemented using the Scoop architecture, while others were developed using a number of 4GLs. This case study allows you to compare the object-oriented approach to the data-driven approach offered by most 4GL systems.

Intended Audience

Many application developers are committed to relational database management systems (RDBMSs). Although they may be interested in object orientation, they are often hesitant (or unable) to throw out existing relational or legacy products. This book shows how they can continue using their existing products and still benefit from an object-oriented approach. More generally, this book is of importance to anyone interested in object-oriented design and is unwilling to be committed to a specific database system. On the contrary, by using the techniques described here developers will be able to run their applications on a wide range of low-end and high-end database management systems (DBMSs).

Although some parts of this book are quite technical, several chapters should also be useful to managers. They will provide managers responsible for directing the information systems strategy of a company with some insights in and background about issues relevant to "modern client/server software." Especially of interest to them should be Chapters 1, 2, 3, 4, 11, 12, and 13.

Another target group is the designers who need access to database systems for applications that are not really database-related. This book will show them how they can eliminate most of the details concerned with databases and thus be able to focus on the fundamental parts of their programs.

Although the examples are given in C++ and the reader is expected to have some familiarity reading C++ code, the reader does not have to be an expert C++ programmer to benefit from the book. Knowledge of object-orientation is helpful; however, the reader need not be an object-oriented specialist. All that is required to benefit from the book is an open mind.

Feedback

Comments, criticisms, and suggestions about this book are greatly appreciated. They can be sent by e-mail to Peter Heinckiens at *Peter.Heinckiens@rug.ac.be*.

Acknowledgments

Many of the ideas proposed in this book originated from the experience I obtained working with Professor Ghislain Hoffman. It was he who, many years ago, sparked my enthusiasm for object orientation, and he is one of the first people I met who seemed to fully understand this concept and its implications for today's software systems. Working with him not only provided me with an invaluable experience, but also taught me a necessary lesson in pragmatism.

This book has benefited substantially from many fruitful discussions I had with Philippe Van Damme, Herman Tromp, and Johan Hoffman. They served as sounding boards for my often only partially formed ideas, and they meticulously reviewed my draft manuscript. I am particularly grateful to Philippe, who not only reviewed my final draft, but who read and reread my manuscript from its early days. His comments led to often heated discussions, but they were of immense value and resulted in many improvements to this book.

The detailed reviews by Rick Cattell, Margaret Ellis, John Lakos, and Mary Loomis are greatly appreciated. Margaret and John not only went into the technical aspects of my manuscript, but also did a very thorough job of polishing my English. Mary and Rick provided some very pertinent suggestions. Mary's offer to write my foreword was a big encouragement for me, not only because of my respect for her professionally, but especially because of my respect for her as a person.

I received helpful comments and support from many of my friends and colleagues. I particularly want to thank Bob Adams, Peter Arnold, Kris Carron, Daniel Chang, Patrick Delbeke, Hendrik Devos, Martin Hedley, Michael Hoffman, Rony Lanssiers, Geert Premereur, Herman Steyaert, Misty Taylor, Bart Van den Berghe, Bart Van Renterghem, and Richard Wiener.

I want to give special thanks to my family—Michelle Van Hollebeke, Marc Heinckiens, Jeannine Fiers, Masha Heinckiens, and Mo Khalfa—for their love, support, and understanding.

They not only supported me morally, but also helped with another major task I undertook during the writing of this book: renovating a house. Without them, I would not have been able to finish both of these jobs successfully.

Finally, I want to thank Katie Duffy, Mike Hendrickson, Marina Lang, and Marilyn Rash at Addison Wesley Longman and Laura Michaels of Montview Publications for their very professional and pleasant way of working with me. Katie, Mike, and Marina have been very supportive and helpful throughout this whole project. Marilyn made sure that I (almost) made all my deadlines, and Laura did an excellent job of copyediting.

About the Author

Peter M. Heinckiens holds the Belgian equivalent of an M.S. degree in electrical engineering. He works for the Information Technology Department of the University of Ghent, where he is responsible for coordinating the strategic planning and deployment of software technology throughout the university's administrative section. His function at the University puts him in the unique position of being able to do research while also confronting the real issues and problems that business faces today.

In addition, he often teaches and consults for industry. As a consultant, he has been involved in introducing object-oriented techniques in large-scale projects. Many of these projects were designed using the techniques described in this book. He is also a frequent speaker at international conferences and is a contributor to several technical and scientific magazines.

Contents