原 版 风 暴 系 列

PEARSON
Addison Wesley

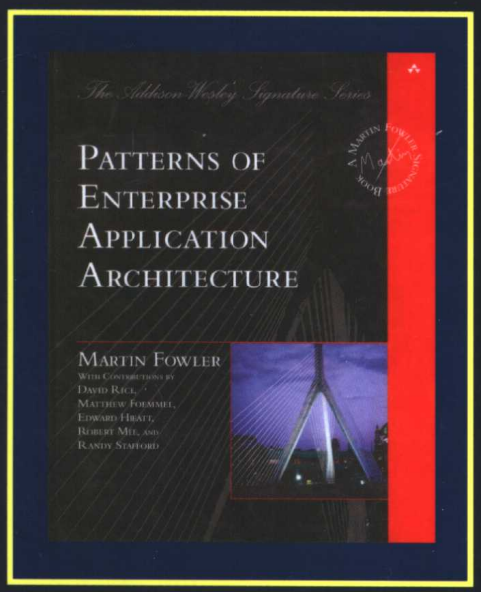
Patterns of Enterprise Application Architecture

企业应用架构模式

（影印版）

[美] Martin Fowler 著

美国
Software Development
Productivity Award
大奖作品



- JavaWorld 评选的最佳 Java 图书
- 《重构》作者 Martin Fowler 最新作品
- 40 余种模式覆盖企业应用架构常见问题
- 介绍每种模式的用法和实现并附有范例代码

Pearson Education

中国电力出版社
www.infopower.com.cn

原版风暴系列

Patterns of Enterprise Application Architecture

企业应用架构模式

（影印版）

[美] Martin Fowler 著

中国电力出版社
www.infopower.com.cn

Patterns of Enterprise Application Architecture(ISBN 0-321-12742-0)
Martin Fowler

北京市版权局著作权合同登记号 图字：01-2004-1301
For sale and distribution in the People's Republic of China exclusively (except Taiwan, Hong Kong SAR and Macao SAR).
仅限于中华人民共和国境内（不包括中国香港、澳门特别行政区和中国台湾地区）销售发行。

图书在版编目（CIP）数据
企业应用架构模式 /（美）福勒（Fowler, M.）著. 影印本. —北京：中国电力出版社，2004
（原版风暴系列）
ISBN 7-5083-2201-0
Ⅰ.企... Ⅱ.福... Ⅲ.软件工具－程序设计－应用－企业管理－英文 Ⅳ.F270.7
中国版本图书馆 CIP 数据核字（2004）第 029865 号

丛 书 名: 原版风暴系列
书　　名: 企业应用架构模式（影印版）
编　　著: （美）Martin Fowler
责任编辑: 姚贵胜
出版发行: 中国电力出版社
地址：北京市三里河路6号　　邮政编码：100044
电话：（010）88515918　　传　　真：（010）88518169
印　　刷: 北京丰源印刷厂
开　　本: 787×1092　1/16　　印　　张: 34
书　　号: ISBN 7-5083-2201-0
版　　次: 2004 年 5 月北京第 1 版　　2004 年 5 月第 1 次印刷
定　　价: 49.80 元

For Denys William Fowler, 1922–2000
in memoriam

—Martin

Preface

In the spring of 1999 I flew to Chicago to consult on a project being done by ThoughtWorks, a small but rapidly growing application development company. The project was one of those ambitious enterprise application projects: a back-end leasing system. Essentially it deals with everything that happens to a lease after you've signed on the dotted line: sending out bills, handling someone upgrading one of the assets on the lease, chasing people who don't pay their bills on time, and figuring out what happens when someone returns the assets early. That doesn't sound too bad until you realize that leasing agreements are infinitely varied and horrendously complicated. The business "logic" rarely fits any logical pattern, because, after all, it's written by business people to capture business, where odd small variations can make all the difference in winning a deal. Each of those little victories adds yet more complexity to the system.

That's the kind of thing that gets me excited: how to take all that complexity and come up with a system of objects that can make the problem more tractable. Indeed, I believe that the primary benefit of objects is in making complex logic tractable. Developing a good *Domain Model (116)* for a complex business problem is difficult but wonderfully satisfying.

Yet that's not the end of the problem. Our domain model had to be persisted to a database, and, like many projects, we were using a relational database. We also had to connect this model to a user interface, provide support to allow remote applications to use our software, and integrate our software with third-party packages. All of this on a new technology called J2EE, which nobody in the world had any real experience in using.

Even though this technology was new, we did have the benefit of experience. I'd been doing this kind of thing for ages with C++, Smalltalk, and CORBA. Many of the ThoughtWorkers had a lot of experience with Forte. We already had the key architectural ideas in our heads, and we just had to figure out how

to apply them to J2EE. Looking back on it three years later, the design is not perfect but it has stood the test of time pretty damn well.

That's the kind of situation this book was written for. Over the years I've seen many enterprise application projects. These projects often contain similar design ideas that have proven effective in dealing with the inevitable complexity that enterprise applications possess. This book is a starting point to capture these design ideas as patterns.

The book is organized in two parts, with the first part a set of narrative chapters on a number of important topics in the design of enterprise applications. These chapters introduce various problems in the architecture of enterprise applications and their solutions. However, they don't go into much detail on these solutions. The details of the solutions are in the second part, organized as patterns. These patterns are a reference, and I don't expect you to read them cover to cover. My intention is that you read the narrative chapters in Part 1 from start to finish to get a broad picture of what the book covers; then you dip into the patterns chapters of Part 2 as your interest and needs drive you. Thus, the book is a short narrative book and a longer reference book combined into one.

This is a book on enterprise application design. Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data. Examples include reservation systems, financial systems, supply chain systems, and many others that run modern business. Enterprise applications have their own particular challenges and solutions, and they are different from embedded systems, control systems, telecoms, or desktop productivity software. Thus, if you work in these other fields, there's nothing really in this book for you (unless you want to get a feel for what enterprise applications are like.) For a general book on software architecture, I'd recommend [POSA].

There are many architectural issues in building enterprise applications. I'm afraid this book can't be a comprehensive guide to them. In building software I'm a great believer in iterative development. At the heart of iterative development is the notion that you should deliver software as soon as you have something useful to the user, even if it's not complete. Although there are many differences between writing a book and writing software, this notion is one that I think the two share. That said, this book is an incomplete but (I trust) useful compendium of advice on enterprise application architecture. The primary topics I talk about are

- Layering of enterprise applications
- Structuring domain (business) logic

- Structuring a Web user interface
- Linking in-memory modules (particularly objects) to a relational database
- Handling session state in stateless environments
- Principles of distribution

The list of things I don't talk about is rather longer. I really fancied writing about organizing validation, incorporating messaging and asynchronous communication, security, error handling, clustering, application integration, architectural refactoring, structuring rich-client user interfaces, among other topics. However, because of space and time constraints and lack of cogitation, you won't find them in this book. I can only hope to see some patterns for this work in the near future. Perhaps I'll do a second volume someday and get into these topics, or maybe someone else will fill these and other gaps.

Of these, message-based communication is a particularly big issue. People who are integrating multiple applications are increasingly making use of asynchronous message-based communication approaches. There's much to be said for using them within an application as well.

This book is not intended to be specific for any particular software platform. I first came across these patterns while working with Smalltalk, C++, and CORBA in the late '80s and early '90s. In the late '90s I started to do extensive work in Java and found that these patterns applied well to both early Java/CORBA systems and later J2EE-based work. More recently I've been doing some initial work with Microsoft's .NET platform and find the patterns apply again. My ThoughtWorks colleagues have also introduced their experiences, particularly with Forte. I can't claim generality across all platforms that have ever been or will be used for enterprise applications, but so far these patterns have shown enough recurrence to be useful.

I have provided code examples for most of the patterns. My choice of language for them is based on what I think most readers are likely to be able to read and understand. Java is a good choice here. Anyone who can read C or C++ can read Java, yet Java is much less complex than C++. Essentially most C++ programmers can read Java but not vice versa. I'm an object bigot, so I inevitably lean to an OO language. As a result, most of the code examples are in Java. As I was working on the book, Microsoft started stabilizing its .NET environment, and its C# language has most of the same properties as Java for an author. So I did some of the code examples in C# as well, although that introduced some risk since developers don't have much experience with .NET and so the idioms for using it well are less mature. Both are C-based languages, so if you can read one

you should be able to read both, even if you aren't deeply into that language or platform. My aim was to use a language that the largest amount of software developers can read, even if it's not their primary or preferred language. (My apologies to those who like Smalltalk, Delphi, Visual Basic, Perl, Python, Ruby, COBOL, or any other language. I know you think you know a better language than Java or C#. All I can say is I do, too!)

The examples are there for inspiration and explanation of the ideas in the patterns. They aren't canned solutions; in all cases you'll need to do a fair bit of work to fit them into your application. Patterns are useful starting points, but they are not destinations.

Who This Book Is For

I've written this book for programmers, designers, and architects who are building enterprise applications and who want to improve either their understanding of architectural issues or their communication about them.

I'm assuming that most of my readers will fall into two groups: those with modest needs who are looking to build their own software and readers with more demanding needs who will be using a tool. For those of modest needs, my intention is that these patterns should get you started. In many areas you'll need more than the patterns will give you, but I'll provide you more of a headstart in this field than I got. For tool users I hope this book will give you some idea of what's happening under the hood and also help you choose which of the tool-supported patterns to use. Using, say, an object-relational mapping tool still means that you have to make decisions about how to map certain situations. Reading the patterns should give you some guidance in making the choices.

There is a third category; those with demanding needs who want to build their own software. The first thing I'd say here is to look carefully at using tools. I've seen more than one project get sucked into a long exercise at building frameworks, which wasn't what the project was really about. If you're still convinced, go ahead. Remember in this case that many of the code examples in this book are deliberately simplified to help understanding, and you'll find you'll need to do a lot tweaking to handle the greater demands you face.

Since patterns are common solutions to recurring problems, there's a good chance that you have already come across some of them. If you've been working in enterprise applications for a while, you may well know most of them. I'm not claiming to present anything new in this book. Indeed, I claim the opposite—this is a book of (for our industry) old ideas. If you're new to this field, I

hope the book will help you learn about these techniques. If you're familiar with the techniques, I hope the book will help you communicate and teach them to others. An important part of patterns is trying to build a common vocabulary, so you can say that this class is a *Remote Facade (388)* and other designers will know what you mean.

Acknowledgments

As with any book, what's written here has a great deal to do with the many people who have worked with me in various ways over the years. Lots of people have helped in lots of ways. Often I don't recall important things people said that went into this book, but I can acknowledge those contributions I do remember.

I'll start with my contributors. David Rice, a colleague of mine at ThoughtWorks, has made a huge contribution—a good tenth of the book. As we worked hard to hit the deadline (while he was also supporting a client), we had several late-night instant message conversations where he confessed to finally seeing why writing a book is both so hard and so compulsive.

Matt Foemmel is another ThoughtWorker, and although the Arctic will need air conditioning before he writes prose for fun, he's been a great contributor of code examples (as well as a very succinct critic of the book.) I was pleased that Randy Stafford contributed *Service Layer (133)* as he's been such a strong advocate for it. I'd also like to thank Edward Hieatt and Rob Mee for their contribution, which arose from Rob's noticing a gap while he was doing his review of the text. He became my favorite reviewer: Not only does he notice something missing, he helps write a section to fix it!

As usual, I owe more than I can say to my first-class panel of official reviewers:

John Brewer	Rob Mee
Kyle Brown	Gerard Meszaros
Jens Coldewey	Dirk Riehle
John Crupi	Randy Stafford
Leonard Fenster	David Siegel
Alan Knight	Kai Yu

I could almost list the ThoughtWorks telephone directory here, for so many of my colleagues have helped this project by talking over their designs and experiences with me. Many patterns formed in my mind because I had the

opportunity to talk with the many talented designers we have, so I have little choice but to thank the whole company.

Kyle Brown, Rachel Reinitz, and Bobby Woolf have gone out of their way to have long and detailed review sessions with me in North Carolina. Their fine-tooth comb has injected all sorts of wisdom, not including this particularly heinous mixed metaphor. In particular I've enjoyed several long telephone calls with Kyle that contributed more than I can list.

Early in 2000 I prepared a talk for Java One with Alan Knight and Kai Yu that was the earliest genesis of this material. As well as thanking them for their help in that, I should also thank Josh Mackenzie, Rebecca Parsons, and Dave Rice for helping me refine these talks, and the ideas, later on. Jim Newkirk did a great deal in helping me get used to the new world of .NET.

I've learned a lot from the many people working in this field with whom I've had good conversations and collaborations. In particular I'd like to thank Colleen Roe, David Muirhead, and Randy Stafford for sharing their work on the Foodsmart example system at Gemstone. I've also had great conversations at the Crested Butte workshop that Bruce Eckel has hosted and must thank all the people who attended that event in the last couple of years. Joshua Kerievsky didn't have time to do a full review, but he was an excellent patterns consultant.

As usual, I had the remarkable help of the UIUC reading group with their unique brand of no-holds-barred audio reviews. My thanks to: Ariel Gertzenstein, Bosko Zivaljevic , Brad Jones, Brian Foote, Brian Marick, Federico Balaguer, Joseph Yoder, John Brant, Mike Hewner, Ralph Johnson, and Weerasak Witthawaskul.

Dragos Manolescu, an ex-UIUC hitman, got his own group together to give me feedback. My thanks to Muhammad Anan, Brian Doyle, Emad Ghosheh, Glenn Graessle, Daniel Hein, Prabhaharan Kumarakulasingam, Joe Quint, John Reinke, Kevin Reynolds, Sripriya Srinivasan, and Tirumala Vaddiraju.

Kent Beck has given me more good ideas than I can remember. But I do remember that he came up with the name for *Special Case (496)*. Jim Odell was responsible for getting me into the world of consulting, teaching, and writing—no acknowledgment will ever do his help justice.

As I was writing this book, I put drafts on the Web. During this time many people sent me e-mails pointing out problems, asking questions, or talking about alternatives. These people include Michael Banks, Mark Bernstein, Graham Berrisford, Bjorn Beskow, Bryan Boreham, Sean Broadley, Peris Brodsky, Paul Campbell, Chester Chen, John Coakley, Bob Corrick, Pascal Costanza, Andy Czerwonka, Martin Diehl, Daniel Drasin, Juan Gomez Duaso, Don Dwiggins, Peter Foreman, Russell Freeman, Peter Gassmann, Jason Gorman, Dan Green,

Lars Gregori, Rick Hansen, Tobin Harris, Russel Healey, Christian Heller, Richard Henderson, Kyle Hermenean, Carsten Heyl, Akira Hirasawa, Eric Kaun, Kirk Knoernschild, Jesper Ladegaard, Chris Lopez, Paolo Marino, Jeremy Miller, Ivan Mitrovic, Thomas Neumann, Judy Obee, Paolo Parovel, Trevor Pinkney, Tomas Restrepo, Joel Rieder, Matthew Roberts, Stefan Roock, Ken Rosha, Andy Schneider, Alexandre Semenov, Stan Silvert, Geoff Soutter, Volker Termath, Christopher Thames, Volker Turau, Knut Wannheden, Marc Wallace, Stefan Wenig, Brad Wiemerslage, Mark Windholtz, Michael Yoon.

There are many others who gave input whose names I either never knew or can't remember, but my thanks is no less heartfelt.

My biggest thanks is, as ever, to my wife Cindy, whose company I appreciate much more than anyone can appreciate this book.

Colophon

This is the first book that I wrote using XML and related technologies. The master text was written as a series of XML documents using trusty TextPad. I also used a home-grown DTD. While I was working I used XSLT to generate the web pages for the HTML site. For the diagrams I relied on my old friend Visio using Pavel Hruby's wonderful UML templates (much better than those that come with the tool. I have a link on my Web site if you want them.) I wrote a small program that automatically imported the code examples into the output, which saved me from the usual nightmare of code cut and paste. For my first draft I tried XSL-FO with Apache FOP. At the time it wasn't quite up to the job, so for later work I wrote scripts in XSLT and Ruby to import the text into FrameMaker.

I used several open source tools while working on this book—in particular, JUnit, NUnit, ant, Xerces, Xalan, Tomcat, Jboss, Ruby, and Hsql. My thanks to the many developers of these tools. There was also a long list of commercial tools. In particular, I relied on Visual Studio for .NET and on IntelliJ's wonderful Idea—the first IDE that's excited me since Smalltalk—for Java.

The book was acquired for Addison Wesley by Mike Hendrickson who, assisted by Ross Venables, has supervised its publication. I started work on the manuscript in November 2000 and released the final draft to production in June 2002. As I write this, the book is due for release in November 2002 at OOPSLA.

Sarah Weaver was the production editor, coordinating the editing, composition, proofreading, indexing, and production of final files. Dianne Wood was

the copy editor, carrying out the tricky job of cleaning up my English without introducing any untoward refinement. Kim Arney Mulcahy composed the book into the design you see here, cleaned up the diagrams, set the text in Sabon, and prepared the final Framemaker files for the printer. The text design is based on the format we used for *Refactoring*. Cheryl Ferguson proofread the pages and ferreted out any errors that had slipped through the cracks. Irv Hershman prepared the index.

About the Cover Picture

During the couple of years I spent writing this book a more significant construction project was going on in Boston. The Leonard P. Zakim Bunker Hill Bridge (try fitting that name on a road sign) will replace the ugly double-decker that now carries Interstate 93 over the Charles River. The Zakim bridge is a cable-stayed bridge, a style that hasn't been widely used in the U.S. so far, but is very popular in Europe. The Zakim bridge isn't particularly long, but it is the world's widest cable-stayed bridge and also the first U.S. cable-stayed bridge to have an asymmetric design. It's a very beautiful bridge, but that doesn't stop me from teasing Cindy about Henry Petroski's conjecture that we are due for a major failure in a cable-stayed bridge soon.

Martin Fowler, Melrose, Massachusetts, August 2002
http://martinfowler.com

Contents

Preface xv
Who This Book Is For xviii
Acknowledgments xix
Colophon xxi

Introduction 1
Architecture 1
Enterprise Applications 2
Kinds of Enterprise Application 5
Thinking About Performance 6
Patterns 9
The Structure of the Patterns 11
Limitations of These Patterns 13

PART 1: The Narratives 15

Chapter 1: Layering 17
The Evolution of Layers in Enterprise Applications 18
The Three Principal Layers 19
Choosing Where to Run Your Layers 22

Chapter 2: Organizing Domain Logic 25
Making a Choice 29
Service Layer 30

Chapter 3: Mapping to Relational Databases 33
Architectural Patterns 33
The Behavioral Problem 38

Reading in Data 40
Structural Mapping Patterns 41
Mapping Relationships 41
Inheritance 45
Building the Mapping 47
Double Mapping 48
Using Metadata 49
Database Connections 50
Some Miscellaneous Points 52
Further Reading 53

Chapter 4: Web Presentation 55

View Patterns 58
Input Controller Patterns 61
Further Reading 61

Chapter 5: Concurrency *(by Martin Fowler and David Rice)* 63

Concurrency Problems 64
Execution Contexts 65
Isolation and Immutability 66
Optimistic and Pessimistic Concurrency Control 67
Preventing Inconsistent Reads 68
Deadlocks 70
Transactions 71
ACID 71
Transactional Resources 72
Reducing Transaction Isolation for Liveness 73
Business and System Transactions 74
Patterns for Offline Concurrency Control 76
Application Server Concurrency 78
Further Reading 80

Chapter 6: Session State 81

The Value of Statelessness 81
Session State 83
Ways to Store Session State 84

Chapter 7: Distribution Strategies 87

The Allure of Distributed Objects 87
Remote and Local Interfaces 88

Where You Have to Distribute 90
Working with the Distribution Boundary 91
Interfaces for Distribution 92

Chapter 8: Putting It All Together 95
Starting with the Domain Layer 96
Down to the Data Source Layer 97
Data Source for *Transaction Script (110)* 97
Data Source for *Table Module (125)* 98
Data Source for *Domain Model (116)* 98
The Presentation Layer 99
Some Technology-Specific Advice 100
Java and J2EE 100
.NET 101
Stored Procedures 102
Web Services 103
Other Layering Schemes 103

PART 2: The Patterns 107

Chapter 9: Domain Logic Patterns 109
Transaction Script 110
How It Works 110
When to Use It 111
The Revenue Recognition Problem 112
Example: Revenue Recognition (Java) 113
Domain Model 116
How It Works 116
When to Use It 119
Further Reading 119
Example: Revenue Recognition (Java) 120
Table Module 125
How It Works 126
When to Use It 128
Example: Revenue Recognition with a Table Module (C#) 129
Service Layer *(by Randy Stafford)* 133
How It Works 134
When to Use It 137

Further Reading 137
Example: Revenue Recognition (Java) 138

Chapter 10: Data Source Architectural Patterns 143

Table Data Gateway 144
How It Works 144
When to Use It 145
Further Reading 146
Example: Person Gateway (C#) 146
Example: Using ADO.NET Data Sets (C#) 148
Row Data Gateway 152
How It Works 152
When to Use It 153
Example: A Person Record (Java) 155
Example: A Data Holder for a Domain Object (Java) 158
Active Record 160
How It Works 160
When to Use It 161
Example: A Simple Person (Java) 162
Data Mapper 165
How It Works 165
When to Use It 170
Example: A Simple Database Mapper (Java) 171
Example: Separating the Finders (Java) 176
Example: Creating an Empty Object (Java) 179

Chapter 11: Object-Relational Behavioral Patterns 183

Unit of Work 184
How It Works 184
When to Use It 189
Example: *Unit of Work* with Object Registration (Java) *(by David Rice)* 190
Identity Map 195
How It Works 195
When to Use It 198
Example: Methods for an *Identity Map* (Java) 198

Lazy Load 200
How It Works 200
When to Use It 203
Example: Lazy Initialization (Java) 203
Example: Virtual Proxy (Java) 203
Example: Using a Value Holder (Java) 205
Example: Using Ghosts (C#) 206

Chapter 12: Object-Relational Structural Patterns 215

Identity Field 216
How It Works 216
When to Use It 220
Further Reading 221
Example: Integral Key (C#) 221
Example: Using a Key Table (Java) 222
Example: Using a Compound Key (Java) 224
Foreign Key Mapping 236
How It Works 236
When to Use It 239
Example: Single-Valued Reference (Java) 240
Example: Multitable Find (Java) 243
Example: Collection of References (C#) 244
Association Table Mapping 248
How It Works 248
When to Use It 249
Example: Employees and Skills (C#) 250
Example: Using Direct SQL (Java) 253
Example: Using a Single Query for Multiple Employees (Java) *(by Matt Foemmel and Martin Fowler)* 256
Dependent Mapping 262
How It Works 262
When to Use It 263
Example: Albums and Tracks (Java) 264
Embedded Value 268
How It Works 268
When to Use It 268