Designing with EJBs, Databases, and Directory Servers

Building

Java™ Enterprise Applications

Volume 1: Architecture

O'REILLY®

Brett McLaughlin

Building Java™ Enterprise Applications Volume I: Architecture

Brett McLaughlin

Publisher: O'Reilly

First Edition March 2002

ISBN: 0-569-00123-1, 318 pages

Volume 1 of this advanced 3-volume guide explores the infrastructure issues so important to good application design. It isn't just a book about Entity Beans and JNDI. It takes you step by step through building the back end, designing the data store so that it gives you convenient access to the data your application needs; designing a directory; figuring out how to handle security and where to store security credentials you need; and so on.

Table of Contents

Preface

If you're basing your livelihood on Java these days, you are going to run across at least one enterprise application programming project; if it hasn't come upon you already, it's just around the corner. I've been faced with more than twenty at this point in my career, and see many more in my future. Each time I get into these projects, I find myself paging through book after book and searching the Web, looking for the same information time after time. Additionally, I've developed a bit of a toolkit for handling common enterprise tasks.

What I have determined is that there are many terrific books on specific technologies like Enterprise JavaBeans, servlets, and the Java Message Service. These books cover the details of these APIs and explain how to use them. I have also found, though, that there is no resource in existence that describes connecting these components in an intelligent way. No coherent examples are documented and explained that tell how best to code façade patterns, attach entity beans to directory servers, use servlets and JSP with EJB without killing performance, or a host of other common tasks. At the same time, these very issues are the heart of my job description, and probably of many other programmers' as well.

Rather than simply write a short article or two and fall short of really addressing the topic (something I see lots of people doing), I convinced O'Reilly & Associates to put forth an exhaustive series on enterprise programming in Java. I'm proud to say that you have in your hands the first volume of that series. It covers the back-end of application programming and explains databases, entity beans, session beans, the Java Message Service, JNDI, RMI, LDAP, and a whole lot more.

The topic will be extended in the next two volumes, which are already planned. The second volume will cover traditional web applications, including HTTP, HTML, servlets, JSP, and XML presentation solutions. The third volume will detail the web services paradigm, demonstrating the use of UDDI, SOAP, WSDL, and other emerging technologies.

In each volume, you will find extensive code (the code listings in this book, without comments, total well over 100 pages, about 30% of the actual book), without needless instruction or banter. I've gotten straight to the point, and tried to let you see code, not discussion of code, whenever possible. I hope that you enjoy the series, and that it aids you in your own enterprise application programming.

Organization

This book starts from the back of an enterprise application, moves from introduction into design and planning, through the database and directory server, and into the code you'll need to use this data. Here are concise descriptions of each chapter.

Chapter 1

> This chapter expands on the basic information in this Preface. It provides a blueprint for the series as well as the topics included in the chapters of this book.

Chapter 2

As suggested by the title, this chapter presents the vital planning and requirements phase of enterprise programming. It explains how decisions are made and how business needs are mapped to technical requirements, and outlines the process of taking a vague description and converting it to a technical blueprint.

Chapter 3

This chapter starts to dig into technical details. It takes the blueprints from Chapter 2 and begins to implement these in terms of data storage. You'll learn how to handle issues surrounding relational databases, write the SQL to create the data store, and develop constraints for the database. You'll also learn about directory servers and create a directory for the book's sample application.

Chapter 4

This chapter details the basics of entity beans in terms of enterprise programming. You'll create your first entity bean for the sample application, learn about IDs and sequences, and set the groundwork for the rest of the application.

Chapter 5

This chapter deals with more advanced concepts. IDs and sequences will be handled in a more generic fashion, and you'll mix session beans with entity beans, learn about information maps, and delve into more advanced CMP entity beans.

Chapter 6

This chapter introduces the manager component, explaining how data can be abstracted into Java components. Specifically, you'll write code to provide access to the directory server created earlier, and tie this component in with already-developed entity beans and databases.

Chapter 7

This chapter puts the finishing touches on the data access layer. You'll deal with threading and multiple directory server instances, as well as client applications. Finally, testing will be put in place to ensure that everything is working correctly to this point.

Chapter 8

This chapter moves from the data layer into the business layer. It further explains using the manager component, specifically with session beans. You'll also find out the best approaches to connecting your session beans to the entities and managers already in place

Chapter 9

This chapter completes the business layer with a discussion of using JMS and message-driven beans. You'll create a messaging layer in your application as well as clients that interact with it. Finally, basic packaging issues are detailed and related to the components already developed.

Chapter 10

This final chapter gives some general advice for moving beyond this first volume into web applications and web services. It also provides some practical information and resources for continuing in your application development.

Appendixes

The appendixes cover deployment of SQL scripts, installation of directory servers, application server setup and configuration, and supplemental code listings. These are chock-full of technical details that didn't easily fit into the chapters.

Software and Versions

This book covers a variety of APIs, but all fall underneath the Java 2 Enterprise Edition (J2EE) umbrella. I've used the 1.3 version of this platform, which is the "latest and greatest" available. You can download J2EE 1.3 and find out more about it online at http://java.sun.com/j2ee/.

The nature of application programming in the enterprise requires an application server on which to deploy your components. This requires a lot of vendor-specific deployment and packaging details. I've avoided these paradigms throughout the book, instead focusing on the vendor-neutral code that you will need to write. However, the appendixes at the end of this book detail deployment of various vendors' tools, specifically BEA Weblogic, the most popular large-scale application server available. This is a J2EE 1.3 application server, so you will be set with it or any other 1.3-compatible server.

The source for the examples in this book is contained completely within the book itself. Both source and binary forms of all examples (including extensive Javadoc not necessarily included in the text) are available online at http://www.newinstance.com/.

Conventions Used in This Book

I use the following font conventions in this book:

Italic is used for:

- Unix pathnames, filenames, and program names
- Internet addresses, such as domain names and URLs
- Object names and classes
- New terms where they are defined

`Constant width` is used for:

- Command lines and options that should be typed verbatim
- Names and keywords in Java programs, including method names, variable names, and class names
- XML element names and tags, attribute names, and other XML constructs that appear as they would within an XML document

`Constant width bold` is used for:

- Highlighting emphasized areas in code

EJB names are printed in roman. (An EJB name is not necessarily the name of a class or any other Java object.)

This icon signifies a note relating to the nearby text.

This icon signifies a warning relating to the nearby text.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

There is a web page for this book, where we list errata, examples, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/javentappsv1

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com/

Also visit the author's web site, http://www.newinstance.com/.

Acknowledgments

I have to think Mike Loukides and Kyle Hart, my right-hand man and woman at O'Reilly, for helping guide a very difficult book to its end. The first words of this book were actually written in November of 1999 (yes, you read that right!), so it's been a long time coming. Thanks also to Diana Reid at BEA for support and much-needed help on getting things running with BEA Weblogic.

I'd be in a heap of trouble without the support of my extended family: Gary and Shirley Greathouse, Quinn and Joni Greathouse, Larry and Judy McLaughlin, Shannon McLaughlin, and Sarah Jane Burden. Also to Laura and Laura Jordan, who made me an uncle with the addition of little Nathan (Nate to those who he drools on), who provided much-needed laughs when things got tough. I love all of you.

Of course, the biggest debt of gratitude lies with my wife. She simply makes life worth getting up for, never complains (too much) when I work long hours, and is always excited when I finish even though she has no idea what all this Java stuff is about. And, as if all that isn't enough, in June she's giving me a baby boy! If I take longer to answer mail this summer (2002), it's only because I'm learning to be a daddy with little Dean, my upcoming first child. Can you tell I'm excited? I love you Leigh and Dean (one day he'll realize this was the first time his name was in print).

Again, to the Lord who got me this far: Even so, come Lord Jesus.

Chapter 1. Introduction

Java has become a confusing world. Five years ago, there were few decisions to make once you started programming in Java—you used AWT for graphical user interfaces, sockets for network programming, and hacked together everything else you needed. Since then, though, the APIs available for the Java language have grown, and grown. . . and grown. Now you can dabble in Swing, servlets, Enterprise JavaBeans (EJB), JavaMail, and more. Additionally, there are now packages of APIs, like the Java 2 Micro Edition (J2ME) and Java 2 Enterprise Edition (J2EE). While these packages seem to be nicely wrapped bundles of useful APIs, they don't help the average developer figure out how to piece together the APIs contained in these packages. Though it's simple to find documentation on the individual APIs, getting the "big picture" is difficult, at best. One of the most interesting, but difficult, aspects of Java today is building Java enterprise applications using the J2EE package.

All of this has led the folks at O'Reilly to be interested in a book specifically focused on building enterprise applications with these APIs. Instead of small, piecemeal examples, we've found that readers want large applications built from the ground up, and explanations of design decisions. Additionally, readers have been adamant about seeing more than just the Java part of the picture; they want to know how to set up a database, and get an LDAP store running, and integrate these. How does a UDDI registry fit into the equation? I'm going to address all of these issues in this series (yes, I said series!) of books, *Building Java Enterprise Applications*. You hold Volume I in your hands.

So, this chapter is a true introduction. Not only will it introduce you to what I'll be covering in this book and the materials you'll need to follow along, but it will also tell you how this series is going to be put together. You'll see what's coming in Volumes II and III, how the examples are structured, and what topics will be covered in this book as well as future ones.

I'm glad you're willing to come along with me as we try something new. And, with that, let's get down to the details of building enterprise applications.

1.1 Building Java Enterprise Applications

From the first page of the first chapter to the last page of the last index, this series is going to focus on *building* applications. That probably sounds redundant, since you picked up this book knowing the title, but let me explain what I mean. First, I'm not going to explain the basics of the technologies used in this book. If you don't know what an entity bean is, or haven't ever written a SQL statement, or want to learn about JSPs, this book isn't for you. I'd recommend you pick up a copy of the O'Reilly book on the subject you want to learn about, and start there. Section 1.3.3 at the end of this chapter is a good reference for linking a subject to the right O'Reilly book.

Second, this book is aimed squarely at the enterprise developer, and especially at someone who has an existing or upcoming project that uses all or part of the J2EE platform. I'll explain later what constitutes an enterprise application, but this book will be most helpful if you have some real business problems to solve and can apply the concepts in these chapters directly to them.

Third, I expect you to be comfortable with (and hopefully, desirous of) lots of code. I'm going to try to keep explanations to the bare minimum on basic concepts, and instead focus on

tougher problems, real-world issues that aren't covered in other books, and typical mistakes I see in day-to-day programming. If you're not ready to wade through a lot of code (thousands of lines in this volume, for starters), you might want to set this down and pick it up again once you've had a little more experience (as if any real programmer would put something down because it's over their head!).

Fourth, this book focuses on writing applications from the ground up, using only Java as the programming language. While many enterprise applications do have to deal with legacy code or non-Java system integration, that is a topic well unto itself. To keep things clear and concise, this book deals with systems that are entirely based on the Java programming language. While the third volume on web services will certainly touch on integration with other languages, this architecture volume does not.

And finally, I'm hoping that you're willing to work through this volume, and even the rest of the series, chapter by chapter, example by example. I'll be taking you through the building of a non-trivial application in this book, and continue on with that example over the next two volumes. Although all the code covered will be available online, I've presented things in a manner that assumes you're going through the code with me. So even if you don't usually do this sort of thing, you might want to try it for this book, as it will really help you out. Also, the next two volumes will assume that you've got the code from this book working, as we'll be building on top of that infrastructure. To help you see how things will fit together, let's now walk through the three volumes that will make up this series.

Lest any of you go to the bank on the description of the series presented here, I should warn you that as with all plans, things may change. Additionally, the folks at O'Reilly have had lots of discussion about whether to first put out a volume on traditional web applications (servlets, JSP, HTML) or on web services (SOAP, UDDI, WSDL). So, if you've got an opinion, let us know! There are details on getting in touch with us in the Preface of this book, and I look forward to hearing your thoughts.

1.1.1 Volume I: Architecture

This first volume focuses on application architecture and serves as the foundation for the next two volumes. I'll dive a little further into the specifics of what this book covers in the next section.

Any enterprise application has two baseline components: design and data stores. The first of these components, design, turns out to be more about concepts and theory than about actual programming. In fact, most developers rush right through this step because they want to get to coding, and almost inevitably end up paying a price for that haste later on. In light of that, this book pays a lot of attention to design decisions involved in enterprise applications. Additionally, it lays out the process flow for database interaction, and sets up connectors for allowing our later work with web applications and web services to interact with the infrastructure set up in this book.

Additionally, this volume will spend a lot of time detailing how to develop data stores for use in these applications. Obviously, this involves databases, from designing tables and columns

to dealing with database sequences and triggers. Since each database has its own unique features, appendixes are included to offer advice on vendor-specific variations in SQL and on how to optimize your code for specific databases. Additionally, I'll spend a good bit of time delving into directory services and explaining how authentication data should be handled differently from application data. This will set the stage for the EJBs discussed in this book, which are also used heavily in the second and third volumes.

1.1.2 Volume II: Web Applications

The second volume in the series will continue where Volume I leaves off, adding a web application front-end to the architecture designed in the first book. In this volume, *web application* means using J2EE technologies (servlets, JDBC, JSPs) and HTML to construct an HTTP-accessible application front-end. In addition to explaining how these APIs fit together, this volume will also connect these front-end components to the back-ends created in Volume I. RMI, EJBs, JDBC, and more will be explained in light of the web application.

I'll also explain how various XML-based solutions like XSL and XML transformations can provide alternatives to HTML user interfaces. Although not completely integrated into the J2EE platform, XML and related technologies are becoming a vital part of any large-scale application, especially one that serves both static and dynamic content. I'll also look at XML data binding, RSS, and other means of communicating content between application front-ends.

Finally, some of the satellite components of J2EE, such as JavaMail, will be explained and discussed in relation to a functioning web application. While not critical for typical applications, these APIs can be immensely helpful in implementing an additional layer of communication between your applications and the end user. By the end of this volume, you'll not only have a complete understanding of web applications, but you'll have built a front-to-back practical solution (using the example code of Volumes I and II).

1.1.3 Volume III: Web Services

The third volume in this series will focus specifically on web services. It takes the business components discussed in Volume I (EJBs and other Java classes) and explains how they can be converted into web services using technologies such as SOAP and WSDL. Issues related to security, communication, and service registration will be explored. This is presented as a contrast to the web application interface discussed in Volume II.

This volume will also discuss the considerations involved with transmitting data across a network. Custom data types, large amounts of information, and object serialization are all important considerations, and will be given detailed coverage. You'll also learn how UDDI registries and WSDL are important not only in allowing component access, but also in restricting that access to only those methods you want to expose. Finally, exposing EJBs will be covered in detail.

1.2 Architecture

Now that you have a good idea of how the volumes in this series progress, I want to focus on what will be covered in this book. This description follows the flow of the book itself, and lets

you know where to turn if you're looking for something specific. I'll also give you a little more detail here than what is in the Preface.

1.2.1 Databases

After walking through some design issues, the first technical topic in this book is that of databases. Although almost every Java developer working on enterprise applications has used a database, very few are competent database developers. In other words, programmers know how to create rows and columns, but have very little understanding of the best way to tune tables, of how to perform database normalization, or of making a database work in an efficient, useful way.

In the chapters on database design and setup, I'll show how to create a database structure via the Structured Query Language (SQL). More importantly, I'll focus on how to set up a good relational structure and examine how EJBs need to access the data. This discussion should allow you to move from using a database to mastering one, at least in the context of enterprise applications. Discussions will be applicable to any database vendor.

1.2.2 Directory Servers

While traditional relational databases are still the prevalent force for data storage in enterprise applications, alternative data mediums are becoming popular. XML-based databases and object-oriented databases are in direct competition with relational databases, and directory servers offer a complementary solution to existing databases. For data that is read far more often than it is written, directory servers excel in performance. Examples of this sort of data are authentication credentials, such as usernames and passwords, which tend to be performance-driven. In other words, the less time a user waits to log in, the better your application is perceived.

This book takes an extensive look at directory servers in order to show you how to develop systems that integrate multiple types of data stores. I'll explain how to set up the directory store schema (which is analogous to the tables and columns of a relational database) and how to populate the directory store. I'll also show you how the Java Naming and Directory Interface (JNDI) can provide fast access to a directory server. Finally, I'll cover the tricky issues that surround using multiple data stores: replication, data overlap, and keeping data in sync and uncorrupted.

1.2.3 Enterprise JavaBeans

Once you've got a data store (actually, a couple of them) in place, I'll finally move on to Java, and accessing that data through Java. In addition to the JNDI access for directory servers, you'll learn how to use Enterprise JavaBeans (EJB) to interact with a database. I'll cover setting up your EJB container, writing entity beans for data access, and using session beans to provide a layer between your entity beans and the rest of your application. Finally, I'll detail how message-driven beans can allow communication between components that was almost impossible in earlier versions of the EJB specification.

Of course, we'll quickly move beyond these basics. I'll demonstrate the impact that EJB 2.0 has on your enterprise applications, and cover more complex issues such as using database sequences, direct access to entity beans, and how the container affects your EJB design. I'll

also detail the ins and outs of Remote Method Invocation (RMI) and how to make it perform at its best. Several chapters will be devoted to the EJB layer, so you'll have plenty of Java code to sink your teeth into: entity beans, session beans, and message-driven beans will all be explored in relation to the enterprise application.

1.3 What You'll Need

Before getting into the thick of things, let's take a moment to cover what you'll need to work through this book. Most crucial are the APIs involved, but also important are the application server, the tools I'll refer to, and all the support facilities for writing enterprise applications. You'll also probably have your own set of tools (code editors, HTML editors, etc.), and you should not have too much trouble adapting to any of the instructions for specific products that you use.

1.3.1 APIs

First and foremost, this book is focused on the 1.3 version of the J2EE specification. You can download the J2EE specification from Sun online at http://java.sun.com/j2ee. I also highly recommend that you download the J2EE SDK (essentially the reference implementation), which can be used for running the example code.

Let me say a word about application servers. There are as many application server vendors as there are colors, and picking one isn't always a trivial task. Additionally, trying to cover the nuances of each application server in a single book is simply impossible; you'll always find a vendor or version that doesn't fit the instructions, and in those cases a book's instructions can cause confusion instead of resolving it. To keep this to a minimum, I've taken two steps. First, the content in the chapters of this book is focused on APIs, code, and deployment descriptors, and will work on any J2EE 1.3 application server. In other words, the chapters are all vendor-neutral. However, this leaves a lot of vendor-specific detail up in the air, as most application servers have specific instructions for setup and deployment. To accommodate this, the appendixes in this book will show you how to get the examples to work using the BEA Weblogic application server.

If you work in an environment where another application server is in use, you can take your applications and deploy them to that application server, using the specific vendor's documentation. The result is an application that is as portable as it can be in today's world of too-many variations on the J2EE theme. Additionally, as demand and time dictate, instructions for working with other popular application servers will be posted online at this book's web site, http://www.newinstance.com/. I'm going to handle this process much like an open source project, so if you go online and don't see your vendor covered, I welcome your help and will work with you to get instructions online for your application server. Hopefully, this will be the best compromise between getting you timely and accurate information, and not creating confusion throughout the book's text.

There is also specific software needed for chapters that go beyond Java; for example, you'll need a directory server for the LDAP chapters and a database for the data store chapters. I'll discuss specific alternatives in those chapters and explain what factors can influence your choices in these areas. I try to always recommend (at a minimum) an open source option and a popular commercial alternative. More often than not, one of these will result in a good match for your needs.

1.3.2 Tools and Utilities

I also recommend a few tools and utilities for this book. While you can certainly make your own choices here, I'll let you know what has worked for me. First, you'll want a Java Integrated Development Environment (IDE). While I often use wordpad, *vi*, or Emacs for editing code, large projects demand keeping up with three, four, or more active files. It's here that an IDE can really help out. I prefer jEdit, available for free at http://www.jedit.org/. There are tons of helpful plug-ins, Java syntax highlighting is included, and it has good support with new versions coming out fast and often.

I also recommend that you have a tool for working with databases that allows fast SQL querying. Here, I am fond of a commercial tool, SQL Navigator, which is available for purchase at http://www.quest.com/sql_navigator/. This tool allows interactive querying, a nice interface for setting up your database schema, and a lot more. It's also particularly useful when dealing with Oracle, its preferred database, as it allows you to use PL/SQL, triggers, and other features specific to Oracle. Outside of SQL Navigator, there are many other free tools available for working with databases.

Finally, quite a bit of XML will be in play throughout the EJB chapters. It's needed to write deployment descriptors, and I'll also examine using XML for properties and configuration information. Additionally, many application servers add vendor-specific XML descriptors that you'll need at deployment time. I recommend an XML editor to make validation of these files easy. While you can (as I did until recently) write some command-line tools using an XML parser to handle this task, I again have recently taken up using an IDE. jEdit works well here, and I have also had some success with XMLSpy, available at http://www.xmlspy.com/. All these tools are optional, and I won't dwell on them in the text, but they can really increase productivity and make life a little easier.

1.3.3 Related Works

In addition to everything I've said so far, I'm a big advocate of books as an aid in learning and programming. A famous preacher, Lester Roloff, once said, "The best memory is the pencil." I tend to agree, as I'm constantly making notes about this method or that class, trying to remember what they do. However, there are a lot of books already written with these notes categorized, indexed, and explained in detail, so I'll provide you a short list of books that may be helpful as you work through this volume.

Generally, these are books on the technologies that are detailed in this work, and will help you get up to speed on the basics of these technologies. Many times, I assume you have knowledge of the topics in these books, and they are all worthwhile additions to your library.

- *Enterprise JavaBeans*, by Richard Monson-Haefel
- *Database Programming with JDBC and Java*, by George Reese
- *Java Enterprise in a Nutshell*, by David Flanagan, Jim Farley, William Crawford, and Kris Magnusson
- *Java Message Service*, by Richard Monson-Haefel and David Chappell
- *MySQL and mSQL*, by Randy Jay Yarger, George Reese, and Tim King
- *Oracle Design*, by Dave Ensor and Ian Stevenson

All of these are published by O'Reilly. Obviously there are many other helpful books out there, but these should get you started. Armed with this information, you're ready to move beyond introduction into the world of enterprise application programming.

Chapter 2. Blueprints

Let's begin to delve into enterprise applications. With some basic knowledge of the Java APIs and related technologies (such as XML) that are involved with these applications, you are as qualified as the next programmer to start building applications! This is a new frontier, even though it's been three or four years since the J2EE specification was released. That may sound a bit far-fetched, but technology is moving at an incredible rate, as are the APIs that support it. Just two years ago, applications had far fewer tools, technologies, and specifications upon which to build. For these reasons, you start with most other programmers on a generally level playing field. And as each phase of building an application is addressed, I discuss the principles that will guide you in your own applications, using any combination of APIs and tools.

However, discussing these complex applications in the abstract is like talking about music (which is like dancing about architecture, according to Miles Davis). In other words, trying to describe how to build an application without in fact building one is nearly impossible. For that reason, this entire series discusses the Java APIs and code within the context of a large, enterprise application that will be accessible through a web interface (in Volume II) and as a web service (in Volume III). Starting in this chapter, I will detail a fictional company, Forethought Brokerage, and discuss the application they are building (or rather, that *you* are building for them). Beginning with only a set of requirements, you will construct the Forethought application from the ground up, including data storage, API selection, and of course implementation. At the end of the series, the application will finally be ready to run, complete with several advanced features that are usable in your own applications. In this first volume, you'll build a data store that includes a database, a directory server, and numerous Enterprise JavaBeans.

This chapter begins the process by presenting a set of requirements. I will take these requirements and design blueprints for the application, "roughing out" each portion of the application and explaining each decision made. With this set of blueprints in hand, it's possible to detail each section of the application. Additionally, in a commercial environment, multiple teams could work on different portions of the application in parallel; this is possible only with a well-designed set of plans for the application, agreed upon before development begins. Although you are the only developer working on the example application, following this practice will teach you how to design your own applications so that multiple programmers can work on them. Once a general set of requirements is laid out and met, I'll run through a brief survey of the key technologies used throughout the rest of this book. If you are familiar with databases, directory servers, enterprise Java, and XML, you can probably skim these later sections of the chapter. However, if you're new to enterprise programming, these descriptions will help prepare you for the chapters that follow.

I will also go beyond just the data and business layers, which this book focuses on, and describe the presentation layer of the application. This will apply to the web interface detailed in Volume II, but will also give you perspective on how things fit together, and provide you with a good idea of how to proceed if you don't want to wait for the next volume in the series.

2.1 Forethought Brokerage

Like any good building, an application begins with a set of requirements, often having little to do with implementation details. The first challenge of constructing an application, then, is to

translate these requirements into technological outlines and a plan for action. While this can be simple when the person or group defining these requirements is technical, it is far more often the case that this mapping of requirements to an application blueprint is the most difficult portion of architecting an application. A marketing or product management group explaining their needs rarely has an idea of what is technically feasible, or even possible, with today's programming languages. Additionally, these initial requirements have a way of changing during a project, resulting in a moving target for completing a "successful" application. In fact, this is the first lesson in building an application: an application that meets the initial set of requirements is not automatically a success! Instead, it must anticipate the changing of requirements and be able to adapt in kind. For this reason, a flexible architecture and well-designed set of blueprints can lead to customer sign-off, protecting the application designer from these changes, or at least providing a reasonable window of change when requirements do evolve. This is the kind of architecture and blueprint that must be developed for Forethought Brokerage.[1]

2.1.1 The Company

Forethought Brokerage has been serving their clients in a traditional investment brokerage sense for nearly 20 years. Specializing in long-term clients and customer service, the brokerage has until recently run their entire operation largely through a paper office, using carbon receipts, conference calls, and face-to-face meetings. They have monitored their clients' funds through frequent phone calls, monitoring the market, and by sweat and hard work. Although this has succeeded in building their client base and keeping them in business for almost 20 years, it has also caused some problems. They have had to remain a locally based business, as they have no facility to handle clients around the country and the world, and their established pattern of personal consultations begins to break down over distance. Additionally, monitoring funds in 24 time zones instead of one is a significant increase in workload.

Forethought has also had a longtime fear of problems related to the paper trail on which their office relies. Even though Forethought has an offsite storage location, searches through paperwork and misfiled receipts have caused many a late night for partners and immeasurable stress for clients. The company realizes that the computer age has taken over in business, and wants to move into an electronic form of communication and storage. This would also enable them to establish additional offices and expand geographically while using one unified computer system for their records, and would allow clients to access their profiles and investments online, as many have requested. With this recognition of their problems, Forethought has begun to define the desired functionality of the application they want built.

2.1.2 Identified Needs

Forethought's product management and marketing groups (usually made up of the proverbial "pointy-haired bosses") have determined their company's needs and are ready to supply these needs to you, the lead architect and developer of their new application. These needs must be met for the application to be any sort of success, and many of them must be mapped from a business requirement to a technical one. Let's take a look at what the application requires.

---

[1] Forethought Brokerage is a completely fictional company, and any resemblance to an existing or future company is purely accidental and unintentional.

2.1.2.1 Online accessibility

First and foremost, Forethought wants to move to a web-based solution for their clients and employees (they won't make it to web services until Volume III). As offices open in new locations, these offices should be able to operate within the Forethought system through simple Internet access. Additionally, online accessibility means that agents away from the office can still access their clients' investments. Forethought also wants a means of securing access to the application, through a username and password, at a minimum. They also would like any other security appropriate for the privileged information that clients and brokers would view online.

Forethought has also determined that using a simple web browser should be sufficient to access the application. This enables easy rollout of the application, and avoids any costs of delivering disks or CDs with special software to clients wanting to access their accounts online. Since any user with a PC can be expected to have a web browser on their computer, Internet access to the application through a browser is ideal.

2.1.2.2 Supports wireless devices

As the company expands, agents need to travel more, both between offices and to clients that are geographically dispersed. With this travel comes a need to communicate, which of course is most common through mobile phones. Forethought wants to take advantage of today's wireless phones, and use Internet access as a means of supplying remote agents information about their clients quickly. An agent on the road should be able to use a WAP or HDML phone to connect to the Forethought web application and quickly gain basic information about a client and his or her accounts.

The same thinking applies to employees with Palm Pilot or other handheld devices. Delivering content to these Internet-capable devices should be possible with the application. Of course, like any company, Forethought wants to keep maintenance costs as low as possible, while still providing content to these varying devices. In other words, reusability of the application's content is important.

2.1.2.3 Handles scheduling

Scheduling is also an important aspect of Forethought's needs. As mentioned, employees (particularly brokers) will be traveling, so the tracking of meetings, appointments, and events will be critical to the company's success. Without the ability to determine where a broker must be at what time, that broker is useless to the company. Additionally, brokers shouldn't be tied to their calendar applications on a specific desktop computer. Laptops, multiple desktops, and wireless devices should all be able to access the same shared schedule, allowing the broker to check and maintain his schedule from anywhere he can access the Forethought application via the Internet.

2.1.2.4 Stores information about employees as well as clients

In addition to providing clients online access to their accounts, the application should also be able to serve Forethought as an information and intranet service. In other words, referencing details about other brokers could help an agent give referrals to clients who are moving, and could also help management monitor employees across the country and the world. This sort of

dual-purpose application, where both clients and internal workers use the provided services, is becoming more and more common.

2.1.2.5 "Fast performance and standards compliance"

All too often, marketing and management groups toss a statement like "standards-based" or "performance-driven" into an application's requirements. It would seem that these types of statements are meant to appease the technical nature of the developers working on the application, but in fact these statements are nebulous at best, and often entirely useless. How fast must the application respond? Is a World Wide Web Consortium (W3C) recommendation a standard? Are de facto standards like the Simple API for XML (SAX) standard? What sort of benchmarks should be performed? All of these questions are left ambiguously defined with vague requirements like "fast performance and standards compliance."

While the knowledge of these issues by the marketing and product groups is often as indeterminate as the questions themselves, the points are worth noting. Choosing a technology or solution that is not supportable or that may be antiquated in several years will leave you having to explain your bad decisions to upper management. While this is not a blanket recommendation to accept all standards hook, line, and sinker, it is a good idea to justify all decisions made. As an example, using large portions of the J2EE specification makes a lot of sense, as Sun will certainly support Java and the J2EE platform for many years to come. However, if a part of this or any other specification doesn't stand up to the task it's designed for, other solutions should be examined, even if they are not "as standard." Just make sure you justify straying from the well-trodden path. At the end of the day, or week, or project, you are accountable for your decisions. Ensure that you can explain all of them.

2.1.3 Proposed Solutions

With the requirements set out by Forethought, it is time to look at solving each problem, and then turning the solutions into a single coherent application. However, the last task (creating one logical application design) is often much harder than solving the individual problems that requirements pose. While a database may be a better means of storing data in one case, a directory server may be more appropriate in another. Combining both data sources is the complex problem. The same goes for handling multiple presentation layers created from similar content. You should look at solving the specific problems first, and then design an overall application to incorporate the various solutions you decide upon.

You will also need to determine which technologies and APIs should be used. As this is a book on Java, the decisions recommended shouldn't come as a great surprise—of course we will use Java! However, in a company not already sold on Java, you would need to justify this decision just as you would justify using XML, or servlets, or EJB. If you aren't sure about Java or need an introductory text comparing the language with other popular programming languages, check out *Exploring Java*, by Patrick Niemeyer, or *Java in a Nutshell*, by David Flanagan (both from O'Reilly).

2.1.3.1 Java and J2EE for web delivery

Java has arguably become the language of choice for network programming. While Perl, PHP, and Python are becoming more common, Java still has a strong, solidified position in the

enterprise application space. The language is also certainly more web-oriented than C, C++, or Microsoft's C#.

With the release of the Java 2 Enterprise Edition (J2EE), Sun gave many programmers a major missing piece of the Java puzzle: a guideline for developing enterprise applications. More than just a collection of APIs, the J2EE platform also comes with the Application Programming Model (APM), which specifies how applications should be built and pieces together many APIs that puzzled many developers for years. While parts of the APM are questionable or appear to be unfinished, the net effect is that more programmers than ever before have embraced Java, admitting that it has officially "matured." For these reasons, we will use the J2EE APIs and the APM as a starting point for our application. The J2EE APIs include servlets, JSP, EJB, and JMS. While we may use only parts of some of these APIs and discard others altogether, starting with this proven model allows us to deliver Forethought's application online to their clients and other employees, the first requirement of the application.

2.1.3.2 JSP, XML, and XSL for content and presentation

In addition to online accessibility, Forethought wanted to be able to deliver their new application to wireless devices, and particularly to brokers in the field. The decision to make here is how to separate content from presentation, and try to reuse the content with different presentations. Before going further, I should define my terms a little more concretely. *Content* is the business data that is viewed by an application client (a wireless phone, handheld organizer, web browser, and others). The key phrase in this definition is "business data": this is typically the balance of an account, a user's personal information, or an employee's scheduled meetings. However, this is raw data, without markup. In other words, the content of a page might consist of this information:

```
Brett Hund
Broker
2545550289
Waco
1212 City River Drive
Waco, Texas
76712
```

This is an entry for a broker. It contains the broker's name and title on the first two lines, his phone number on the third line, office on the fourth, and office address on the final three lines. In any Java enterprise application, servlets are usually the best choice for obtaining content from a data store or set of business components and then handing that content off to a presentation technology. Servlets are covered in greater detail in Volume II on web applications.

You'll also want to turn that content into *presentation*. Presentation here refers to a formatted HTML page, a WML deck for a wireless phone, or any other formatted data suitable for display to a user. Typically, using JSP, servlets, and now XML and XSL are all excellent choices for turning simple content into a fancier presentation. I'll focus on these concepts in Volume II as well.

2.1.3.3 Services architecture

With Forethought's requirement for scheduling, you will have to do your first bit of true creative work. Java, J2EE, XML, and XSL all require programming, but they are built upon proven concepts. However, there are no stock APIs for handling scheduling, and a set of tools and utilities for this aspect of the application will have to be created from scratch. This quickly turns into a complex problem: consider business logic that sets up introductory meetings with new clients, but must assure that a potential broker has no existing meetings already set up at the desired time.

Additionally, building the scheduling component into a more robust *services framework* can really pay off over the long term. By services framework, I mean a system that allows any component that corresponds to a specific set of guidelines (such as a particular Java interface) to be integrated into an application. Although this is not a book on component development and won't go too heavily into this framework, some groundwork to generalize how components interact with applications will be detailed and coded. Years down the line, all your components will mesh into a common system instead of existing piecemeal throughout the application. The business logic to handle appointments and client interaction is covered in Chapter 9, and scheduling and services are covered in Chapter 10.

2.1.3.4 Storing data

The requirement of storing information about clients and employees is one of the simplest to handle. It requires a decision about the medium for data storage. The two most prevalent options are relational databases (RDBMS) and directory servers (often using the Lightweight Directory Access Protocol, or LDAP). Although there are other options, such as object-oriented database management systems (OODBMS), they are not as well-accepted or proven technologies, and therefore are not the best solution for Forethought's traditional data needs.

In the case of the Forethought application, you don't necessarily have to choose one or the other—in fact, using both a database and a directory server makes a lot of sense. Pure data storage, such as handling employees' and clients' personal information, is definitely in the realm of the database. This sort of information, used often in both read (view) and write (update) operations, is best suited for storage in an RDBMS, which is optimized for general access. However, Forethought also needs security for their application, through a username and password combination. This information is read far more often than it is written, and authentication is typically expected to be a fast operation. The reasons that make a database ideal for general information make it poor for authentication data: its stability for writing results in slower reading. Here, a directory server tuned for fast searches and frequent reads is a perfect fit. Therefore, a combination approach is well suited in this case, and solves the problem of data storage for the application. Databases and directory servers are covered in Chapter 3.

As for accessing this data, proven solutions exist, all within the J2EE programming model. Enterprise JavaBeans (EJB) is perfect for database access and is covered in Chapter 4 and Chapter 5. Directory servers can be accessed most easily through the Java Naming and Directory Interface (JNDI). Usable within any Java code, beans could also be written to provide directory server access; however, for reasons discussed in Chapter 6, it is often better to use normal Java classes for this facet of an application.

2.1.3.5 Servlets, EJB, caching, and performance

The last requirement discussed, that of "fast performance and standards compliance," is a bit vague. However, all the solutions discussed so far are based on these very premises: Java, J2EE, XML, XSL, and all the rest are accepted standards. And using a directory server for authentication was a decision made for performance reasons. In other words, good design decisions generally involve these principles, even without marketing or product management reminding you of them. I'll also detail using caching and design patterns that can improve performance as we go along.

At this point, the individual requirements of the application have all been addressed, and you now face the difficult task of integrating these solutions into a larger, complete system. I've tried to discuss each component separately from those around it: the data storage is dealt with separately from the services and data access, which is separated from the business logic, which is then separated from content, which is in turn separated from presentation. Each portion of the application operates in isolation and supplies data to the next layer as needed. This architecture can allow easy updates and additions of functionality over time. It also makes debugging simple, since a problem can be quickly isolated to a specific application layer. With these concepts in mind, let's now look at designing our complete application, layer by layer. The rest of the book is split into sections that correspond to each of these layers.

2.2 The Data Layer

The foundation of any application is the data that it contains and utilizes. Without data, there is no need for an application! Unfortunately, designing a data layer does not produce immediate visible results—no screens appear, no business logic occurs, and management is rarely impressed with entity-relationship diagrams. Because of this, the data layer is often designed and implemented hastily, leaving the rest of the application to suffer and compensate for early mistakes. Instead of taking this precarious approach, this book covers the data layer first and attempts to design it soundly.

In addition to data storage, data access is typically part of the data layer. However, these different functions can be separated from each other; do not be tempted to model your data differently because of a product or technique used in the data access layer. More often than not, the data of an application outlasts the application itself. Data formatted for a specific product or application server may be completely unusable for other products that expect data in a standard format.[2] Only *after* the data has been modeled and the storage mediums designed should data access be considered. This section covers databases and directory servers as well as data access methods.

2.2.1 Databases

Once you decide to use a relational database, the number of decisions left decreases quite a bit. First, you must choose a database vendor. Second, the database must be accessible

[2] I emphasize this point because there are several application servers out there that require modifications to the database in order to "perform optimally"; for example, the Persistence PowerTier EJB Server used to suggest adding columns in database tables for the OCA attribute, as well as modeling data in an OO rather than relational format. These types of changes may improve application performance over the short term but almost always cause problems in the long run, and should be avoided at all costs.

through a JDBC (Java Database Connectivity) driver.[3] Then, the data design must be determined, and finally, the data schema should be populated.

Determining what database to use can be very difficult; the data of an application often outlasts other parts of an application, or becomes used by other applications, over time. This means that your database vendor (and your resulting relationship with the vendor) will play a critical role. Often, though, this decision is taken out of the developer's hands; many large companies establish a standard vendor and simply purchase new licenses for additional database instances as needed. In these cases, you will simply need to become familiar with the selected vendor. However, in the case where no standard exists, there are numerous options. Trying to recommend a single option for all cases is impossible; instead, certain conditions favor specific vendors. On large systems, and particularly on Sun Solaris hardware, Oracle is an excellent choice. Linux servers and clusters often use Oracle as well, and open source solutions like MySQL, PostgreSQL, and InstantDB are also popular. Microsoft platforms tend to work best with the Microsoft database offering, MS SQL Server. And there are many, many other vendors to choose from, such as Sybase, Cloudscape, mSQL, and Interbase. While the example code shown in this book utilizes only standard ANSI SQL and should work on any of these databases, the appendixes cover various database-specific SQL idioms.

JDBC drivers are available for each of these vendors. Additionally, the JDBC-ODBC bridge driver can be used to access databases on Windows systems if a native JDBC driver is not available, but this situation is rare. In fact, Java has become so prevalent that almost all databases supply a JDBC driver with installation, and application servers (particularly EJB servers) provide JDBC drivers as well.

As for designing the data schema, I'll leave that for the next chapter. For now it is enough to know that aside from the usernames and passwords that will be stored in an LDAP directory server, we will house all application data in a single database instance. I'm also not going to deal with replication or high availability at the database level in this book; these topics are books unto themselves, and shouldn't affect the overall application design. However, this is by no means a recommendation against employing these techniques—data reliability is an important issue, and some sort of redundancy should be built into your application, especially at the data level.

2.2.2 Directory Servers

Using a directory server is not quite as complex as using a database; while the techniques involved are less established, there are fewer options and chances for misuse. The most common problem is attempting to use a directory server as a wholesale replacement for a database. By deciding to store only authentication information in the directory server, that problem has already been avoided in the Forethought application. While there are certainly situations in which other information could be housed in a directory server, keeping things simple is usually a good idea, and this is what is being done for Forethought.

Since LDAP services were first created at the Universities of Michigan and Berkeley, there has been a well-established set of standard object types and structures defined for use.[4]

[3] While these are portrayed as two independent decisions, it is possible that the second decision can affect the first. If you are writing a Java application and your database has no stable JDBC driver, you may have to rethink the database vendor. That said, with the prevalence of Java today, this is not as large a problem as it was years ago.

[4] The longer story of directory servers is rooted in the history of X.509. However, that's pretty dry stuff, so I'm going to leave it to interested parties to research this information on their own. You can start by checking out http://www.openldap.org/.