Expert One-on-One
J2EE™ Design and Development

Rod Johnson

wrox
Programmer to Programmer™

Expert One-on-One
J2EE™ Design and Development

Rod Johnson

wrox
Programmer to Programmer™

Expert One-on-One J2EE™ Design and Development

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

Library of Congress Card Number: 2003107067

ISBN: 0-7645-4385-7

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/RQ/QW/QT/IN

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: WHILE THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK, THEY MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR YOUR SITUATION. YOU SHOULD CONSULT WITH A PROFESSIONAL WHERE APPROPRIATE. NEITHER THE PUBLISHER NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.

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

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

Trademark Acknowledgments

Wrox has endeavored to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, Wrox cannot guarantee the accuracy of this information.

Credits

Author
Rod Johnson

Commissioning Editor
Craig A. Berry

Technical Editors
Kalpana Garde
Niranjan Jahagirdar

Project Managers
Cilmara Lion
Abbas Rangwala

Author Agent
Nicola Phillips

Index
Adrian Axinte
Andrew Criddle
John Collin

Proof Reader
Chris Smith

Technical Reviewers
Simon Brown
John Carnell
Eric Eicke
Meeraj Kunnumpurath
Todd Lauinger
Eric Paul Schley
Andrew Smith
Tobin Titus
Tom Watkins
David Whitney
Dave Writz

Production Coordinators
Abbie Forletta
Manjiri Karande

Illustrations
Santosh Haware
Manjiri Karande

Cover
Dawn Chellingworth
Natalie O'Donnell

About the Author

Rod Johnson is an enterprise Java architect specializing in scalable web applications. Rod spent two years designing and delivering a J2EE solution for FT.com, Europe's largest business portal, before trekking to Everest Base Camp, parenting a baby, and writing this book. He would like to thank Tristan for providing the toughest of these challenges.

Rod completed an arts degree majoring in music and computer science at the University of Sydney. He obtained a PhD in musicology before returning to software development. With a background in C and C++, Rod has worked with both Java and J2EE since their release. He is currently a member of JSR 154 Expert Group defining the Servlet 2.4 specification.

Rod has contributed several chapters to other Wrox publications including *Professional Java Server Programming* (J2EE and J2EE 1.3 editions) and *Professional Java Server Pages* (2nd edition), and is a reviewer for Wrox Press. He has presented at international conferences including Times Java, Mumbai (2001), and his writing has been featured on java.sun.com.

An Australian, Rod is currently living and working in London.

He can be contacted at expert@interface21.com.

Although all authors seem to thank their families, I had no idea how richly such gratitude was deserved until I wrote this book.

Above all, I would like to thank my wife, Kerry, for her love and support throughout a lengthy and stressful process.

Others to give generous support include my parents-in-law, Don and Dana Jamieson.

Of those who have given practical help, I would like to thank my editor, Craig Berry, for helping me to distill a huge topic into a book; and my former colleague, Dominic Fisher, for his business insight and valuable help in formulating realistic requirements for the sample application.

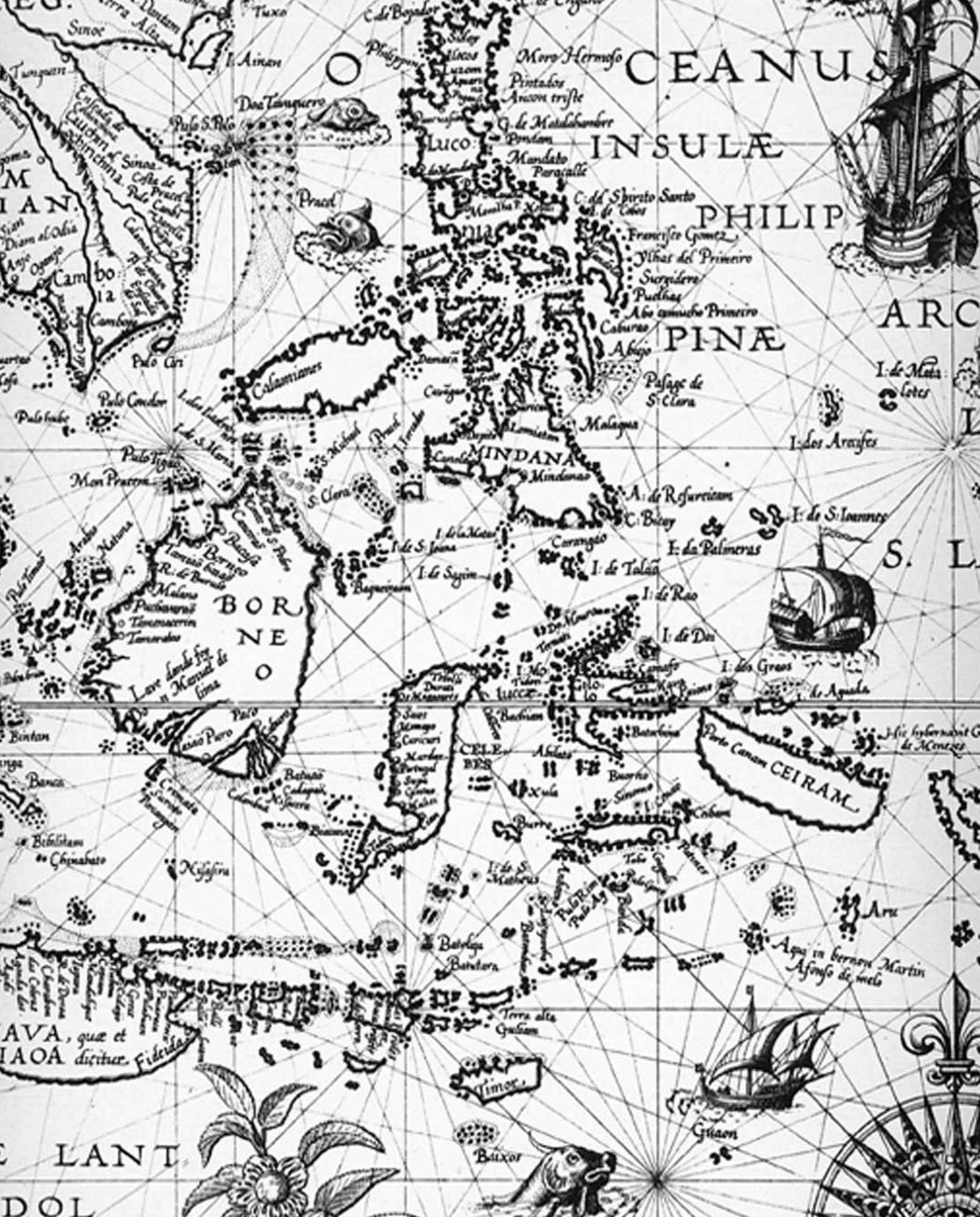
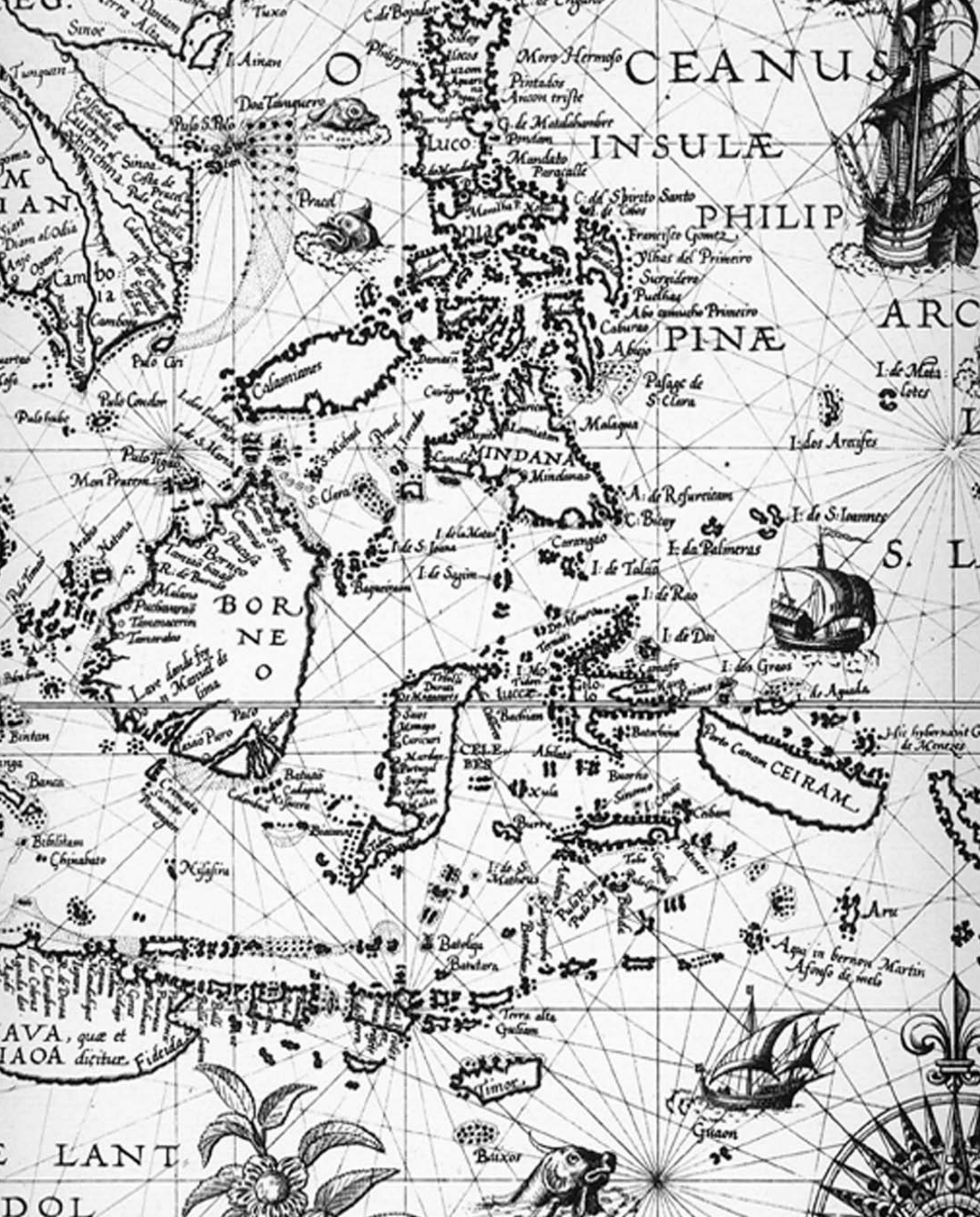

OCEANUS
INSULÆ
PHILIPPINÆ
ARC
S. L
BORNEO
CELEBES
CEIRAM
Porto Camam
MINDANAO
Mindanao
Luçon
Calamianes
Cambo ja
Camboja
I. Ainan
Tuxo
C. de Bojador
Philippinas
Moro Hermoso
Pintados
Ancon triste
G. de Matalabambre
Pondon
Mandato
Paracalle
C. del Spirito Santo
C. de Cobos
Francisco Gomez
Ylhas del Primeiro
Surgidero
Puelhas
Caburao
Abijo
Pasage de S. Clara
I. de Mata lobos
I. dos Arecifes
Malagua
A. de Resurrecçam
C. Bitoy
I. de S. Ioannes
I. da Palmeiras
I. de Talao
I. de Rao
I. de Dei
I. dos Graos
de Aguada
Corangao
I. de la Mata
I. de S. Iuana
I. de Sugin
Bajuriam
S. Clara
Pracel
Doa Tanquerro
Pulo S. Polo
Sinoa
Pulo Cir
Pulo Condor
Pulo Tigao
Mon Pratem
Bintan
Banca
Chinabato
Nussafira
Bachiam
Abilare
Gilolo
Xula
Burro
Batolga
Bantara
I. de S. Matheus
Aru
Terra alta
Gulam
Timor
Baixos
Guaon
Hic hybernavit G. de Menezes
IAVA, quæ et IAOA dicitur
LANT
DOL

Table of Contents


OCEANUS
INSULÆ
PHILIPPINÆ
ARC
S. L
I. Ainan
Tunquem
Sinoa
Camboia
Cambojo
Palo Ceri
Pulo Condor
Pulo Tigao
Mon Pratten
Philippina
Luçon
Moro Hermoso
Pintados
Ancon triste
G. de Matalobambre
Mandato
Paracalle
C. del Spirito Santo
Francisco Gomez
Ylhas del Primeiro
Surgidero
Pueblaq
Caburao
Abrojo
Pasage de S. Clara
Calamianes
Manilla
Malagua
MINDANAO
Mindanao
A. de Resurrecion
C. Bitoy
I. de S. Ioannes
I. da Palmeras
I. de Talao
I. de Rao
I. de Dii
I. dos Graos
I. de Mata lobos
I. dos Arrecifes
I. de Sagin
I. de S. Joana
Corangao
BORNEO
Borneo
Pulo Tiomao
Laue dende foy o Marcial de lima
Bintan
Banca
Bilibitam
Chinabato
Nussafira
CELEBES
Xula
Burro
Ambon
Botavia
Bachian
Gilolo
CEIRAM
Porto Canam
Aru
Aqui in bernou Martin Afonso de melo
I. de S. Matheus
Batolga
Banctara
IAVA, quæ et IAOA dicitur
Terra alta
Timor
Baixos
LANT
DOL

Introduction

I believe that J2EE is the best platform available for enterprise software development today. It combines the proven merits of the Java programming language with the lessons of enterprise software development in the last decade.

Yet this promise is not always fulfilled. The return on investment in many J2EE projects is disappointing. Delivered systems are too often slow and unduly complex. Development time is often disproportionate to the complexity of business requirements.

Why? Not so much because of shortcomings in J2EE as because J2EE is often used badly. This often results from approaches to architecture and development that ignore real world problems. A major contributing factor is the emphasis in many J2EE publications on the J2EE specifications rather than the real world problems people use them to address. Many issues that commonly arise in real applications are simply ignored.

When reading J2EE discussion forums, I'm struck by how little guidance and direction many developers find, and how much time and effort they waste as a result. In many cases, these developers have years of IT experience, and yet are finding it hard to come to grips with J2EE.

The problem is not a lack of information about J2EE components. Many books and web sites do a good job describing servlets, EJBs etc. Enabling technologies such as JNDI, RMI, and JMS are equally well served.

The problem is in getting to the next level – taking these construction materials and using them to build applications that meet real business requirements in a reasonable period of time. Here, I feel that much of the existing literature on J2EE is a hindrance rather than help. There is a gulf between the world of J2EE books – the world as it perhaps should be – and the real world of enterprise software projects.

This book aims to address this problem and provide clear guidance and direction on using J2EE effectively in practice. I'll help you to solve common problems with J2EE and avoid the expensive mistakes often made in J2EE projects. I will guide you through the complexity of the J2EE services and APIs to enable you to build the simplest possible solution, on time and on budget. I'll take a practical, pragmatic approach, questioning J2EE orthodoxy where it has failed to deliver results in practice and suggesting effective, proven approaches.

I feel that no existing book delivers this. The closest is probably *Core J2EE Patterns* from *Prentice Hall (ISBN: 0-130648-84-1)*, which generated much excitement on its release. Here at last was a book that addressed *how* to use J2EE components. *Core J2EE Patterns* is a good book and a valuable resource for J2EE architects and developers. In particular, the terminology it uses has become widely accepted, but it's a Sun publication, and can't help reflecting the "party line".

It also deals purely with the J2EE standards, paying little attention to issues encountered in working with real application servers. It fails to provide clear guidance: too often, it sits on the fence, presenting a variety of very different alternative "patterns". Readers able to choose confidently between them have little to learn from the book.

The more I considered the available publications, sample applications, and discussion forums, the more convinced I became that J2EE needed a healthy dose of pragmatism. J2EE is a great platform; unfortunately, many of the architectural practices promoted for it are not, and don't help to solve many common problems. Many J2EE sample applications, such as Sun's Java Pet Store, are disappointing. They don't face real world problems. They perform poorly, and their code often contains sloppy practices, providing a poor model.

I was also struck by the difference in outlook between developers new to J2EE and those who had actually used J2EE to build enterprise systems. A former colleague used the wonderfully evocative word "gnarly" to describe developers who've come to grips with practical challenges of working with a technology and bear the scars. While those new to J2EE sounded like J2EE evangelists, the "gnarly" developers told a different story. They had had to jettison some of the ideological baggage of the innocents to implement necessary functionality or achieve adequate performance. Like my colleagues and myself, they'd found that reality intruded harshly on the initial vision.

In this book I'll draw on my experience and industry knowledge to help you design and develop solutions that work in practice, without the need for you to go through a painful process of discovering the difference between J2EE theory and reality.

J2EE Myths

I believe that the causes of disappointing outcomes with J2EE can usually be traced to a few common myths, which underpin many explicit and implicit assumptions in development projects:

- J2EE is about portability, between application servers and databases.
- J2EE is the best answer to all the problems of enterprise development. If a problem that would traditionally have been solved using non-J2EE technologies, such as RDBMS stored procedures, can be solved with standard J2EE technology, it's always best to use the "pure" J2EE approach.

- J2EE servers take care of performance and scalability, leaving developers to get on with implementing business logic. Developers can largely ignore the performance implications of J2EE "patterns" and rely on acceptable performance in production.
- J2EE enables developers to forget about low-level problems such as data access and multi-threading, which will be handled transparently by the application server.
- All J2EE applications should use EJB, which is the essential J2EE technology for developing enterprise-class applications.
- Any problems with J2EE will soon be addressed by more sophisticated application servers.

Let's quickly consider each of these myths in turn.

Portability is a great bonus of the J2EE platform. As we'll see, portability *can* be achieved in real applications, but it's not the point of J2EE. The requirement of the vast majority of projects is to build an application that solves a particular problem well on one target platform. An application that runs badly on one platform will never be ported to other platforms (the application might be ported to another operating system that runs on more powerful hardware to gain adequate performance, but that's not the kind of portability that professional developers aspire to).

J2EE orthodoxy holds that an application should be portable across J2EE application servers and must be able to work with different databases. The distinction between these two goals is important, and sometimes missed. Portability between application servers may deliver business value and is usually a realistic goal. Portability between databases is much more fraught, and often provides no business value.

Portability is usually taken to mean *code portability*: the ability to take the application and run it on another platform without any change. I believe that this is an expensive misconception. Naïve emphasis on total code portability often leads to heavy costs in lost productivity and less satisfactory deliverables. **Write Once Run Anywhere (WORA)**, while a reality where Java itself is concerned, is a dangerous slogan to apply to enterprise development, which depends on a range of resources.

> *I'm not talking about the minority of projects to develop "shrink-wrapped" components (usually EJBs). This appealing concept is still to be proven in the market. Furthermore, I'm yet to see a non-trivial component that aimed for both, application server portability (which makes sense in this situation)* and *database portability (which will almost certainly be more trouble than it's worth).*

I prefer **Design Once, Re-implement a Few Interfaces Anywhere (DORAFIA)**. I accept that this is not so catchy, and that people are unlikely to leave Java One chanting it. This more realistic approach is widely used in other domains, such as windowing systems.

The portability myth has led to wide acceptance that J2EE applications can't use the capabilities of today's relational databases, but should use them only as dumb storage. This does great harm in the real world.

This is not to say that I don't believe that J2EE applications can or should be portable. I'm just arguing for a more pragmatic and realistic view of portability. We *can* design J2EE applications to be ported easily; we can't do the same thing with a proprietary technology such as .NET.

It's pleasant to imagine that J2EE is the final stage of the evolution of enterprise architecture; that finally, the application of object technology and the Java language has cracked problems the industry has wrestled with for decades. Unfortunately, this is not the reality, although it's implicitly assumed in many approaches to J2EE development. J2EE builds on many of the technologies that preceded it. It's a step forward, but it won't be the last and it doesn't address all the issues of enterprise software development.

Exaggerated emphasis on portability, along with this J2EE-centric attitude, has led to the assumption that if something can't be done in standard J2EE, it's a design error to do it. This is even creeping into the EJB specification with the introduction of EJB QL: a portable but immature query language that's more complex but far less powerful than the familiar, mature, and largely standard SQL that is available to the great majority of J2EE applications.

I like to think of a J2EE server as the conductor of a group of enterprise resources such as databases. A good conductor is vital to any performance. However, a conductor doesn't attempt to play individual instruments, but leaves this to skilled specialists.

Perhaps the most dangerous myth is that J2EE is the easy route to good performance and scalability, and that efficiency is a lesser concern than approved J2EE "patterns". This leads to naïve and inefficient designs. This is unfortunate, as outside the Java community Java has always been dogged by fears of poor performance. Today, the evidence is that the Java language offers good performance, while some popular J2EE "patterns" offer very poor performance.

We cannot assume that the application server can take care of performance and scalability. In fact, J2EE gives us all the rope we need to tie up not only our J2EE application server, but the database as well. Had optimal performance been the main goal of software development, we'd have been writing web applications in C or assembly language. However, performance *is* vital to the business value of real-world applications. We can't rely on Moore's Law to allow us to solve performance problems with faster hardware. It's possible to create problems that prevent adequate performance, regardless of hardware power.

The idea that the J2EE server should transparently handle low-level details such as data access is appealing. Sometimes it's achievable, but can be dangerous. Again, let's consider the example of relational databases. Oracle, the leading enterprise-class RDBMS, handles locking in a completely different way compared to any other product. The performance implications of using coarse or fine-grained transactions also vary between databases. This means that "portability" can be illusory, as the same code may behave differently in different RDBMS products.

Oracle and other leading products are expensive and have impressive capabilities. Often we'd *want* (or need) to leverage these capabilities directly. J2EE provides valuable standardization in such infrastructure services as transaction management and connection pooling, but we won't be saying goodbye to those fat RDBMS product manuals any time soon.

The "J2EE = EJB" myth can lead to particularly expensive mistakes. EJB is a complex technology that solves some problems well, but adds more complexity than business value in many situations. I feel that most books ignore the very real downside of EJB, and encourage readers to use EJB automatically. In this book, I'll provide a dispassionate view of the strengths and weaknesses of EJB, and clear guidance on when to use EJB.

Allowing the technology used (J2EE or any other technology) to determine the approach to a business problem often leads to poor results. Examples of this mistake include determining that business logic should always be implemented in EJBs, or determining that entity beans are the one correct way to implement data access. The truth is that only a small subset of J2EE components – I would include servlets and stateless session EJBs – are central to most J2EE applications. The value of the others varies greatly depending on the problem in hand.

I advocate a problem-driven, not technology-driven, approach (Sun's "J2EE Blueprints" have probably done as much harm as good, by suggesting a J2EE technology-driven approach). While we should strive to avoid reinventing the wheel, the orthodoxy that we should never ourselves implement something that the server can (however inefficiently), can be costly. The core J2EE infrastructure to handle transaction management, etc., is a godsend; the same cannot be said for all the services described in the J2EE specifications.

Some will argue that all these problems will soon be solved, as J2EE application servers become more sophisticated. For example, ultra-efficient implementations of entity bean **Container-Managed Persistence (CMP)** will prove faster than RDBMS access using raw SQL. This is naïve and carries unacceptable risk. There is little place for faith in IT. Decisions must be made on what has been proven to work, and faith may be misplaced.

There are strong arguments that some features of J2EE, such as entity beans, can *never* be as performant in many situations as some alternatives. Furthermore, the Promised Land is *still* just around the corner. For example, entity beans were soon going to provide brilliant performance when they were first introduced into the EJB specification in 1999. Yet the next two years revealed severe flaws in the original entity bean model. Today, the radical changes in the EJB 2.0 specification are still to be proven, and the EJB 2.1 specification is already trying to address omissions in EJB 2.0.

How is this Book Different?

First, it's an independent view, based on my experience and that of colleagues working with J2EE in production. I don't seek to evangelize. I advocate using J2EE, but caution against J2EE orthodoxy.

Second, it has a practical focus. I want to help you to implement cost-effective applications using J2EE. This book aims to demystify J2EE development. It shows how to use J2EE technologies to reduce, rather than increase, complexity. While I don't focus on any one application server, I discuss some of the issues you're likely to encounter working with real products. This book doesn't shy away from real-world problems that are not naturally addressed by the J2EE specifications. For example, how do we use the Singleton design pattern in the EJB tier? How should we do logging in the EJB tier?

This book doesn't seek to cover the whole of J2EE. It aims to demonstrate effective approaches to solving common problems. For example, it focuses on using J2EE with relational databases, as most J2EE developers face O/R mapping issues. In general, it aims to be of most help in solving the most common problems.

We'll look at a single sample application throughout the book. Rather than use an unrealistic, abstract example as we discuss each issue, we'll look at a small part of a larger, more realistic, whole. The sample application is an online ticketing application. It is designed not to illustrate particular J2EE *technologies* (like many sample applications), but common *problems* facing J2EE architects and developers.

This book is about quality, maintainability, and productivity.

This is the book I wished I'd had as I worked on my first J2EE project. It would have saved me a lot of effort, and my employer a lot of money.

My Approach

This book is *problem*-oriented rather than *specification*-oriented. Unlike many books on J2EE, it doesn't aim to cover all the many services and APIs. Instead, it recognizes that not all parts of J2EE are equally useful, or of interest to all developers, and focuses on those parts that are used in building typical solutions.

Software design is as much art as science. The richness of J2EE means that it is often possible to find more than one good solution to a problem (and many bad solutions). While I make every effort to explain my views (or prejudices), this book naturally reflects my experience of and attitude towards working with J2EE. I present an approach that I've found to work well. However, this doesn't mean that it's the *only* valid approach.

The book reflects my attitudes towards software development in general:

- I try to avoid religious positions. I've never understood the energy and passion that so many developers devote to flame wars. This benefits no one.
- I'm a pragmatist. I care about outcomes more than ideology. When I work on a project, my primary goal is to deliver a quality result on time and budget. The technology I use is a tool towards that goal, not an end in itself.
- I believe that sound OO principles should underpin J2EE development.
- I believe that maintainability is crucial to the value of any deliverable.

> *In keeping with this pragmatic approach, I'll frequently refer to the* ***Pareto Principle****, which states that a small number of causes (20%) are responsible for most (80%) of the effect. The Pareto Principle, originally drawn from economics, is highly applicable to practical software engineering, and we'll come across it repeatedly in approaching J2EE projects. For example, it can suggest that trying to solve all problems in a given area can be much harder (and less cost-effective) than solving just those that matter in most real applications.*

My approach reflects some of the lessons of **Extreme Programming (XP)**. I'm a methodology skeptic, and won't attempt to plug XP. This isn't an XP book, but I feel that XP offers a valuable balance to J2EE theory. In particular, we'll see the value of the following principles:

- Simplicity. XP practitioners advocate doing "the simplest thing that could possibly work".