Expert One-on-One[™] J2EE[™] Development without EJB[™]

Rod Johnson with Juergen Hoeller



Expert One-on-One[™] J2EE[™] Development without EJB[™]

Expert One-on-One[™] J2EE[™] Development without EJB[™]

Rod Johnson with Juergen Hoeller



Expert One-on-One J2EE[™] Development without EJB[™]

Copyright © 2004 by Rod Johnson and Juergen Hoeller. All rights reserved.

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4447, E-mail: permcoordinator@wiley.com.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOT THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HERE-FROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAP-PEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Trademarks: Wiley, the Wiley Publishing logo, Wrox, the Wrox logo, Programmer to Programmer, Expert One-on-One, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates. J2EE and EJB are trademarks of Sun Microsystems, Inc. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Johnson, Rod, Ph.D.
Expert one-on-one J2EE development without EJB / Rod Johnson, Juergen Hoeller.
p. cm.
Includes bibliographical references and index.
ISBN 0-7645-5831-5 (paper/website)
1. Java (Computer program language) 2. Computer software—Development. I. Hoeller, Juergen, 1975– II.
Title.
QA76.73.J38J62 2004
005.13'3—dc22

2004005516

ISBN: 0-7645-5831-5 Printed in the United States of America 10 9 8 7 6 5 4 3 2 1

About the Authors

Rod Johnson is an enterprise Java architect with extensive experience in the insurance, dot-com, and financial industries. He was the J2EE architect of one of Europe's largest web portals, and he has worked as a consultant on a wide range of projects.

Rod has an arts degree majoring in music and computer science from the University of Sydney. He obtained a Ph.D. in musicology before returning to software development. With a background in C and C++, he has been working with both Java and J2EE since their release. He is actively involved in the Java Community Process as a member of the JSR-154 (Servlet 2.4) and JDO 2.0 Expert Groups. He is the author of the best-selling *Expert One-on-One J2EE Design and Development* (Wrox, 2002) and has contributed to several other books on J2EE since 2000.

Rod is prominent in the open source community as co-founder of the Spring Framework open source project (www.springframework.org), which grew out of code published with *Expert One-on-One J2EE Design and Development*. He speaks frequently at leading industry conferences. He is currently based in London.

Rod can be contacted at expert@interface21.com.

I'd like to thank my wife, Kerry, for her continuing love and support. Of those who've given practical help, I'm grateful for contributions from Gary Watson, Andrew Smith, and Jason Carreira for their thorough review of the entire manuscript; Alef Arendsen (reviewing and valuable performance benchmarking); Peter den Haan (thorough review of several chapters); Renaud Pawlak (rigorous review of the AOP material); and Steve Jefferson, Thomas Risberg, and Dmitriy Kopylenko (reviewing).

I'm also grateful to the many developers and architects who have shared their experiences of J2EE development with me, in person and via e-mail.

As always, working with Juergen has been a pleasure.

Juergen Hoeller is a Senior Systems architect and Consultant at werk3AT, a company that delivers complex web solutions and provides J2EE-based consulting in Austria.

Juergen has a masters degree in Computer Science from the University of Linz, specializing in Java, OO modeling, and software engineering. He has worked on a wide range of projects with numerous J2EE application servers, ranging from enterprise application integration to web-based data visualization. Juergen has particular experience in developing J2EE web applications, O/R mapping, and transaction management.

Juergen is co-lead of the Spring Framework and active in many community forums, including TheServerSide.

Most of all, I'd like to thank my spouse, Eva, for her boundless love and support, and for her understanding of my constant lack of time.

Special thanks to my colleagues at werk3AT and in particular to Werner Loibl for respecting all of my activities, and for giving valuable input to Spring and this book.

I'm grateful to Thomas Risberg and Alef Arendsen for their thorough reviews and valuable input, and to all developers who helped sharpen the arguments, both within and outside the Spring team.

It has been a particular pleasure to work with Rod on both Spring and this book.Introduction

Credits

Vice President and Executive Group Publisher Richard Swadley

Vice President and Executive Publisher Bob Ipsen

Vice President and Publisher Joseph B. Wikert

Executive Editorial Director Mary Bednarek

Executive Editor Robert Elliott

Editorial Manager Kathryn A. Malm

Development Editor Adaobi Obi Tulton **Technical Editors**

Gary Watson Andrew Smith Jason Carreira

Production Editors

Felicia Robinson Eric Newman

Copy Editors

C. M. Jones Michael Koch

Media Development Specialist Kit Malone

Text Design & Composition Wiley Composition Services

About the Authors	V
Introduction	xvii
Chapter 1: Why "J2EE without EJB"?	1
EJB Under the Spotlight	1
What's Left of J2EE?	3
J2EE at a Crossroads	4
The Way Forward	5
Themes	5
Lightweight Frameworks and Containers	10
Should We Ever Use EJB?	11
Summary	12
Chapter 2: Goals	13
Productivity	13
The Problem	14
The Traditional J2EE Approach to Productivity Issues	15
Better Solutions for Higher Productivity	20
00	26
The Importance of Business Requirements	28
The Importance of an Empirical Process	28
Summary	29
Chapter 3: Architectures	31
Architectural Building Blocks	31
The Business Services Layer	31
Exposing Business Objects to the World	35
Data Access Layer, or EIS Tier	40
J2EE Architectures	42
EJB Architectures	42
Non-EJB Architectures	47

J2EE Architectures in Practice	54
"Classic" J2EE Remote EJB Architectures	54
Local EJB Architectures	57
Ad hoc Non-EJB Architectures	59
"Lightweight Container" Architecture: The Sample Application	61
Deciding Whether an Application Needs	
an Application Server	62
Summary	63
Chapter 4: The Simplicity Dividend	65
The Cost of Complexity	65
Causes of Complexity in J2EE Applications	66
Architectural Causes of Complexity	66
Cultural Causes of Complexity: The Complexity Industry	71
How Much Complexity is too Much Complexity?	75
Simple or Naïve?	75
Just Good Enough?	77
The Winds of Change	77
Summary	78
Chapter 5: EJB, Five Years On	81
Hype and Experience	81
EJB and the J2EE Industry	82
EJB in Practice	82
An Aging Component Model	82
Java Language Improvements	83
The .NET Challenge	83
Web Services	85
The Rise of Agile Methodologies	86
Confusion Regarding the Aims of EJB	86
The Component Market That Didn't Eventuate	88
The New Paradigm on the Block: The Emergence of AOP	88
What Do We Really Want from EJB, or Why Stateless Session Beans	
Are So Popular	89
Declarative Transaction Management	90
Remoting	92
Clustering	92
Inread Management	94
EJB INSTANCE POOLING	94
Resource Pooling	95
Security	95

Business Object Management	96
EJB Services Summary	97
The Monelithie Distinct Container Problem	31
Inclogence and the Proliferation of Classes	90
	100
Class Loader Hell	100
Testing	100
EJB Overdose	102
Complex Programming Model	102
Simple Things Can Be Hard	103
Is the Goal of Enabling Developers to Ignore the Complexity	
of Enterprise Applications Even Desirable?	103
Loss of Productivity	104
Portability Problems	104
Can EJB Reinvent Itself?	104
Tool Support	104
EJB 3.0	105
Myths and Fallacies	105
J2EE == EJB	106
Questionable Arguments for Using EJB	106
Chassing Whather to Lies EIP	107
Conventional Wiedom	107
Making a Choice Today	107
The Emerging Post FIR Consensus	100
Standards Innovation and Open Source	109
Summary	112
Chapter 6: Lightweight Containers and Inversion of Control	121
Lightweight Containers	122
What Is a Lightweight Container?	122
Why Do We Need a Container at All?	124
Lightweight Containers versus EJB	125
Managing Business Objects	126
Interface-implementation Separation	126
EJB: Only a Partial Solution	127
Inversion of Control	127
IoC Implementation Strategies	128
IoC Containers	135
Portability between IoC Containers	137

Implications for Coding Style, Testing, and Development Process	138
Coding Style	138
Testability	139
Development Process	139
Applying Enterprise Services	139
Summary	141
Chapter 7: Introducing the Spring Framework	143
History and Motivation	143
A Layered Application Framework	144
Basic Building Blocks	145
Spring on J2EE	146
Spring for Web Applications	147
The Core Bean Factory	149
Fundamental Interfaces	149
Populating Beans via XML	151
Non-XML Bean Definition Formats	154
Wiring up Application Objects	155
Autowire and Dependency Checks	159
Constructor Resolution	160
Lifecycle Callbacks	162
Complex Property Values	164
Resource Setup	165
Classic Java and J2EE Resource Access	166
Resource Definitions in a Bean Container	168
Factory Beans	171
The Spring Application Context	175
Lifecycle Callbacks	177
Message Source	178
File Resources	180
Bean Factory Post-processing	182
Summary	184
Chapter 8: Declarative Middleware Using AOP Concepts	187
AOP 101	188
Motivation	188
AOP in J2EE	190
Definitions	191
History	194

EJB as a Subset of AOP	195
AOP Implementation Strategies	197
Dynamic Proxies	197
Dynamic Byte Code Generation	198
Java Code Generation	198
Use of a Custom Class Loader	198
Language Extensions	198
AOP Implementations	199
AspectJ	199
AspectWerkz	201
JBoss 4	201
Spring	203
Nanning	207
The AOP Alliance	207
AOP Design Issues	207
Dangers of AOP	207
AOP Design Recommendations	210
J2EE à la carte	211
AOP in Practice with Spring	212
Using the ProxyFactoryBean	213
Convenience Factory Beans	217
"Autoproxying"	218
Programmatic Usage	219
Using Source-level Metadata to Provide	
an Abstraction above AOP	220
.NET Example	220
Aside: Conceptual Versus Implementation-level Metadata	221
Programmatic Access to Context Information	222
Spring Example	222
EJB 3.0	225
Implications for Programming Style	225
Consistent Naming Conventions	225
Avoiding Reliance on the AOP Infrastructure	226
Checked Exceptions and Advice	227
References	227
Books	227
Papers	227
Articles and Online Resources	227
Summary	228

Chapter 9: Transaction Management	231
High-level Transaction Management	231
Classic J2EE Transaction Management	232
The J2EE Container as Transaction Coordinator	233
Everybody Loves CMT	234
Direct Use of JTA	236
Interlude: Remote Transaction Propagation	237
Lightweight Transaction Infrastructure	238
Transaction Management with the Spring Framework	239
Transaction Definition	240
Programmatic Transaction Demarcation	243
Declarative Transaction Demarcation	246
Transaction Management Strategies	251
Implications for J2EE Server Choice	257
Summary	258
Chapter 10: Persistence	261
Common Persistence Strategies	262
An Overview of Persistence Patterns	262
Popular J2EE Data Access Solutions	263
Choosing a Persistence Strategy	265
Transparent Persistence and Behavior in Domain Objects	268
A Brief History of Java Persistence Technologies	268
The Slow Rise of Java O/R Mapping Solutions	269
The Failure of Entity Beans	271
Data Access Technologies in Practice	271
Resource Handling	272
JDBC	273
iBATIS SQL Maps	275
JDO	278
Hibernate	281
The Data Access Object Pattern	285
Business Objects and Data Access Objects	285
DAOs and Transparent Persistence	287
Types of Data Access Objects	288
DAO Design Issues	289
DAO Infrastructure Issues	292
Data Access with the Spring Framework	293
Generic Data Access Exceptions	293
Business Objects and Data Access Objects Revisited	295

	200
DATIC COL Mana	290
IBATIS SQL Maps	301
JDU Liikamata	302
Hibernate	304
Summary	307
Chapter 11: Remoting	309
Classic J2SE Remoting: RMI	310
Accessing and Exporting RMI Services	311
RMI Invoker for Transparent Remoting	315
Classic J2EE Remoting: EJB	316
Wire Protocols	317
State Management	318
Accessing Remote EJBs	319
Deploying Remote EJBs	324
WSDL-based Web Services: JAX-RPC	325
Accessing Web Services	327
Servlet and EJB Endpoints	332
Lightweight Remoting: Hessian and Burlap	335
Accessing and Exporting Hessian and Burlap Services	336
Summary	339
Chaper 12: Replacing Other EJB Services	341
Thread Management	342
Threading Myths	342
The EJB Threading Model	345
EJB Instance Pooling	346
When Is Pooling Required?	347
The Case Against Instance Pooling	347
Alternatives to EJB Threading and Pooling	349
Threading Models	349
Instance Pooling Summary	358
Declarative Security	359
The EJB Model	359
Flaws in the EJB Model	359
Declarative Security via AOP	359
JMS and Message-driven Beans	360
Summary	360
-	

Chapter 13: Web Tier Design	363
Goals and Architectural Issues	364
Web Tier Design Goals	365
Ad hoc MVC via Servlets and JSPs	366
Integration into Overall Application Architecture	368
Request-driven Web MVC Frameworks	374
Struts 1.1	375
WebWork2	381
Web MVC with the Spring Framework	388
Appropriate View Technologies	401
Alternative Approaches to Web MVC	403
Portals and Portlets	403
Event-driven Web MVC Frameworks	404
A Word on ASP.NET	409
Summary	410
Chapter 14: Unit Testing and Testability	411
Why Testing Matters	412
Goals of Unit Testing	414
Ensuring Testability	415
Programming Style	415
How to Make Your Code Hard to Test	416
Standard Library Challenges	420
Techniques for Improving Testability	421
Inversion of Control	425
AOP	425
Unit Testing Techniques	425
Stubs	425
Mock Objects	426
Writing Effective Tests	430
Test-driven Development (TDD)	433
Benefits	433
Arguments against TDD	434
Practicing TDD	436
Learning TDD	436
Case Study: The Spring Experience	437
Testing Spring Applications	440
Testing POJOs	440
Benefiting from Spring Abstractions	440
When You Do Need to Depend on Spring APIs	441
Testing with an Alternate Configuration	442

Chapter Title

Coverage Analysis and Other Test Tools	443
Test Generators	444
Coverage Tools	444
Mutation Testing Tools	447
Resources	448
Summary	449
Chapter 15: Performance and Scalability	451
Definitions	452
Setting Clear Goals	453
Architectural Choices: The Key to Performance and Scalability	454
Object Distribution, Clusters, and Farms	455
Data Access	461
Other Architectural Issues	462
Implementation Choices	463
The Performance Implications of Dispensing with EJB Service Provision	463
Caching and Code Optimization	471
Tuning and Deployment	476
JVM	476
Application Server	476
Framework Configuration	477
Database Configuration	477
An Evidence-based Approach to Performance	478
Benchmarking	479
Profiling	480
Diagnostics	484
Resources	485
Summary	485
Chapter 16: The Sample Application	489
Pet Store Requirements	490
The iBATIS JPetStore 3.1	490
Middle Tier	491
Remoting	493
Room for Improvement	494
Spring JPetStore	494
Middle Tier	496
Data Access Tier	499
Web Tier	502
Remoting	510

Build and Deployment	516
WAR Deployment Issues	516
Deploying the Spring JPetStore	519
Summary	520
Chapter 17: Conclusion	521
Looking Back	521
Moving Forward	523
Choosing the Best Architecture for Your Application	524
The Lightweight Container Architecture	524
Standards	526
Is Spring the Only Alternative to EJB?	527
Key Messages	529
Guidelines	530
Architecture	531
Programming Style	532
Inversion of Control (IoC) and Dependency Injection	532
AOP	533
Testing	534
Last words	535
Index	537

Introduction

This is a fairly short book, given its scope, because its subject is less complex than you've been led to believe.

J2EE orthodoxy makes heavy work of many simple problems. Indeed it sometimes seems that the J2EE industry is committed to the belief that there are no simple problems.

Many—probably most—J2EE applications are over-engineered, with unnecessarily complex architectures. Over-engineering can be very costly. J2EE developers tend to assume that increased cost up front will be more than balanced by reductions in future costs. Unfortunately, karma doesn't apply to software engineering, and this is often a fallacy. Greater complexity up front means more code to write and maintain, more potential for bugs, more delay in demonstrating functionality to users: ultimately, greater chance of failure, and at greater cost.

J2EE over-engineering usually involves EJB. As I pointed out in *Expert One-on-One J2EE Design and Development*, EJB is often used inappropriately. This is a real problem, because EJB can introduce more complexity than it conceals. Some services provided by EJB are also overrated. For example, few experienced developers or architects who have worked with entity EJBs to access relational data want to repeat the experience—at least, given the alternatives of JDO, Hibernate, and other transparent persistence technologies.

Critiques of EJB have become commonplace since late 2002. It's easy enough to pick the flaws in an imperfect existing technology, without suggesting alternatives. This book breaks new ground in describing and illustrating better approaches for the majority of J2EE applications that derive no benefit from EJB. This book is no mere theoretical discussion, but a practical guide to designing and implementing high-performance J2EE applications on time and budget. Our suggested architectures are backed up by extensive experience, a realistic sample application, and comprehensive open source solutions that meet typical infrastructure needs.

Despite the significant problems that have emerged with EJB, it continues to be adopted too often largely because of fashion and fear. Fashion because even nontechnical managers have heard of EJB and because many books on J2EE architecture barely mention alternatives to EJB for delivering enterprise services, even where excellent alternatives exist. Fear that the alternatives are worse: for example, that without EJB developers will be left to handle complex issues such as transaction management without the training wheels of the EJB container. This book aims to show that these fears are largely unfounded. Where the potential complexity is real, it shows that there are alternatives that do a better job than EJB at addressing the problems concerned.

This book demonstrates a much simpler approach to developing typical J2EE applications than the "classic" J2EE blueprints approach exemplified by the original Java Pet Store. Our approach leads to reduced cost, shorter time to market, greater maintainability, and better performance. The architectural approach described in this book is part of a growing movement towards simpler, more rational J2EE architectures. It's suitable for use with agile methodologies. It draws on recently developed technologies such as Aspect Oriented Programming, and borrows where appropriate from alternative platforms to J2EE such as .NET.

I aim to help you build the simplest possible applications that meet your requirements—and hence, also the cheapest, most maintainable and testable.

The merits of EJB have become a surprisingly emotive issue in the J2EE community. There seems to be a stark polarization between those would never use EJB unless compelled and those who believe that the EJB skeptics are lazy, ignorant heretics, with little middle ground.

As you may suspect, I'm not a fan of EJB. However, I have developed many applications with EJB and speak from experience and intimate knowledge of the EJB specification. I'll also strive to justify my position throughout this book. My goal is to help you develop effective applications, not to combat the use of EJB.

After reading this book, you should be able to assess the value proposition of EJB for each application. If, with a strong understanding of EJB and the alternatives, you believe that your requirements are best addressed by EJB, use EJB. The message of this book is not a black-and-white "don't use EJB."

Who This Book Is For

This book is for J2EE architects and developers who have a solid grasp of the technology and want to use it more productively. It's a book about the *why* and *how* of enterprise applications, rather than the *what*. So you won't find API listings here, and you won't find yet another introduction to basic J2EE services such as JNDI and JTA. There are many good books and articles that address these topics.

The material in this book is particularly relevant to those working on web applications. However, most J2EE developers will find something of value. You may well read this book and decide that EJB is the correct choice for your particular application; even in this case, by applying the criteria set out in this book, you'll know exactly why you are using EJB and what value it is providing.

Aims of This Book

This book aims to help you solve practical problems. It aims to demonstrate a simpler and more productive approach to J2EE development than the traditional J2EE approach exemplified in Sun blueprints and based on the use of EJB to manage business logic.

You might end up having a lot more fun, as well.

What This Book Covers

This book covers:

- D Problems with EJB and received wisdom in J2EE architecture
- □ Key values for successful J2EE projects

- □ Effective architectures for J2EE applications, especially for web applications
- □ Common mistakes in J2EE architecture and implementation
- □ How to find the simplest and most maintainable architecture for your application
- □ Inversion of Control and Aspect-Oriented Programming: two important new technologies that have recently become important to J2EE development.

Chapters 6 through 12 cover replacing EJB services with lighter-weight, more testable alternatives. We emphasize:

- □ Transaction management. This is an essential of enterprise applications, and a popular motivation for using EJB. We'll look at alternative means of transaction management, discussing both declarative and programmatic transaction management.
- Data access in J2EE applications: another central problem that—in contrast to transaction management—EJB addresses very badly.
- □ How AOP can be used to solve many common problems in enterprise software development.

We'll also talk about:

- U Web tier design, and the place of the web tier in a well-designed J2EE application
- Testing and test-driven development, and how to test J2EE applications effectively
- Performance and scalability

Specific technologies considered include:

- Data access with JDBC, Hibernate, and JDO
- U Web tier technologies such as Struts, WebWork, Spring MVC, and JSP
- □ Using open source products to develop a strong application infrastructure, and minimize the amount of code you'll have to develop, test, and maintain in house. Most problems of J2EE infrastructure have been solved by leading open source products; I'll help you focus on tackling those unique to your applications.

Assumed Knowledge

This is not a primer on EJB or J2EE.

We assume knowledge of core J2EE APIs and services such as JNDI, JTA, and database connection pooling.

We assume sound knowledge of J2SE, including reflection, inner classes, dynamic proxies, JDBC, JAXP, and JNDI.

We assume good working knowledge of OO design.

Introduction

You won't need detailed knowledge of EJB, as this book is about *alternatives* to EJB, but it will help if you are familiar with EJB. If you want to get up to speed, try Ed Roman's *Mastering Enterprise JavaBeans*, *Second Edition* (Wiley, 2001).

You will need to understand the basic middleware concepts behind EJB, such as resource pooling, remoting, and transaction management; the basic motivation for adopting n-tier rather than client-server architectures.

You'll need to understand the basic concepts of web interfaces, and the MVC architectural pattern as used in J2EE web applications.

Don't worry if you aren't already familiar with AOP; in Chapter 8 I provide an introduction that enables you to start implementing AOP-enabled applications, and includes a reading list to help you gain deeper knowledge.

Recommended Reading

This book is the sequel to *Expert One-on-One J2EE Design and Development* (Wrox, 2002). You can read this book on its own, but you may want to refer to that book. In particular, it describes in detail many programming practices that are mentioned more briefly here. Chapters 4, 9, and 11–13 are particularly relevant as background to this book.

I also highly recommend Martin Fowler's *Patterns of Enterprise Application Architecture* (Addison-Wesley, 2002): a wise discussion of many problems in enterprise software, with a healthy distance from implementation technologies such as J2EE. Martin Fowler is one of my favorite writers, and he's always worth reading. Fowler's First Law of Distributed Objects ("Don't distribute your objects") is worth the price alone. This book also introduces the term POJO (Plain Old Java Object), coined by Fowler to give plain Java objects buzzword compliance. I'll use it throughout this book.

As I believe that J2EE applications should be OO applications, I highly recommend the classic OO text, *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, and Vlissides, Addison-Wesley, 1995). The 23 patterns listed in this book are still the most valuable in J2EE applications—more valuable than technology-centric "J2EE" patterns.

The second edition of *Core J2EE Patterns* (Alur, Crupi, and Malks, 2003) is important reading, partly because it defines a standard naming for J2EE patterns. I'll refer to several patterns from this book, such as Service Locator, Business Delegate, and Front Controller, and it will be helpful if you already understand them. *Core J2EE Patterns* exemplifies more traditional thinking on J2EE architecture (although the second edition is a worthwhile update), but is nonetheless a very useful resource.

I recommend you keep up to date on current J2EE topics. Some of my favorite J2EE websites are:

- □ TheServerSide. The major J2EE portal, this is a great place to keep up to date with developments in J2EE. You'll find discussions on many J2EE issues, and valuable resources such as articles and book chapters and product reviews.
- Artima.com (www.artima.com). Java/J2EE-oriented software site, run by Bill Venners.
- □ Core J2EE Patterns web site (www.corej2eepatterns.com/index.htm).

Various blogs. Some very important ideas have been explored and discussed amongst Java bloggers. www.javablogs.com is a good starting point for exploring this important information channel. Significant bloggers include Rickard Oberg, Ted Neward, Gavin King, Bob Lee ("Crazy Bob"), and Jon Tirsen.

It's useful to keep up to date on middleware in general, not just J2EE. .NET has a similar overall architecture to J2EE, and the growing patterns literature on .NET is relevant to J2EE. Useful resources include:

- □ MSDN home (http://msdn.microsoft.com/)
- "Enterprise Solution Patterns Using Microsoft .NET" website (http://msdn.microsoft.com/architecture/patterns/Esp/default.aspx)

What You Need to Use This Book

To run the sample application and examples, you'll need:

- □ A J2EE web container or/and application server. For examples using a single database, without the need for JTA, we used Tomcat 5.0.16. For examples using JTA, we used WebLogic 8.1 Express. All the code in this book runs unchanged on most other application servers (an incidental benefit of avoiding EJB!), so feel free to deploy the code onto your favorite application server. Please see the release notes with the sample application for information on which application servers it has been tested on.
- □ A relational database and the appropriate JDBC drivers. We used HSQLDB (http://hsqldb.sourceforge.net/) for the examples, but only minor changes should be necessary to use any other transactional database.
- □ The Spring Framework, available from www.springframework.org. This site contains many further resources for developing with Spring. The sample application was written for Spring 1.0 final, but should run unchanged on later versions of Spring.
- □ The Hibernate 2.1 object-relational mapping product, available from www.hibernate.org/.
- □ Jakarta Ant 1.5.3, the standard Java build tool. While you may prefer to build your projects with Maven (a more complex and powerful build tool) you should certainly understand Ant and the importance of script-driven, repeatable project build steps.
- Various third-party libraries, including Jakarta Commons Logging and Jakarta Commons Attributes. The necessary jar files are included with the full Spring distribution; see documentation with Spring and the sample application for details.

All this software is open source or free for developer use.

The Sample Application

An innovative feature of this book is that it uses an open source sample application. Authors have limited time; for most of us, writing is a hindrance to earning a living. Thus in many books, the sample application is a poor second to completion of the text. *Expert One-on-One J2EE Design and Development* was no exception in this regard, although it did offer unusually sophisticated infrastructure code. An open source sample application can be more realistic than would be possible through the authors' effort alone. This has the great advantage of avoiding the simplifications often seen in sample applications. In J2EE applications, the devil is in the detail, and the value of the overall approach is measured by its ability to address problems of real-world complexity, rather than over-simplified examples.

The sample application is an implementation of the familiar Java pet store. The pet store has simple, well-understood requirements, meaning that we don't need to describe the problem domain in detail.

The code base was originally based on Clinton Begin's JPetStore, available from www.ibatis.com: a valuable example in its own right. Our sample application enhances the JPetStore to use the architecture we advocate, introducing greater use of interfaces, use of an Inversion of Control container, and AOP-enabled declarative transaction management. We believe that the result is simpler and more flexible and extensible than the original JPetStore. We also invite you to compare it with the original Sun Java Pet Store to see how much simpler our approach is in comparison to the J2EE blueprints approach, and how it replaces EJB services with lighter-weight alternatives.

The sample application illustrates the following points discussed in the text:

- □ A well-defined business interface layer
- □ The *Lightweight Container* architecture, discussed in Chapter 3, built on an Inversion of Control container
- □ The use of AOP to provide declarative transaction management to POJOs. The underlying transaction-management technology can be switched between JTA, JDBC, or another technology without any change to Java code.
- □ Use of an MVC web layer, using either Spring MVC or Struts 1.1. The sample includes both alternative implementations of the web tier, accessing the same business interface layer.
- □ Use of the Data Access Object pattern to decouple business logic from database schema and technology for accessing persistent data
- □ The use of source-level metadata attributes to simplify the programming model, especially for transaction management
- □ Remoting support for multiple protocols, built on the business interface layer.

The sample application is available online at www.wrox.com. Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

Because many books have similar titles, you may find it easiest to search by ISBN; the ISBN for this book is 0-7645-5831-5.

Once you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at www.wrox.com/dynamic/books/download.aspx to see the code available for this book and for all other Wrox books.

Conventions

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

Boxes like this one hold important, not-to-be-forgotten information that is directly relevant to the surrounding text.

Tips, hints, tricks, and asides to the current discussion are offset and placed in italic like this.

As for styles in the text:

- □ We *highlight in italic* important words when we introduce them.
- □ We show keyboard strokes like this: Ctrl+A.
- We show file names, URLs, and code within the text in a monospaced font, like so: persistence.properties.
- We present code in two different ways:

In code examples we highlight new and important code with a gray background. The gray highlighting is not used for code that's less important in the present context, or has been shown before.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, such as a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata you may save another reader hours of frustration and at the same time you will be helping us provide even higher-quality information.

To find the errata page for this book, go to www.wrox.com and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at www.wrox.com/misc-pages/booklist.shtml.

If you don't spot "your" error on the Book Errata page, go to

www.wrox.com/contact/techsupport.shtml and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at http://p2p.wrox.com. The forums are a webbased system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers participate in these forums.

At http://p2p.wrox.com you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

- 1. Go to http://p2p.wrox.com and click the Register link.
- 2. Read the terms of use and click Agree.
- 3. Complete the required information to join as well as any optional information you wish to provide and click Submit.
- 4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as to many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

Why "J2EE Without EJB"?

The traditional approach to J2EE architecture has often produced disappointing results: applications that are more complex than their business requirements warrant, show disappointing performance, are hard to test, and cost too much to develop and maintain.

It doesn't need to be so hard. There is a better way for most applications. In this book, we'll describe a simpler, yet powerful architectural approach, building on experience with J2EE and newer technologies such as Inversion of Control and AOP. Replacing EJB with lighter-weight, more flexible, infrastructure typically produces significant benefits. We and many others have used this approach in many production applications, with better results than are usually produced from traditional architectures.

Let's begin with a quick tour of the topics we'll examine in more detail in later chapters.

EJB Under the Spotlight

Like most of my colleagues, I was excited by the promise of EJB when it first appeared. I believed it was the way forward for enterprise middleware. However, I've since revised my opinions, in the light of my experiences and those of many colleagues.

Much has changed since the EJB specification was conceived:

- Parts of the specification's design now seem dated. For example, dynamic proxies, introduced in J2SE 1.3, call into question the container code generation envisaged in the EJB specification and the multiple source files needed to implement every EJB.
- □ The traditional link between EJB and RMI remoting is looking dated, because of the emergence of web services and the recognition that EJBs sometimes need only local interfaces. EJB is a heavyweight model for objects that don't need to offer remote access.

- □ This is a special case of the fact that basing typical applications around distributed business objects—the architectural choice EJB implements best—has proved problematic.
- □ Usage of EJB indicates its strengths and weaknesses. Most developers and architects have restricted their use of EJB to stateless session beans and (if asynchronous calls are needed) message-driven beans. The relative simplicity of the services provided by the EJB container to support SLSBs means that the overhead of an EJB container is hard to justify in such applications.
- □ Although EJB has been around for five years, and its use is a given in many J2EE projects, it has become apparent that its complexity means that many developers still don't understand it. For example, many developer candidates I interview can't correctly describe how EJB containers handle exceptions and how this relates to transaction management.
- □ The EJB specification is becoming more and more complex in an attempt to address problems with EJB. It's now so long and complex that few developers or architects will have time to read and understand it. With specifications, as with applications, the need for continual workarounds and constantly growing complexity suggests fundamental problems.
- □ The complexity of EJB means that productivity in EJB applications is relatively poor. A number of tools try to address this, from "Enterprise" IDEs to XDoclet and other code generation tools, but the complexity still lurks under the surface and imposes ongoing costs.
- □ Rigorous unit testing and **test driven development** have become increasingly, and deservedly, popular. It's become clear that applications making heavy use of EJB are hard to test. Developing EJB applications **test first** requires a lot of fancy footwork; essentially, minimization of the dependence of application code on the EJB container.
- □ The emergence of **Aspect Oriented Programming** (AOP) points the way to more powerful—yet potentially simpler—approaches to the middleware problems addressed by EJB. AOP can be viewed in part as a more general application of the central EJB concepts, although of course it's much more than a potential replacement to EJB.
- □ Source level metadata attributes, as used in .NET, suggest a superior alternative in many cases to the verbose XML-based deployment descriptors used since EJB 1.1. EJB 3.0 looks like it's heading down that road as well, but it's a way off and will carry a lot of baggage.

Experience has also shown EJB to incur greater cost and deliver fewer benefits than were initially predicted. Developers have encountered intractable problems that weren't apparent when EJB first appeared. Experience has shown that EJB fails to deliver in several areas:

- □ It doesn't necessarily reduce complexity. It *introduces* a lot of complexity.
- □ The entity bean experiment for persistence has largely failed.
- □ Applications using EJB tend to be less portable between application servers than applications using other J2EE technologies, such as servlets.
- □ Despite the promises that EJB would prove the key to scalability, EJB systems often perform poorly and don't necessarily scale up well. Although statistics are hard to come by, anecdotal evidence suggests that the overhead of excessive use of EJB necessitates re-architecture or causes outright failure in a significant number of projects.
- □ EJB can make simple things hard. For example, the Singleton design pattern (or alternatives) is hard to implement in EJB.

All of these issues suggest that it's wise to analyze exactly what the value proposition is before using EJB. I hope to equip you with the tools to do this effectively and dispassionately.

In Chapter 5, we'll talk more about EJB and its problems. In the meantime, let's look at where J2EE is today, where I feel it's going, and how this book will help you deliver real solutions on time and budget.

What's Left of J2EE?

You may be wondering, "What's left of J2EE without EJB?"

The answer is: a great deal. J2EE is much more than EJB. Many J2EE developers believe otherwise, and will tell you so when they see this book on your desk, but a dispassionate analysis of what EJB does, and what J2EE does overall, shows that EJB is only a part of a much bigger and more important picture.

J2EE is essentially about standardizing a range of enterprise services, such as naming and directory services (JNDI), transaction management offering a standard API potentially spanning disparate transactional resources (JTS and JTA), connection to legacy systems (JCA), resource pooling, and thread management. The true power of J2EE lies in these services, and this standardization has done great service to the industry.

EJB, on the other hand, is merely one way of leveraging those valuable services, through a particular component model.

We can still access JNDI, JTA, JCA, resource pooling, and other core J2EE services without using EJB. We can do this by writing code that uses them directly (not as hair-raising as it may seem) or—better—using proven libraries and frameworks that abstract their use without imposing the complexity of EJB.

Only a few EJB container services are unique to EJB, and there are good alternatives to those. For example:

- □ Entity beans are the only dedicated data access components in J2EE. However, they're also the most questionable part of J2EE, and there are much better non-J2EE alternatives, such as Hibernate and JDO. In some applications, JDBC is a better option.
- □ **Container Managed Transactions (CMT):** EJBs are the only part of J2EE to enjoy declarative transaction management. This is a valuable service, but as we'll see in Chapters 8 and 9 we can also achieve declarative transaction management using AOP. CMT is a relatively thin layer over the underlying J2EE JTA service. It would be hard (and foolhardy to attempt) to replace an application server's global transaction management, but it's not so hard to access it to develop an alternative form of CMT.
- □ Thread pooling for business objects: We usually don't need this if we're supporting only web clients (or web services clients going through a servlet engine), because a web container provides thread pooling and there's no need to duplicate it in the business object tier. We do need thread pooling to support remote clients over RMI/IIOP, one case in which EJB remains a good, simple technology choice.

□ (Related) **Thread management for business objects:** the ability to implement EJBs as though they are single-threaded. In my experience this is overrated for stateless service objects (the most useful kinds of EJB). EJB can't eliminate all threading complexity anyway, as problems can remain with objects used by EJB facades. There are good alternatives to EJB thread management, discussed in Chapter 12.

Only in the area of remoting is EJB the only way to implement such functionality in standard J2EE. As we'll see, only in RMI/IIOP remoting is EJB clearly an outstanding remoting technology; there are better alternatives for web services remoting.

There's a strong argument that EJB attempts to address a lot of issues it shouldn't. Take O/R mapping. This is a complex problem to which EJB provides a complex yet under-specified solution (entity beans) that simply ignores some of the central problems, such as mapping objects with an inheritance hierarchy to relational database tables. It would have been better for the designers of the EJB specification to leave this problem to those with much more experience of the issues around object persistence.

J2EE is much more than EJB. Using J2EE without EJB, we don't have to reinvent the wheel. We don't need to reimplement J2EE services, just consider alternative ways of tapping into them.

J2EE at a Crossroads

J2EE is at a fascinating point in its evolution. In many respects it's a great success. It has succeeded in bringing standardization where none existed; it has introduced a welcome openness into enterprise software. It has achieved fantastic industry and developer buy-in.

On the other hand, I feel it has come up short on a number of measures. J2EE applications are usually too expensive to develop. J2EE application projects are at least as prone to failure as pre-J2EE projects. (Which means that the failure rate is unacceptably high; developing software is far too hit-and-miss an affair.) In the areas where J2EE has failed, EJB has usually played a significant part.

J2EE has significant issues with ease of development. As I've said, J2EE applications tend to be unnecessarily complex. This is especially true of J2EE web applications, which, like the Sun Java Pet Store, are often absurdly over-engineered.

J2EE is still a relatively young technology. It's not surprising that it's imperfect. It's time to take stock of where it's worked, and where it hasn't worked so well, so that we can eliminate the negatives and enjoy the positives. Because J2EE contains a lot, this essentially means identifying the subset of J2EE that delivers most value, along with some supplementary infrastructure we need to harness it most effectively.

There is a growing movement in the J2EE community toward simpler solutions and less use of EJB. My previous book, *Expert One-on-One J2EE Design and Development* (2002), was a step in the growth of that movement, but was part of a broader trend. I believe this book represents the next step in defining and popularizing such solutions, but it's important to note that I'm by no means alone. Fellow pioneers include Rickard Oberg and Jon Tirsen (of Nanning Aspects), who have helped to demonstrate the power