
WebObjects Overview



January 2002

🍏 Apple Computer, Inc.
© 2000–2002 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Computer, Inc. Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Mac, and WebObjects are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Enterprise Objects and Enterprise Objects Framework are trademarks of NeXT Software, Inc., registered in the United States and other countries.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Apple Computer, Inc. is independent of Sun Microsystems, Inc. Simultaneously published in the United States and Canada

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures and Tables 7

Chapter 1 About This Book 9

Why Read This Book 9
Further Investigations 10
 Installed Developer Documentation 11
 Online Documentation 11

Chapter 2 What Is WebObjects? 13

Dynamic HTML Publishing 13
Web-Enabled Client-Server Applications 16
 HTML-Based WebObjects Applications 16
 Java Client Applications 17
Rapid Development 19
 Direct to Web 19
 Direct to Java Client 20
The WebObjects Advantage 20
 Streamlined Database Access 20
 Separation of Presentation, Logic, and Data 21
 State Management 21
 Modular Development 22
 Pure Java 22
 Scalability and Performance 22

Chapter 3 Enterprise Objects 25

What Is an Enterprise Object? 25
 Enterprise Objects and the Model-View-Controller Paradigm 28
Mapping Your Enterprise Objects to Database Tables 29

C O N T E N T S

WebObjects Support for Enterprise-Object Instances	31
The Enterprise Objects Advantage	34

Chapter 4 HTML-Based Applications 37

A Programmer's View of WebObjects	37
Separating Presentation Code from Event-Handling Logic	38
Dynamic Elements	39
Reusing Components	40
Maintaining State	41
Separating Presentation Code from Business Logic	41
The WebObjects Architecture	43
Developing a WebObjects HTML Application	44
Project Builder	45
WebObjects Builder	46
Guidelines for Choosing the HTML-Based Approach	47

Chapter 5 Direct to Web Applications 49

How Direct to Web Works	55
Developing a Direct to Web Application	58
The Direct to Web Assistant	58
Further Customizing Your Direct to Web Application	61
Advantages of the Direct to Web Approach	62
Limitations	63
Guidelines for Choosing Direct to Web	64

Chapter 6 WebObjects Desktop Applications 65

Java Client Features	66
Better User Experience	67
Object Distribution	67
The Best of WebObjects	67
Deployment Options	68
Rapid Application Development	68
When To Choose Java Client	69

C O N T E N T S

Two Approaches to Java Client	71
Choosing an Approach	74
Java Client Architecture	76
Managing the User Interface	79
Data Synchronization Between Client and Server	79
Dynamic User Interface Generation	81
Java Client and Other Three-Tier Systems	81
Development Tasks and Tools	83
Designing Enterprise Objects for Java Client	83
Creating the User Interface (Non-Direct)	84
Customizing the User Interface (Direct Approach)	85

Chapter 7 J2EE Integration 87

Enterprise JavaBeans	88
JSP and Servlets	89
JNDI	89

Chapter 8 Choosing Your Approach 91

Internet and Intranet Deployment	91
User Interface Requirements	92
Rich Widget Selection and Fast Response Times	92
Specific Layout and Flow Requirements	92
Rapid Development Considerations	93
Combining Approaches	94
Combining HTML-based and Java Client Approaches	94
Adding Rapid Development	95
Summary	95
Where to Go From Here	96

Glossary	97
----------	----

Index	101
-------	-----

Figures and Tables

Chapter 2 What Is WebObjects? 13

Figure 2-1	A static publishing site	14
Figure 2-2	A dynamic publishing site	15
Figure 2-3	A website running Java Client applications	18
Figure 2-4	Multiple instances, multiple applications	23

Chapter 3 Enterprise Objects 25

Figure 3-1	Connecting enterprise objects to data and the user interface	27
Figure 3-2	Mapping between an enterprise object class and a single table	30
Figure 3-3	Mapping relationships	31
Figure 3-4	Implementing business logic in enterprise objects	34
Figure 3-5	Implementing business logic in the user interface application	35
Figure 3-6	Implementing business logic in the database	36

Chapter 4 HTML-Based Applications 37

Figure 4-1	The files of a WebObjects component	39
Figure 4-2	How enterprise objects relate to a WebObjects component	42
Figure 4-3	WebObjects HTML-based application communication chain	43
Figure 4-4	Project Builder	45
Figure 4-5	WebObjects Builder	46
Table 4-1	Example Dynamic Elements	40

Chapter 5 Direct to Web Applications 49

Figure 5-1	A login page	49
Figure 5-2	A query-all page	50
Figure 5-3	A query page	51
Figure 5-4	A list page	52

FIGURES AND TABLES

Figure 5-5	An inspect page	52
Figure 5-6	An edit page	53
Figure 5-7	An edit relationship page	53
Figure 5-8	The menu header	54
Figure 5-9	An example Neutral look page	54
Figure 5-10	An example Basic look page	55
Figure 5-11	Determining attributes from the entity	56
Figure 5-12	The Direct to Web rule system	57
Figure 5-13	The Direct to Web Assistant	59
Figure 5-14	The Entities pane of the Direct to Web Assistant	60
Figure 5-15	The Rule Editor	61

Chapter 6 WebObjects Desktop Applications 65

Figure 6-1	A sample Java Client application	66
	Deployment Options	68
	When To Choose Java Client	69
Figure 6-4	Java Client's distributed, multitier architecture	76
Figure 6-5	Architecture of a Java Client application	77
Figure 6-6	Architecture of a Direct to Java Client application	78
Figure 6-7	Data flow in a Java Client application	80
Figure 6-8	Composing a user interface with Interface Builder	85
Table 6-1	Comparison of Java Client and Direct to Java Client	75

About This Book

WebObjects is an application server with tools, technologies and capabilities to create Internet and intranet applications. It has an object-oriented architecture that promotes quick development of reusable components. WebObjects is extremely scalable and supports high transaction volumes.

This book introduces the architecture, technologies, development tools, and development approaches of WebObjects to developers and others interested in how WebObjects works.

Why Read This Book

WebObjects Overview is written for developers who want to start using WebObjects. However, anyone interested in WebObjects technology will benefit by reading this book.

For the most part, this book does not assume you have a background in object-oriented programming. However, WebObjects is based on object-oriented frameworks written in Java, an object-oriented language. You should be familiar with object-oriented programming if you intend to write WebObjects applications. There are many books available on the subject if you aren't.

A hallmark advantage of WebObjects is the database connectivity it provides. To fully appreciate this technology, you should have some understanding of databases (although this book doesn't require it). Again, there are many books available that address database technology.

About This Book

Because WebObjects provides four distinct approaches to developing applications, this book discusses them one by one, and compares their pros and cons to help you decide which approach is appropriate for your application.

This book has the following chapters:

- “What Is WebObjects?” (page 13) introduces the technologies of WebObjects and how they fit together.
- “Enterprise Objects” (page 25) describes the objects that lie at the heart of all WebObjects applications and encapsulate your application’s business logic and data.
- “HTML-Based Applications” (page 37) describes the approach that allows you to create HTML applications for the World Wide Web.
- “Direct to Web Applications” (page 49) describes the rapid development version of the HTML-based application approach.
- “WebObjects Desktop Applications” (page 65) describes the approach with which you can produce a graphical user interface application that runs on a client machine.
- “J2EE Integration” (page 87) explains how WebObjects implements Enterprise JavaBeans and how it integrates with JavaServer Pages (JSP) and servlets.
- “Choosing Your Approach” (page 91) summarizes the pros and cons of these approaches, and then outlines the process you should go through to decide which approach or combination of approaches is appropriate for your environment.

Further Investigations

This book serves as a starting point. It surveys the technologies of WebObjects without providing the details. This section lists sources of WebObjects information for software developers. It is by no means an exhaustive list, and Apple’s contribution to this list will grow.

Installed Developer Documentation

When you install the WebObjects Developer package on your computer, the Installer puts developer documentation into the following locations:

- **Frameworks.** Information inextricably associated with a framework is usually installed in a localized subdirectory of the framework. This method of packaging ensures that the documentation moves with the framework when it is moved (or is copied) to another location. It also makes it possible to have localized versions of the documentation (although development and deployment tools are localized for English only).
- **Development tools.** Help information for applications such as Project Builder and Interface Builder is installed with the application. When users request it from the Help menu, the application launches Help Viewer to display it.
- **Example code.** A variety of sample programs are installed in `/Developer/Examples/JavaWebObjects` (`$NEXT_ROOT/Developer/Examples/JavaWebObjects` on Windows 2000) showing you how to perform common tasks using WebObjects.
- **All information that is not specific to frameworks or development applications is installed in `/Developer/Documentation/WebObjects` (`$NEXT_ROOT/Documentation/Developer` on Windows 2000). On Mac OS X, the Installer also creates symbolic links to the framework documentation in this location.**

To access the developer documentation on Mac OS X, you open the `WOHomePage.html` file in your Web browser. This file is located in the `/Developer/Documentation/WebObjects` directory.

To access the developer documentation on Windows 2000, you use the WOInfoCenter application. To access the WOInfoCenter, go to the Start menu and select Program > WebObjects > WOInfoCenter.

Online Documentation

You can find WebObjects documentation and resources at <http://developer.apple.com/webobjects>.

What Is WebObjects?

From an information technology perspective, WebObjects is a scalable, high-availability, high-performance application server. From the viewpoint of a developer, though, WebObjects is an extensible object-oriented platform upon which you can rapidly develop and deploy Web applications that integrate existing data and systems. WebObjects is especially suited to dynamically publishing data on the Web and bringing the increased connectivity of the Web to traditional client-server and desktop applications.

The Web was created to simplify access to electronically published documents. Originally just static text pages with hyperlinks to other documents, Web pages quickly evolved into highly graphical animated presentations. Along the way, a degree of interactivity was introduced, allowing people browsing the Web to fill out forms and thereby supply data to the server.

WebObjects allows you to take the next logical step. With it, you can produce full-fledged Web-accessible applications, for use either across the Internet or within a corporate intranet. These applications can be HTML-based, and thus accessible through a Web browser, or can have the full interactivity of a stand-alone application.

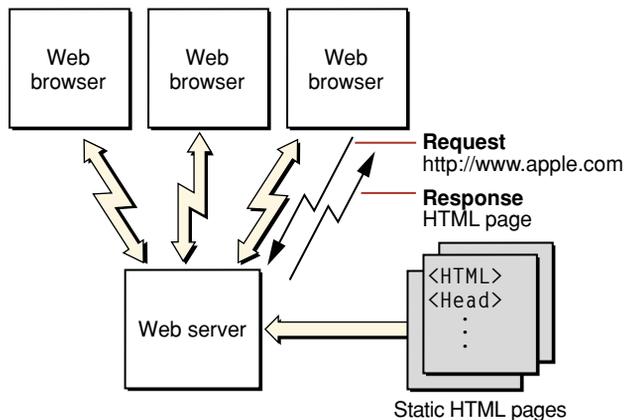
Dynamic HTML Publishing

Much of the content on the Web is textual or graphical material that doesn't change much over time. However, there is increasing demand for sites that publish ever-changing data: breaking news stories, up-to-the-minute stock quotes, or the current weather are good examples.

What Is WebObjects?

A typical website is organized like Figure 2-1. A user's Web browser requests pages using URLs (Uniform Resource Locators). These requests are sent over the network to the HTTP server, which analyzes each request and selects the appropriate Web page to return to the user's browser. This Web page is simply a text file that contains HTML code. Using the HTML tags embedded within the file received from the HTTP server, the browser renders the page.

Figure 2-1 A static publishing site



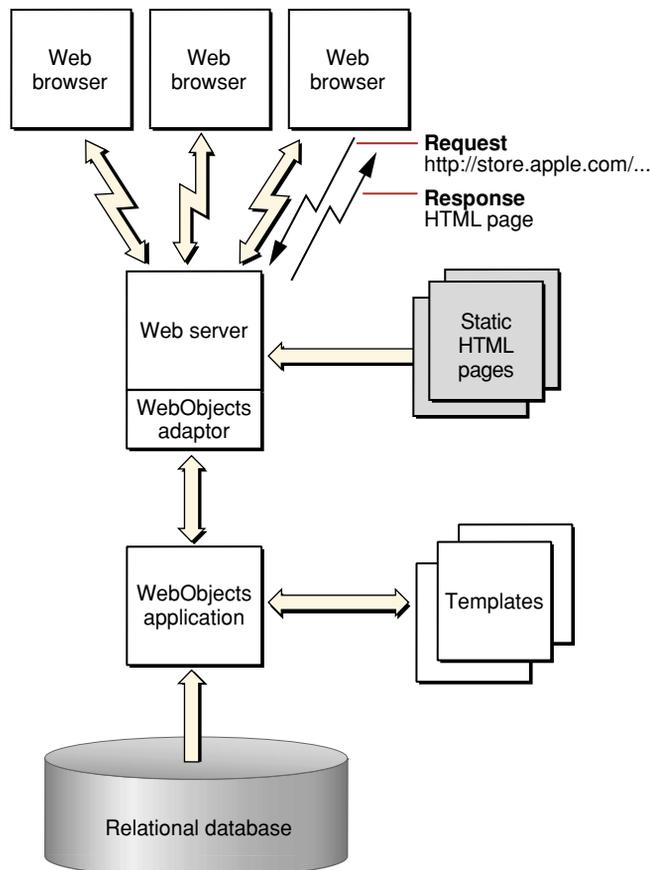
Static publishing sites are easy to maintain. There are a number of tools in the market that allow you to create HTML pages with a relatively small amount of effort. And as long as the content of your pages doesn't change too often, it isn't that difficult to keep them up-to-date. Dynamic publishing sites, however, are a different story. Without WebObjects it could take a small army to keep a breaking news site up to date.

WebObjects was designed from the beginning to allow you to quickly and easily publish dynamic data over the Web. You create HTML templates that indicate where on the Web page the dynamic data is to be placed, and a WebObjects application fills in the content when your application is accessed. The process is much like a mail merge. The information your Web pages publish can reside in a database or some other permanent data storage (files, perhaps), or it can even be calculated or generated at the time a page is accessed. The pages are also highly interactive—you can fully specify the way the user navigates through them.

What Is WebObjects?

Figure 2-2 shows a WebObjects-based dynamic publishing site. Again, the request (in the form of a URL) originates with a client browser. If the HTTP server detects that the request is to be handled by WebObjects, it passes the request to an HTTP adaptor. The adaptor packages the incoming request in a form the WebObjects application can understand and forwards it to the application. Based upon templates you've defined and the relevant data from the data store, the application generates an HTML page that it passes back through the adaptor to the Web server. The Web server sends the page to the client browser, which renders it.

Figure 2-2 A dynamic publishing site



What Is WebObjects?

This type of WebObjects application is referred to as “HTML-based,” since the result is a series of dynamically generated HTML pages.

Instead of using an HTTP adaptor, you can deploy applications through servlet containers. This approach allows you take advantage of your servlet server’s application deployment facilities. For more information on this approach, see “JSP and Servlets” (page 89).

Web-Enabled Client-Server Applications

Although the majority of websites primarily publish static data, the number of sites that publish dynamic content is growing rapidly. Many corporations use intranets, the Internet, or both to provide easy access to internal applications and data. An online store selling books, music, or even computers is one example of a Web-enabled client-server application.

Web-enabled applications can have huge advantages over traditional applications. Clients don’t have to install the application, which not only saves client disk space but ensures that the user always has the most up-to-date version of the application. As well, the client computers can be Macintosh computers, PCs, or workstations—anything that can run a Web browser with the necessary capabilities.

WebObjects allows you to develop two different flavors of Web-enabled application: HTML-based applications and Java Client applications. HTML-based applications are analogous to Common Gateway Interface (CGI) applications and consist of dynamically-generated HTML pages accessed through a Web browser. Java Client moves part of your application to the client-side computer and enlists Sun’s Java Foundation Classes (JFC) to give it the rich user interface found in a more traditional desktop application.

HTML-Based WebObjects Applications

When you need to develop an HTML-based application like a shopping cart, you can create it quickly and easily with the WebObjects development tools. WebObjects supplies a large number of prebuilt components—Web pages, or portions of Web pages, from which you can build your Web application’s interface. These components range from simple user interface widgets (for example, submit

What Is WebObjects?

buttons, checkboxes, and tables) to complex ones (for example, toolbars). The set of components that you can use with WebObjects is extensible, so you can create components that can be reused across all of your Web applications.

Your application isn't entirely built out of components. You create WebObjects applications from a combination of components and Java classes. You put your application-specific business logic in some of these classes. WebObjects provides the rest of them.

The basic structure of an HTML-based Web application matches that of a dynamic publishing site that uses WebObjects. Thus, [Figure 2-2](#) (page 15) applies to HTML-based Web applications as well.

Java Client Applications

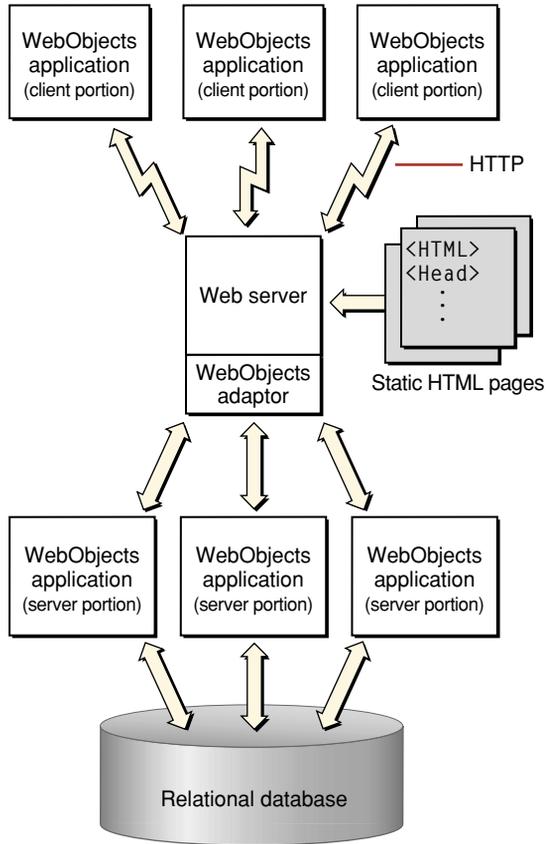
When you need the fast and rich user interface of desktop client-server applications, you can partition your application so that a portion of it—including all or part of the user interface—runs in Java directly on the client. Client-server communication is handled by WebObjects. WebObjects applications that are partitioned in this way are known as **Java Client** applications.

Java Client distributes the objects of your WebObjects application between the application server and one or more clients—typically Java applications. It is based on a distributed multi-tier client-server architecture where processing duties are divided between a client, an application server, a database server, and an HTTP server. With a Java Client application, you can partition business objects containing business logic and data into a client side and a server side. This partitioning can improve performance and at the same time help to secure legacy data and business rules.

[Figure 2-3](#) (page 18) illustrates a Java Client application in which the client portion is running as an application installed on the user's computer. As with an HTML-based WebObjects application, the application can communicate with the server side using HTTP. In addition, Java Client passes objects between a portion of your application residing on the user's computer and the portion of your application that remains on the application server.

What Is WebObjects?

Figure 2-3 A website running Java Client applications



Java Client allows your application to look and feel like a traditional desktop application and still take full advantage of the power of WebObjects.

Rapid Development

WebObjects is both powerful and flexible. With that power and flexibility, however, comes a certain degree of complexity. For many applications, whether HTML-based or Java Client-based, it's more important to develop the application quickly than strive for maximum flexibility or polish. As an example, a simple data-browsing and editing application, intended only for internal use by a system administrator, probably wouldn't warrant the same degree of effort you would put into an Internet-enabled application accessible by the general public. To simplify the development of applications like the former, WebObjects includes a set of rapid-development technologies: Direct to Web and Direct to Java Client.

Direct to Web and Direct to Java Client are similar in approach. Their primary difference is in how the application interacts with the end user. Direct to Web creates HTML-based WebObjects applications, whereas Direct to Java Client creates WebObjects applications that employ Java Client to partition the application between server and client. Both are useful not only for creating simple database browser applications, but in many situations can also serve as rapid prototyping tools. Because Direct to Web and Direct to Java Client both allow customization on various levels, they are well-suited for bootstrapping and creating your mission-critical applications.

Direct to Web

Direct to Web is a system for creating HTML-based WebObjects applications that access a database. All Direct to Web needs to create the application is a model of the database, which you can build using EOModeler (a data-modeling tool).

Direct to Web applications are not a set of static Web pages. Instead, Direct to Web uses information from the data model available at runtime to dynamically generate the pages. Consequently, you can modify your application's configuration at runtime—using the Direct to Web Assistant—to hide objects of a particular class, hide their properties, reorder properties, and change the way they are displayed without recompiling or relaunching the application.

What Is WebObjects?

Out of the box, Direct to Web generates Web pages for nine common database tasks, including querying, editing, and listing. To do this, Direct to Web uses a task-specific component called a template that can perform the task on any entity. The templates, in conjunction with a set of rules (which you can customize), are the essential elements of your Direct to Web application.

Direct to Web is highly customizable. For example, you can change the appearance of the standard templates, mix traditional HTML-based WebObjects components with Direct to Web pages, and create custom components and templates that implement specialized behavior.

Direct to Java Client

Like Direct to Web, Direct to Java Client generates a user interface for common database tasks using rules to control program flow and it has an assistant that allows you to modify your applications at runtime. The primary difference between Direct to Web and Direct to Java Client is the type of application each produces: Direct to Java Client produces Java Client applications that have the fast and rich user interfaces associated with desktop applications. Thus, Direct to Java Client applications have the same client-side requirements that other Java Client applications do.

The WebObjects Advantage

WebObjects encapsulates a number of key technologies that give it a significant advantage over other application servers.

Streamlined Database Access

Much of the data that is (or could be) presented on the Web already exists in electronic form. Not only can it be a challenge to create a website or Web application to present your data using conventional tools, it can also be a challenge just to access the data itself. Some products rely on hand- or assistant-generated SQL (Structured Query Language), leading to database-specific code that is difficult to optimize. WebObjects avoids these problems by using Enterprise Objects, a model-based

What Is WebObjects?

mechanism for cleanly instantiating business objects directly from database tables. WebObjects handles all the interactions with the database including fetching, caching, and saving. This allows you to write your business logic against actual objects independent of the underlying datasource. You can modify schemas, add or change databases, or even use totally a different storage mechanism without needing to rewrite your application.

WebObjects applications can access any database with a JDBC 2.0 driver. JDBC is an interface between Java platforms and databases.

Separation of Presentation, Logic, and Data

An ideal Web application development system simplifies maintenance and encourages code reuse by enforcing a clean separation of presentation (HTML), logic (Java), and data (SQL). This modularity is inherent in the WebObjects programming model, which uses reusable components to generate Web pages directly from enterprise objects without the need to embed scripts or Java code inside your HTML. A component contains a template, which you—or a professional Web designer—can lay out and edit using standard Web authoring tools. A component can also implement custom behavior using a separate Java source file. Neither the template nor the Java source file includes model-specific information.

State Management

The HTTP protocol used on the Web is inherently stateless; that is, each HTTP request arrives independently of earlier requests, and it is up to Web applications to recognize which ones come from an individual user or session. Therefore, most Web applications of consequence—as well as some of the more interesting dynamic publishing sites—need to keep state information, such as login information or a shopping basket, associated with each user session.

Without using cookies, WebObjects provides objects that allow you to maintain information for the life of a particular client session, or longer. This makes it particularly easy to implement an application like a Web-based online store: you don't have to do anything special to maintain the contents of the user's shopping cart or other data over the life of the session. In addition, your online store could even monitor individual customer buying patterns and then highlight items they're more likely to be interested in the next time they visit your site.

What Is WebObjects?

Modular Development

The power of WebObjects comes from a tightly integrated set of tools and frameworks, facilitating the rapid assembly of complex applications. At the heart of this system is Project Builder, an integrated development environment (IDE) that manages your Java business logic and tracks all the supporting models and components. As mentioned above, WebObjects also includes powerful assistants and frameworks that allow the rapid creation of HTML or Java Client applications directly from the database. Advanced developers can tap into the Java APIs underlying WebObjects' frameworks, allowing virtually unlimited customization and expandability.

Pure Java

WebObjects applications are 100% Pure Java, which means they can be deployed on any platform with a certified Java 2 virtual machine.

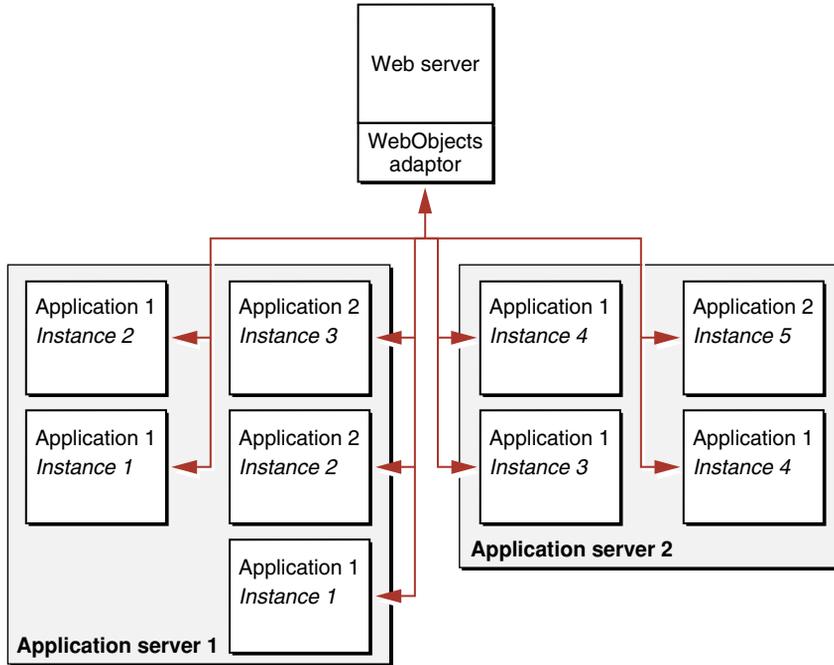
Scalability and Performance

Static websites and traditional client-server applications have one strong suit: they both leverage the power of the client platform, minimizing the load on the server. It doesn't take all that much processing power to serve up a set of static Web pages. Dynamic Web applications, although a tremendous advance over static pages, require additional server power to access the dynamic data and construct the Web pages or Java Client user interface on the fly.

The WebObjects application server is both efficient and scalable. With WebObjects, if more power, reliability, or failover protection is needed, you can run multiple instances of your application, either on one or on multiple application servers (see [Figure 2-4](#) (page 23)). You can choose from one of several load-balancing algorithms (or create your own) to determine which application instance each new user should connect to. And, either locally or from a remote location, you can analyze site loads and usage patterns and then start or stop additional application instances as necessary. Load balancing is a very powerful feature of WebObjects that allows you to add more server capacity as the need arises without needing to implement a load-balancing algorithm yourself.

What Is WebObjects?

Figure 2-4 Multiple instances, multiple applications



Enterprise Objects

As mentioned in “[What Is WebObjects?](#)” (page 13), WebObjects applications gain much of their usefulness by interacting with a persistent data store, that is, a datasource, which is usually a database. In WebObjects, database tables are represented as collections of Java classes called **enterprise objects**. Enterprise objects contain the bulk of your application’s **business logic**, the part of the application you write regardless of which of the four approaches you take.

This chapter introduces enterprise objects, describes how they map to a database, outlines how WebObjects supports and interacts with them, and enumerates the advantages of the Enterprise Objects approach over other approaches available in the industry. You may wish to read the first section, “[What Is an Enterprise Object?](#)” (page 25), and the last section, “[The Enterprise Objects Advantage](#)” (page 34), and skip the rest of the chapter, which is written for those familiar with relational databases.

What Is an Enterprise Object?

An enterprise object is like any other object in that it couples data with the methods for operating on that data. However, an enterprise object class has certain characteristics that distinguish it from other classes:

- It has properties that map to stored or persistent data; an enterprise object instance typically corresponds to a single row or record in a database table.
- It knows how to interact with other parts of WebObjects to give and receive values for its properties. This is done through a mechanism known as **key-value coding**, which enables the setting and getting of these properties.

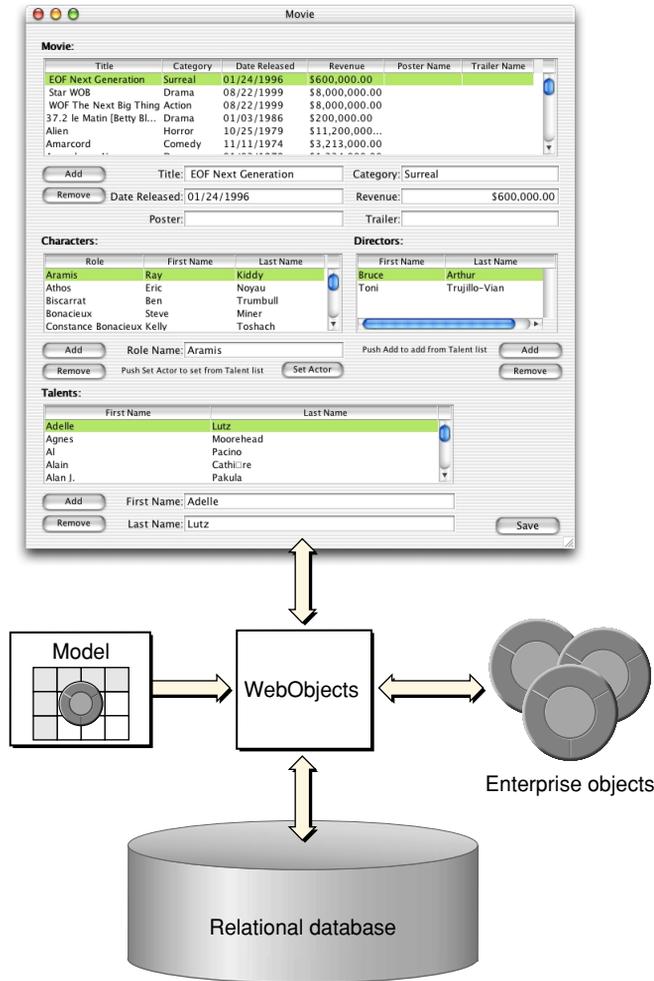
Enterprise Objects

In addition to providing classes that manage a set of enterprise-object instances in memory, WebObjects defines an API to which enterprise objects must conform, as well as default implementations for this API. As a result, you only need to concentrate on the parts of your enterprise-object classes specific to your application.

To maximize the reusability and extensibility of your objects, they shouldn't embed knowledge of the user interface or database alongside the business logic. For example, if you embed knowledge of your user interface, you can't reuse the objects because each application's user interface is different. Similarly, if you embed knowledge of your database, you'll have to update your objects every time you modify the database's schema.

If not in the business objects, then where does this knowledge go? It's handled by WebObjects as shown in [Figure 3-1](#) (page 27).

Figure 3-1 Connecting enterprise objects to data and the user interface



WebObjects provides a database-to-objects mapping, called a **model**, so your objects are independent of the database. WebObjects also provides an objects-to-interface mapping so they are independent of the user interface. This approach allows you to create libraries of enterprise objects that can be used in as

Enterprise Objects

many applications as you need, with any user interface, and with any database server. Therefore, you can concentrate on coding the logic of your business while WebObjects takes care of the rest.

For example, you could create an enterprise object called Customer that defines such business rules as *customers must have a work or home phone number*, or *customers cannot spend more than their credit limit*. Without rewriting your business logic, you could use these objects in a public, Web-based application and an internal customer service application. You could also switch the database that serves the customer data.

Enterprise Objects and the Model-View-Controller Paradigm

A common and useful paradigm for object-oriented applications, particularly business applications, is Model-View-Controller (MVC). Derived from Smalltalk-80, MVC proposes three types of objects in an application, separated by abstract boundaries, and communicating with each other across those boundaries.

Model objects represent special knowledge and expertise. For example, model objects can hold a company's data and define the logic that manipulates that data. Model objects are not directly displayed. They often are reusable, distributed, persistent, and portable to a variety of platforms.

Note: WebObjects uses the term *model* differently from MVC. In WebObjects, a model establishes and maintains a correspondence between an enterprise object class and data stored in a relational database. In MVC, model objects represent the special knowledge of the application.

View objects represent things visible on the user interface (for example, windows, buttons, and so on). A view object is ignorant of the data it displays. View objects in general are reusable, which helps in providing consistency between applications.

Acting as a mediator between model and view objects in an application is the controller object. There is usually one per application or window. A controller object communicates data back and forth between the model objects and view objects. Since what a controller does is very specific to an application, it is generally not reusable even though it often constitutes much of an application's code.

Enterprise Objects

Because of the controller's central mediating role, model objects need not know about the state and events of the user interface, and view objects need not know about the programmatic interfaces of model objects.

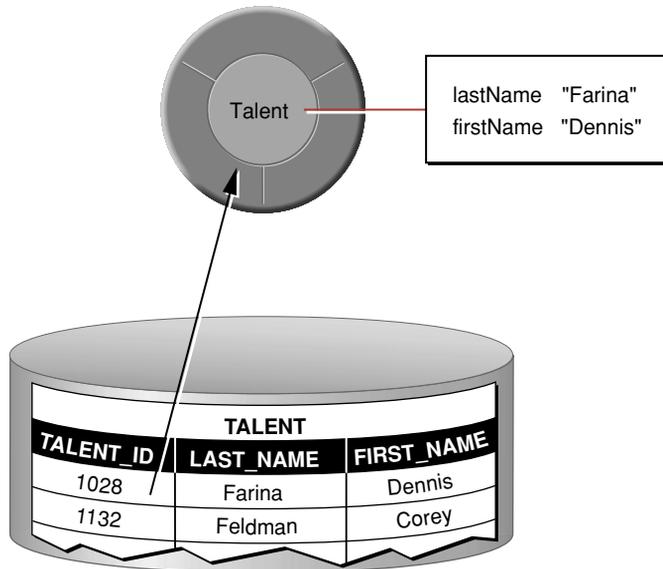
From the perspective of this paradigm, enterprise objects are model objects. However, WebObjects also extends the MVC paradigm. Enterprise objects are also independent of their persistent storage mechanism. Enterprise objects do not need to know about the database that holds their data, and the database doesn't need to know about the enterprise objects.

Mapping Your Enterprise Objects to Database Tables

Enterprise objects make use of a separate file, known as a model, to specify a mapping between tables in the database and your enterprise-object classes. This is formally called an **entity-relationship** (E-R) model. You use EOModeler to create and maintain these models. With EOModeler you can

- read the data dictionary from a database to create a default model, which can then be tailored to suit the needs of your application
- define data entities that represent the tables in your database
- define the attributes of each entity; these attributes usually correspond to columns on a table
- specify relationships between entities and referential integrity rules for these relationships
- generate source code files (enterprise-object classes) for the entities you specify
- define fetch specifications (queries) that you can invoke by name in your applications
- create, modify, or delete tables or databases

A model represents a level of abstraction above the database. The database-to-objects mapping embodied in a model sets up a correspondence between database tables and the model's entities; frequently, table rows map to instances of the appropriate data entity, as shown in [Figure 3-2](#) (page 30).

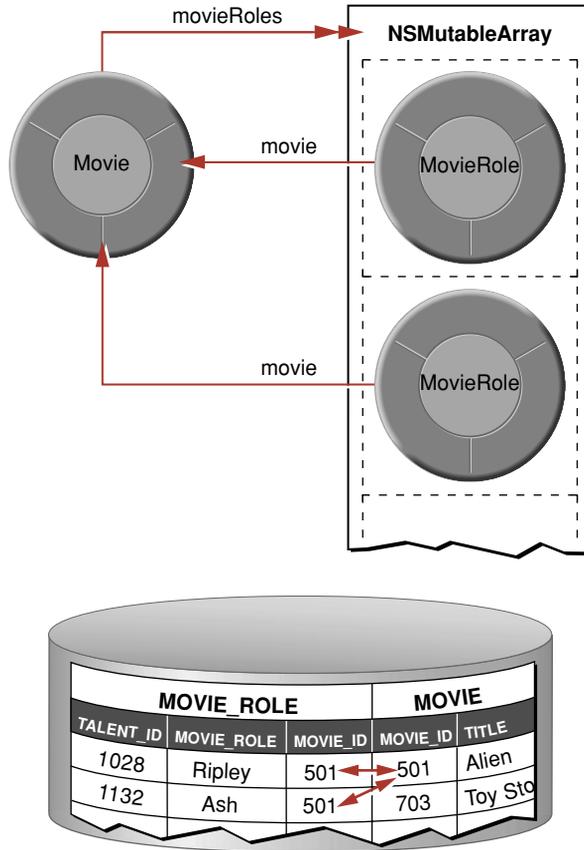
Figure 3-2 Mapping between an enterprise object class and a single table

In actual practice, the mapping is more flexible than this. For example:

- You can map an entity to a single table, a subset of a table, or to more than one table. For instance, you can map a `Person` entity's `firstName` and `lastName` attributes to a `PERSON` table but its `streetAddress`, `city`, `state` and `zipCode` attributes to an `ADDRESS` table.
- Generally, an attribute is mapped to a single column, but the column-to-attribute correspondence is similarly flexible. You can map an attribute to a *derived* column, such as `price * discount` or `salary * 12`.
- You can map an entity to one or more tables.

In addition to mapping tables to entities and columns to attributes, `WebObjects` maps primary and foreign key columns to relationships between objects. `WebObjects` defines two types of relationships—to-ones and to-manys—which are both illustrated in [Figure 3-3](#). The relationship a `MovieRole` has to its `Movie` is a to-one relationship, while the relationship a `Movie` has to its `MovieRoles` is a to-many.

Figure 3-3 Mapping relationships



WebObjects Support for Enterprise-Object Instances

After your program has accumulated changes to enterprise-object instances, WebObjects analyzes the instances, generates the necessary database operations (SQL code), and executes those operations to synchronize the database with

Enterprise Objects

in-memory enterprise-object instances. WebObjects has mechanisms for ensuring the integrity of your data is maintained between your application and the datasource without sacrificing performance or flexibility:

Validation

A good part of your application's business logic is usually validation (for example, verifying that customers don't exceed their credit limits, return dates don't come before their corresponding check-out dates, and so on). In your enterprise-object classes, you implement methods that check for invalid data, and WebObjects automatically invokes them before saving anything to the datasource.

Referential-integrity enforcement

In your model you can specify rules governing the relationships between entities, such as whether a to-one relationship is optional or mandatory. You can also specify delete rules—actions that must occur when an enterprise-object instance is deleted. For example, if you have a Department entity, you can specify that when instances of it are deleted, all the related employees in that department are also deleted (a cascading delete), all the employees in that department are updated to have no department (nullify), or the department deletion is rejected if it has any employees (deny).

Automatic primary and foreign key generation

You do not need to maintain database artifacts such as primary and foreign key values in your application; WebObjects keeps track of them for you. Primary and foreign keys aren't usually meaningful parts of a business model; rather, they're attributes created in a relational database to express relationships between entities. Key values can be generated and propagated automatically.

Transaction management

Most transactions are handled for you, using the native transaction management features of your database to group database operations that correspond to the changes that have been made to enterprise-object instances in memory. You don't have to worry about beginning, committing, or rolling back transactions unless you want to fine-tune transaction-management behavior. WebObjects also provides a separate in-memory transaction management feature that allows you to create nested contexts in which a child context's changes are folded into the parent context only upon successful completion of an in-memory operation.

Enterprise Objects

Locking

WebObjects offers three types of locking: pessimistic, optimistic, and on-demand. Pessimistic locking uses your database server's native locking mechanism to lock rows as they're fetched and prevents update conflicts by never allowing two users to look at the same enterprise-object instance at the same time. Optimistic locking doesn't detect update conflicts until you try to save changes to the database; if a row has changed since it was originally fetched, the save operation is cancelled. On-demand locking is a mixture of the other two: it locks a row after you fetch it but before you attempt to modify it. The lock can fail for one of two reasons: either the row has changed since you fetched it (optimistic locking), or because someone else already has a lock on the row (pessimistic locking).

Faulting

When WebObjects fetches a row, it creates enterprise-object instances representing the destinations of the corresponding entity's relationships. By default, WebObjects doesn't immediately fetch data for the destination objects of relationships, however. Fetching is fairly expensive, and further, if WebObjects fetched rows related to the one explicitly asked for, it would also have to fetch the rows related to those, and so on, until all of the interrelated rows in the database had been retrieved. For many applications, this would be a waste of time and resources. To avoid this, WebObjects creates empty destination objects, called faults, that fetch their data the first time they're accessed. This process, known as *faulting*, is automatic.

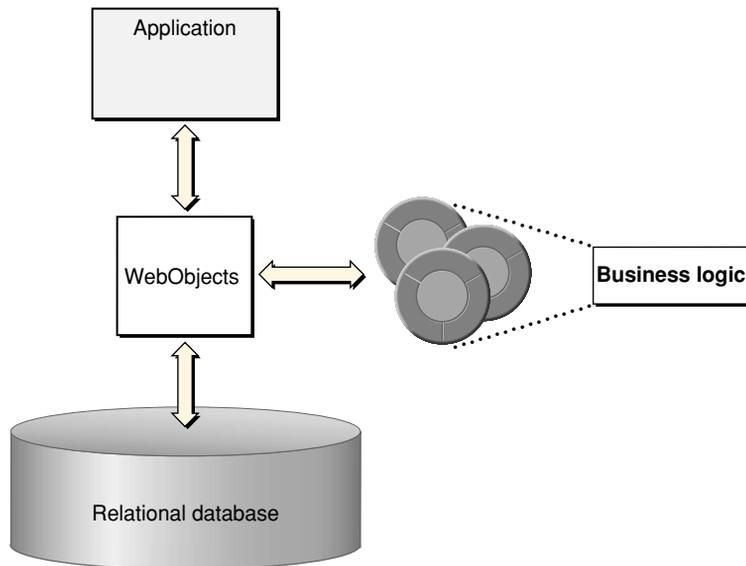
Uniquing

In marrying relational databases to object-oriented programming, one of the key requirements is that a row in the database be associated with only one enterprise object in a given context in your application. WebObjects maintains the mapping of each enterprise-object instance to its corresponding table row, and uses this information to ensure that, within a given context, your object set does not include two (possibly inconsistent) objects for the same row. Uniquing of enterprise-object instances, as this process is called, reduces memory usage and allows you to know with confidence that the object you're interacting with represents the true state of its associated row as it was last fetched.

The Enterprise Objects Advantage

A hallmark feature of WebObjects, especially in comparison to other solutions, is the separation of the business logic from the datasource and the user interface. In WebObjects, you put the business logic in the enterprise objects, as shown in Figure 3-4.

Figure 3-4 Implementing business logic in enterprise objects



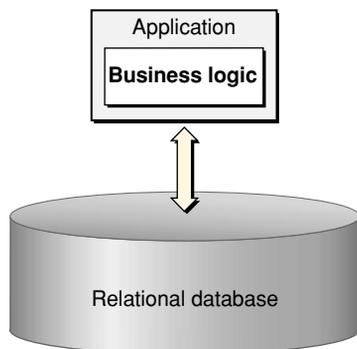
Another approach (Figure 3-5) is to implement business logic in the Web or desktop application. The WebObjects approach better this approach in the following ways:

- **It offers greater reuse.** In WebObjects, you code your business logic once, and each application that accesses your database can use it. You don't have to re-code your business logic into each screen or Web page.

Enterprise Objects

- **It's more maintainable.** With WebObjects, you don't have to duplicate your business logic. Thus you can easily make substantial changes to your rules without resorting to finding and fixing every affected page in every affected application. You can also easily track changes to your schema.
- **It improves data integrity.** In WebObjects, you don't need to rely on all application developers to implement the business rules correctly. If one application has an error, it is less likely to corrupt your database.
- **It scales better.** In WebObjects, you can improve your application's performance without having to provide your users with faster systems. Instead, you can simply move some computation-intensive processing to fast computers.

Figure 3-5 Implementing business logic in the user interface application



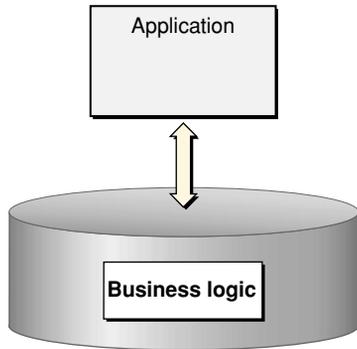
Another approach (Figure 3-6) is to implement your business rules in the database—with stored procedures, rules, constraints, and triggers, for example. The WebObjects approach betters this approach in the following ways:

- **It offers improved interactivity.** If you implement your business rules in the datasource, you need to make a round trip to the database every time the user performs an action. Alternatively, you can batch up database changes, which prevents the user from receiving immediate feedback. In WebObjects applications, changes immediately appear in the user interface, but you access the database only when saving these changes or fetching data.

Enterprise Objects

- **It improves back-end portability.** Database vendors have different ways of implementing logic. If you have to support more than one database and you're using WebObjects, you don't have to implement the logic multiple times and thus suffer maintenance problems.
- **Java is a good development language.** With WebObjects, you program in Java, an industrial-strength language designed from the ground up to be object-oriented. The programmable variants of SQL usually have some object-oriented features, but are basically procedural languages.

Figure 3-6 Implementing business logic in the database



HTML-Based Applications

The HTML-based application approach allows you to create applications that dynamically generate HTML pages. WebObjects provides graphical tools and a set of flexible frameworks with which you can develop elaborate applications. This chapter describes how a WebObjects HTML application works, the parts of a WebObjects application the programmer sees, the advantages of using this approach, and what the development process is like.

A Programmer's View of WebObjects

The following features of WebObjects ease the development of HTML-based Web applications:

- **The HTML and code reside in separate files.** An object called a **component** represents a Web page and consists of separate files for HTML and Java code.
- **WebObjects provides dynamic versions of static HTML elements.** These are called **dynamic elements**.
- **You can reuse HTML and Java code.** Components can be embedded within other components as if they were dynamic elements.
- **WebObjects automatically maintains state information.** WebObjects overcomes the inherent statelessness of HTTP and maintains session state (like a shopping cart) and application state (like application statistics).
- **Your presentation code remains separate from your business logic.** Enterprise objects, discussed in “Enterprise Objects” (page 25), contain all of your business logic. This allows you to reuse your business logic in multiple Web pages and even multiple applications.

These advantages are discussed in more detail in the sections that follow.

Separating Presentation Code from Event-Handling Logic

In WebObjects, a Web page is represented by a **component**, an object that has both content and behavior. A component can also represent a portion of a page but usually represents an entire page, so the word *page* is used interchangeably with the word *component*.

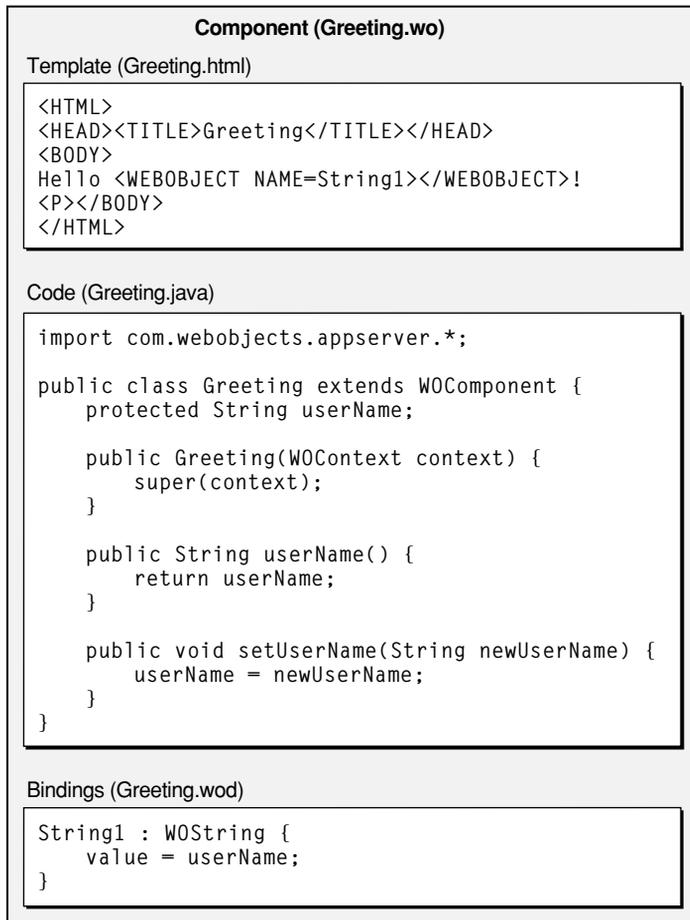
Components consist of

- **a template in HTML code that specifies how the component looks.** This file can be edited by any HTML editor or text editor.
- **event-handling logic that specifies how the component acts.** You specify this with a standard Java source file.
- **bindings that associate the component's layout specification (HTML code) with its event-handling methods.** These bindings are stored in a text file.

Separating the presentation code, event-handling logic, and bindings makes it much easier to maintain a Web application. A graphic artist can modify the presentation code, thus modifying the appearance of the page, without breaking its event-handling logic. A programmer can completely rewrite the event-handling logic without accidentally changing the page's layout.

You do not need to edit all three files separately. WebObjects Builder, a graphical component editing tool provided with WebObjects Development, edits the template, bindings, and event-handling code files simultaneously, relieving you of having to manually synchronize them. WebObjects Builder is described in more detail in “WebObjects Builder” (page 46).

Figure 4-1 (page 39) shows the three files in an example component.

Figure 4-1 The files of a WebObjects component

Dynamic Elements

The template file in Figure 4-1 looks like any other HTML file except for the element with the `<WEBOBJECT>` tag. In this example, this tag represents a **dynamic element**. Dynamic elements are the basic building blocks of a WebObjects HTML application. They link an application's behavior with the HTML page shown in the Web browser, and their contents are defined at runtime. A dynamic element appears in the template as a `<WEBOBJECT>` tag with a corresponding `</WEBOBJECT>` closing tag.

Some dynamic elements have no HTML counterpart; WORepetition and WOConditional are examples. Table 4-1 lists some of the more commonly used dynamic elements.

Table 4-1 Example Dynamic Elements

Element Name	Description
WOBrowser	selection list that displays multiple items at a time
WOCheckBox	checkbox user interface control
WOConditional	determines whether a portion of the component (or Web page) is generated
WOForm	container element that generates a fill-in form
WOHyperlink	generates a hypertext link
WOImage	displays an image
WORadioButton	represents a toggle switch
WORepetition	container element that repeats its contents (that is, everything between the <WEBOBJECT...> and </WEBOBJECT...> tags in the template file) a given number of times
WOResetButton	button that clears a form
WOString	dynamically generated string
WOSubmitButton	submit button
WOText	multiline field for text input and display
WOTextField	single-line field for text input and display

Reusing Components

You can embed a component within another component. For example, a component might represent only the header or footer of a page; you can nest it inside of a component that represents the rest of the page. A component designed to be nested within another component is called a **reusable component**, shared component, or

HTML-Based Applications

subcomponent. Like dynamic elements, reusable components appear in the template as a `<WEBOBJECT>` tag with a corresponding `</WEBOBJECT>` closing tag, allowing you to extend WebObjects's repertoire of dynamic elements.

The WOExtensions framework provided with WebObjects contains many useful reusable components like tables, radio button matrices, tab panels, and collapsible content. In addition, Direct to Web provides reusable components for editing, listing, selecting, inspecting, and querying enterprise-object instances.

Maintaining State

In addition to the components, each WebObjects application has a number of **sessions** and an **application object**.

A **session** represents a period during which a particular user is accessing your application. Because users on different client computers (or multiple browser windows) may be accessing your application at the same time, a single application typically hosts more than one session at a time. Session objects encapsulate the state of a single session. These objects persist beyond the HTTP request-response cycle, and store (and restore) the pages of a session, the values of session variables, and any other state that components need to persist throughout a session. In addition, each session has its own copy of the components that its user has requested.

Session variables can be used in shopping cart applications to represent the items in the shopping cart. Email applications can use session variables to keep track of whether the user has logged in or not.

The application object is responsible for interfacing with an HTTP adaptor and forwarding HTTP requests to a dispatcher that, in turn, passes them to the appropriate session and component. The application object also passes the HTML response from the active component back to the adaptor. In addition, the application object manages sessions, application resources, and components.

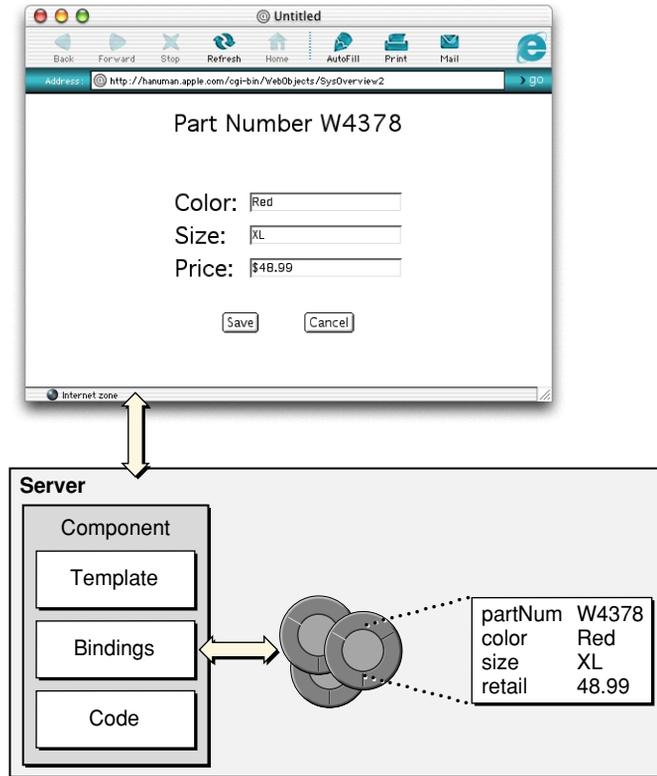
Separating Presentation Code from Business Logic

In HTML-based WebObjects applications (as in all WebObjects applications), enterprise objects encapsulate the application's business logic and provide the connection with the application's databases. Since enterprise objects are objects, they can appear as variables in components, sessions, or the application object. A

HTML-Based Applications

component's bindings file relates the component's enterprise-object instances to the attributes of its dynamic elements. Figure 4-2 shows how enterprise objects relate to a component in a WebObjects application.

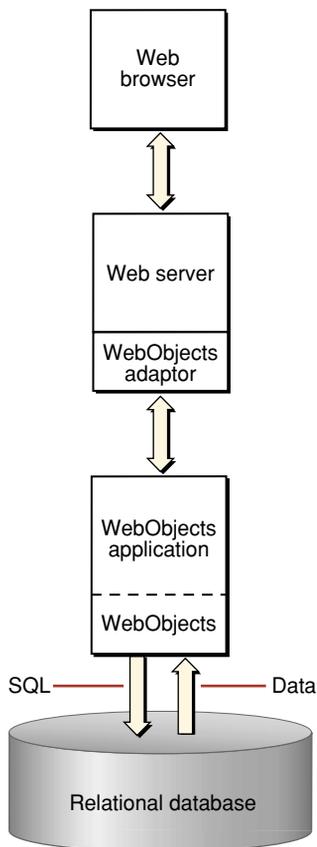
Figure 4-2 How enterprise objects relate to a WebObjects component



The WebObjects Architecture

When you run a WebObjects application, it communicates with the Web browser using the process illustrated in [Figure 4-3](#).

Figure 4-3 WebObjects HTML-based application communication chain



HTML-Based Applications

Here is a brief description of the elements involved in the communication process:

- **A Web browser.** WebObjects supports all Web browsers that conform to HTML 3.2. Of course, if your application uses more advanced features like JavaScript or QuickTime, the users' browsers must support these features.
- **A Web server.** WebObjects supports any HTTP server that uses the Common Gateway Interface (CGI), the Netscape Server API (NSAPI), the Internet Server API (ISAPI), or the Apache module API. Although necessary for deployment, you don't actually need a Web server while you develop your WebObjects applications. Alternatively, you can deploy WebObjects applications as servlets inside a servlet container. See "JSP and Servlets" (page 89) for more information.
- **An HTTP adaptor.** The HTTP adaptor connects WebObjects applications to the Web by acting as an intermediary between Web applications and HTTP servers. Note that the HTTP adaptor may not be a separate process but a Web server plugin. An HTTP adaptor is not needed when an application is deployed as a servlet.
- **A WebObjects application instance.** The application instance receives incoming requests and responds to them, usually by returning a dynamically generated HTML page. You can run multiple instances of an application when one instance is insufficient to handle the application's user load. The application process is made up of your business code and the WebObjects frameworks.

Developing a WebObjects HTML Application

Developing a WebObjects application is a matter of creating your templates, bindings, and Java code files. Although these files are text based and thus could be created using a text editor, WebObjects provides graphical tools that simplify the entire process. The sequence of tasks used to create a WebObjects HTML application with these tools is as follows:

- Create a model using EOModeler.
- Create a project using Project Builder.
- Edit your components with WebObjects Builder.

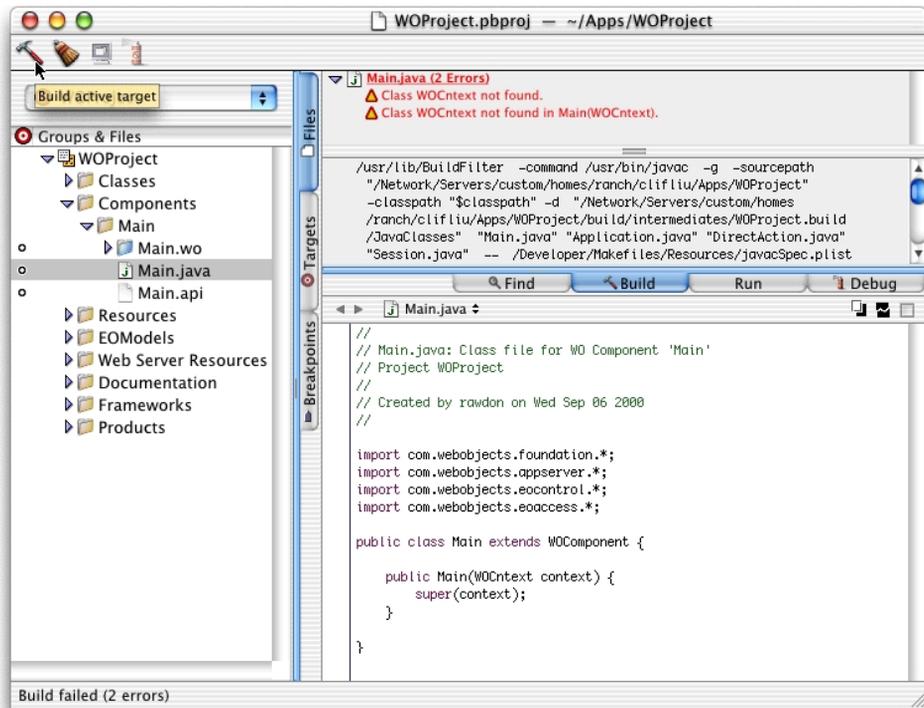
You have already been introduced to EOModeler. Project Builder and WebObjects Builder are discussed in the following sections.

Project Builder

As its name implies, Project Builder manages all of the constituent parts of your application, including source code files, WebObjects components, frameworks, makefiles, graphics and sound files, and the like. You use Project Builder to edit your code files, compile, debug, and launch your application for development testing. Project Builder's assistant helps you create new WebObjects components. You also can launch the other development tools from within Project Builder.

Figure 4-4 (page 45) shows Project Builder in use.

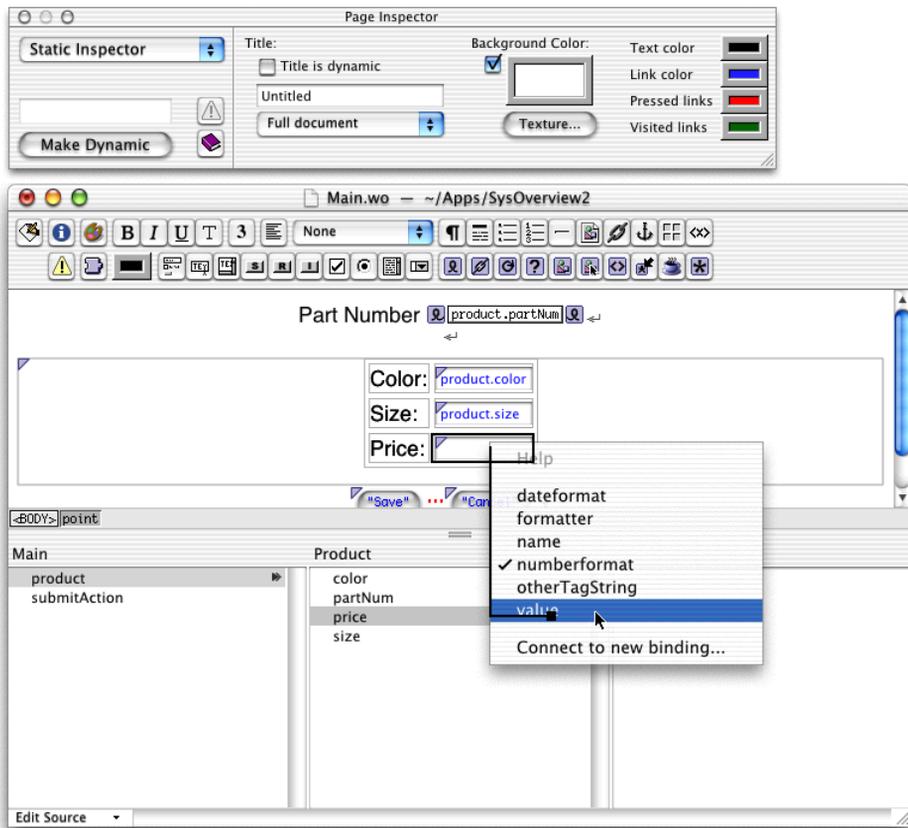
Figure 4-4 Project Builder



WebObjects Builder

You use WebObjects Builder to edit your application's components. WebObjects Builder allows you to graphically edit a component's HTML template. If you prefer, you can switch to the source view from which you can edit the template as an HTML text file. WebObjects Builder also allows you to graphically bind the dynamic elements on your template to variables and methods within your code; you simply drag from a variable to the dynamic element as shown in [Figure 4-5](#) (page 46).

Figure 4-5 WebObjects Builder



Guidelines for Choosing the HTML-Based Approach

The WebObjects HTML approach has the following advantages:

- **Portability.** Any user with a Web browser can access a WebObjects HTML application.
- **Flexibility.** You can create intricate HTML-based applications with relative ease.
- **Reduced system administration.** With the HTML-based application approach, you can publish data such as breaking news and stock prices without having to rewrite the HTML code each time, reducing the number of people necessary to keep the website up-to-date.

In some cases, you can use the Direct to Web rapid development system to create an HTML-based application. Direct to Web works particularly well for data-driven applications, prototypes, and internal applications. See “[Direct to Web Applications](#)” (page 49) for more information.

Direct to Web Applications

Direct to Web is a technology that creates HTML-based Web applications that use enterprise objects and consequently access databases. All you need to provide is the model that specifies the database-to-objects mapping and Direct to Web instantly creates an application.

Direct to Web applications have a particular structure. Every Direct to Web application begins on a login page (Figure 5-1). By default, this page provides an interface to authenticate the user but does not actually perform any authentication. Because the login page is a standard WebObjects component, you can change its behavior.

Figure 5-1 A login page



Direct to Web Applications

After the user logs in, Direct to Web displays its first dynamically generated page: a query-all page (Figure 5-2). This page allows the user to specify the enterprise objects he or she wants to work with. The user can query for any type of enterprise object that is visible in the application (the developer decides which types are visible and which are not).

Figure 5-2 A query-all page

Find...					
Movie	where	title	is	=	<input type="text"/> more..
MovieRole	where	roleName	is	=	<input type="text"/> more..
PlotSummary	where	summary	is	=	<input type="text"/> more..
Review	where	reviewer	is	=	<input type="text"/> more..
Studio	where	name	is	=	<input type="text"/> more..
Talent	where	lastName	is	=	<input type="text"/> more..
TalentPhoto	where	photo	is	=	<input type="text"/> more..
Voting	where	numberOfVotes	is	=	<input type="text"/> more..

If the query-all page is not specific enough, the user can click one of the hyperlinks labeled “more..”, which brings up a query page specific to the corresponding type of enterprise object (Figure 5-3). In this page, the user can specify the values for several properties at the same time. The resulting query is the logical “and” of the individual queries for the properties.

Figure 5-3 A query page

Movie Query	
Category	starts with <input type="text"/> <input type="button" value="↓"/>
Date Released	<input type="text"/> to <input type="text"/> <small>Format: Apr DD, YYYY</small>
Title	starts with <input type="text"/> <input type="button" value="↓"/>
Directors	where Last Name is <input type="text"/>
Revenue	<input type="text"/> to <input type="text"/> <small>(Number)</small>
Roles	where Role Name is <input type="text"/>
Plot Summary	where Summary is <input type="text"/>
Studio	where Name is <input type="text"/>
Poster Name	starts with <input type="text"/> <input type="button" value="↓"/>
Trailer Name	starts with <input type="text"/> <input type="button" value="↓"/>
Reviews	where Reviewer is <input type="text"/>
Voting	where Number Of Votes is <input type="text"/>
Rated	starts with <input type="text"/> <input type="button" value="↓"/>
	

When the user clicks the Query button on the query page or the magnifying glass icon on the query-all page, Direct to Web displays the enterprise objects matching the query on a list page (Figure 5-4). This page presents the enterprise objects in batches; the user can change the batch size and navigate from batch to batch.

Figure 5-4 A list page

43 Movie(s) Display <input type="text" value="5"/> items < Page <input type="text" value="2"/> of 9 >						
	Category	Date Released	Title	Revenue	Studio	
Edit	Drama	Sep 08,1993	Blue (Three Colors)	500000		Delete
Edit	Drama	Nov 11,1986	Blue Velvet	300000	De Laurentis	Delete
Edit	Drama	Feb 01,1995	Brazil	3000000		Delete
Edit	Dramas	Dec 28,1955	Casablanca	11200000	Warner Brothers	Delete
Edit	Drama	Nov 11,1941	Citizen Kane	100000	RKO Radio Pictures	Delete



Note that each Movie enterprise object on the list page in [Figure 5-4](#) has an Edit button, which indicates that Movie objects are read-write. The developer can configure whether a type of enterprise object is read-only or read-write.

If the Movie objects are read-only, an Inspect button appears on each row instead of an Edit button. If the user clicks the Inspect button next to one of the enterprise objects, Direct to Web displays an inspect page for the object ([Figure 5-5](#)) that reveals more detailed information about the object.

Figure 5-5 An inspect page

Studio	
Budget	23,000,000.00
Name	Universal Pictures
Movies	▶ 8 Movies
	

If the objects displayed on the list page are writable and the user clicks the Edit button next to one of them, Direct to Web displays an edit page for the object ([Figure 5-6](#)). On the edit page, the user can edit the attributes for the object or click the Edit button next to one of the relationships to edit the relationship.

Figure 5-6 An edit page

Studio						
Budget	<input type="text" value="23000000"/>					
Name	<input type="text" value="Universal Pictures"/>					
Movies	<table border="1"> <tr> <td>16, 131, 1000000, Midnight Run, Action</td> <td rowspan="4"> <input type="button" value="Edit"/> </td> </tr> <tr> <td>16, 144, 1000000, Uncle Buck, Comedy</td> </tr> <tr> <td>16, 149, 300000, Fletch, Comedy</td> </tr> <tr> <td>16, 166, 400000, Jurassic Park, Action</td> </tr> </table>	16, 131, 1000000, Midnight Run, Action	<input type="button" value="Edit"/>	16, 144, 1000000, Uncle Buck, Comedy	16, 149, 300000, Fletch, Comedy	16, 166, 400000, Jurassic Park, Action
16, 131, 1000000, Midnight Run, Action	<input type="button" value="Edit"/>					
16, 144, 1000000, Uncle Buck, Comedy						
16, 149, 300000, Fletch, Comedy						
16, 166, 400000, Jurassic Park, Action						
<table border="0"> <tr> <td><input type="button" value="Delete"/></td> <td><input type="button" value="Cancel"/></td> <td><input type="button" value="Save"/></td> </tr> </table>		<input type="button" value="Delete"/>	<input type="button" value="Cancel"/>	<input type="button" value="Save"/>		
<input type="button" value="Delete"/>	<input type="button" value="Cancel"/>	<input type="button" value="Save"/>				

The user edits a relationship using an edit-relationship page (Figure 5-7), which edits to-many and to-one relationships.

Figure 5-7 An edit relationship page

Current Roles(s)						
6 MovieRole(s)	<table border="1"> <tr> <td>Jimmy Serrano</td> <td rowspan="4"> <input type="button" value="Remove"/> </td> </tr> <tr> <td>Eddie Moscone</td> </tr> <tr> <td>Jack Walsh</td> </tr> <tr> <td>Alonzo Mosely</td> </tr> </table>	Jimmy Serrano	<input type="button" value="Remove"/>	Eddie Moscone	Jack Walsh	Alonzo Mosely
Jimmy Serrano	<input type="button" value="Remove"/>					
Eddie Moscone						
Jack Walsh						
Alonzo Mosely						
<table border="0"> <tr> <td><input type="button" value="Return"/></td> <td><input type="button" value="Build Query"/></td> <td><input type="button" value="New Record"/></td> </tr> </table>		<input type="button" value="Return"/>	<input type="button" value="Build Query"/>	<input type="button" value="New Record"/>		
<input type="button" value="Return"/>	<input type="button" value="Build Query"/>	<input type="button" value="New Record"/>				
MovieRole Query						
Role Name	<input type="text" value="starts with"/>					
Movie	where Title is <input type="text"/>					
Talent	where Last Name is <input type="text"/>					
<input type="button" value="Query DB"/>						

With the exception of the login page, every Direct to Web page has an area containing a menu and buttons that assist in navigating around the application (Figure 5-8). This is called the **menu header**.

Figure 5-8 The menu header



Every Direct to Web application appears in one of three **looks**. A look is a visual theme, and affects the layout and appearance of the pages. The example pages you have seen are in the Basic look. Direct to Web also supports two other looks: the Neutral look (Figure 5-9) and the WebObjects look (Figure 5-10 (page 55)).

Figure 5-9 An example Neutral look page

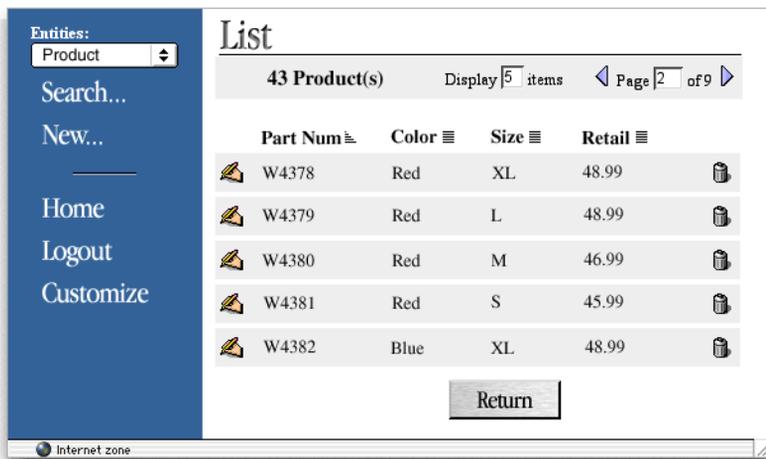


Figure 5-10 An example Basic look page

The screenshot shows a web application interface for a movie database. At the top, there is a navigation bar with a search box labeled 'Entities:' containing the word 'Movie'. To the right of the search box are buttons for 'Build Query', 'New Record', 'Home', and 'Logout'. Further right is a 'DIRECT TO WEB' logo with an Apple logo. Below the navigation bar is a status bar showing '88 Movie(s)', 'Display 10 items', and a pagination control for 'Page 1 of 9'. The main content is a table with columns: Category, Date Released, Poster Name, Rated, Revenue, Studio Id, Title, and Trailer Name. Each row has an 'Edit' link on the left and a 'Delete' link on the right. Below the table is a 'Return' button.

	Category	Date Released	Poster Name	Rated	Revenue	Studio Id	Title	Trailer Name	
Edit	Action	Aug 22,1999		G	8000000	52	WOF The Next Big Thing		Delete
Edit	Action	Apr 24,1985		R	300000	26	The Terminator		Delete
Edit	Action	Nov 11,1993		PG-13	400000	16	Jurassic Park		Delete
Edit	Action	Nov 11,1990		R	300000	35	Total Recall		Delete
Edit	Action	Dec 28,1972		R	300000	7	The Godfather		Delete
Edit	Action	Dec 27,1981		G	14400000	7	Raiders of the Lost Ark		Delete
Edit	Action	Jan 03,1988		R	1000000	16	Midnight Run		Delete
Edit	Action	Jan 03,1990		R	2021000	9	GoodFellas		Delete
Edit	Action	Jan 03,1987		R	1390000	7	The Untouchables		Delete
Edit	Action	Dec 29,1977		PG	600000	12	Star Wars		Delete

How Direct to Web Works

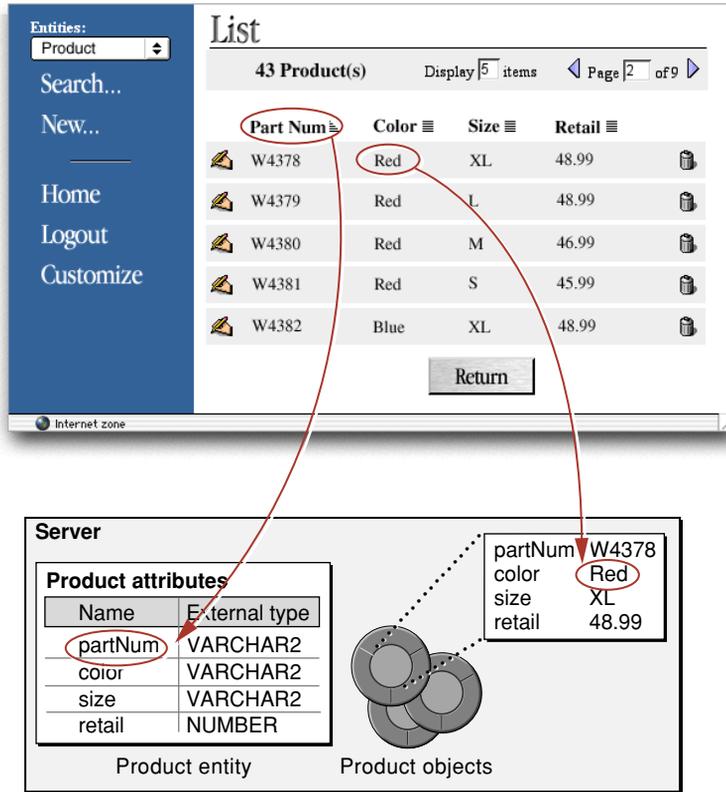
As you have seen, Direct to Web applications have a fixed structure. They consist of a set of task pages (for example, query, list, and edit pages) that work for any type of enterprise object. These task pages are created using special WebObjects components called Direct to Web templates.

A Direct to Web template uses information from the entities of the enterprise objects it displays. An entity is the piece of the model that specifies how a table maps to a specific enterprise object. The Direct to Web template takes advantage of the entity's property information (that is, information about the entity's attributes and relationships) and determines the properties it needs to display. For example, a

Direct to Web Applications

Direct to Web template displaying a list page for Movie objects can determine that it needs to display the title, release date, category, and other attributes for each movie on the page (Figure 5-11).

Figure 5-11 Determining attributes from the entity

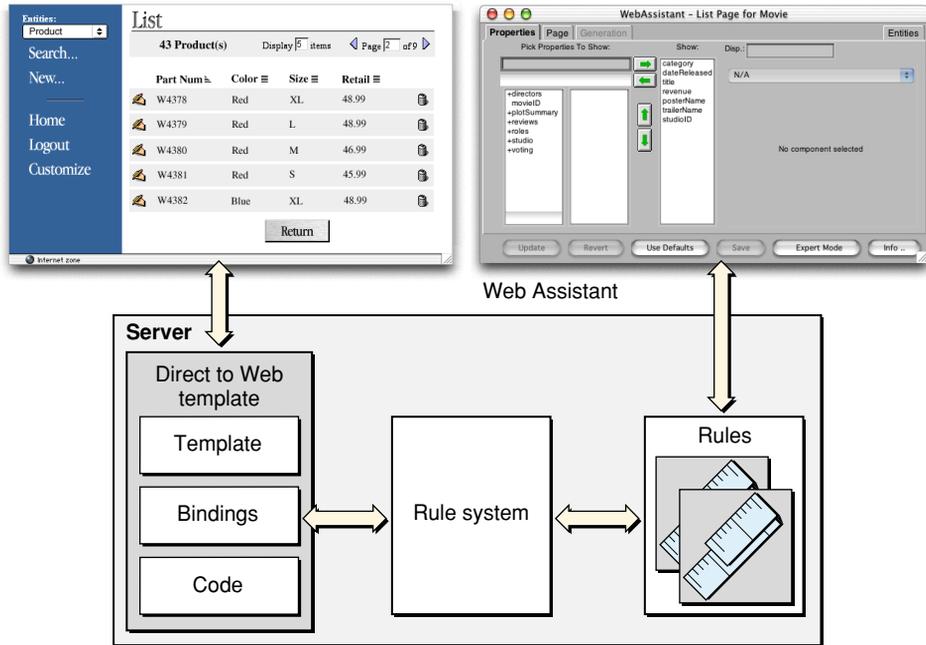


Direct to Web applications can be configured using a Java applet called the Direct to Web Assistant. The configuration information is stored as a database of rules. Rules say something like "if the task page is a list page and the entity is the Movie entity, do not display the banner." Each rule has a priority and rules with higher priority override rules with lower priority. Direct to Web defines a set of default rules that define the basic application behavior. You can define higher priority rules

Direct to Web Applications

that override the default rules for special cases. This is exactly what the Direct to Web Assistant does. Figure 5-12 shows the relationship between the Direct to Web template, the rule system, the rule database, and the Direct to Web Assistant.

Figure 5-12 The Direct to Web rule system



Note that when you configure your application with the Direct to Web Assistant, you don't need to recompile your code to try your changes. Direct to Web is not a code generation wizard. It generates Web pages at runtime based on the templates and the rules.

Developing a Direct to Web Application

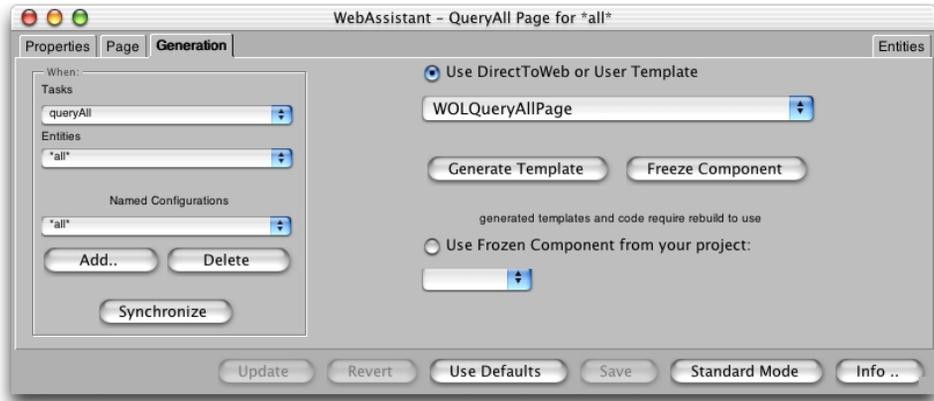
There are four steps to creating a Direct to Web application:

- Create a Direct to Web project using Project Builder.
- Create a model using EOModeler.
- Customize your Direct to Web application using the Direct to Web Assistant (optional).
- Further customize your Direct to Web application (optional).

Of the four steps, the last two are unique to Direct to Web and are discussed in more detail below.

The Direct to Web Assistant

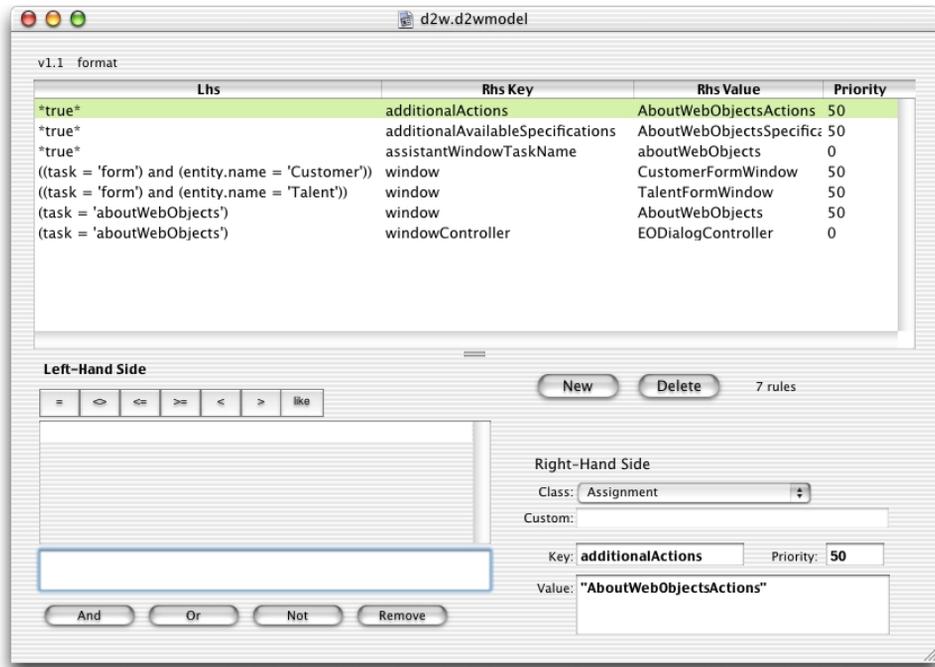
The Direct to Web Assistant is a Java applet that runs at the same time as your application. It communicates directly with Direct to Web and allows you to reconfigure your application in many ways. [Figure 5-14](#) (page 60) shows the Direct to Web Assistant in use.

Figure 5-13 The Direct to Web Assistant

With the assistant, you can designate which entities are read-write, read-only, or hidden, as shown in [Chapter Figure 5-14](#) (page 60). You can also set appearance parameters for most of the pages that Direct to Web generates. For example, you can control whether or not the page displays with a banner. You can also change the background color for the table the page displays, if applicable. The assistant also permits you to configure the way properties (attributes and relationships) appear on list, edit, and inspect pages.

Figure 5-14 The Entities pane of the Direct to Web Assistant

As mentioned earlier, the assistant defines a set of rules that override the default Direct to Web rules. Thus, the assistant is the preferred way to modify rules. However, sometimes you need to change the default rules or override the default rules in ways the assistant can't. You can use an application called the Rule Editor to edit the rules directly. [Figure 5-15](#) shows the Rule Editor.

Figure 5-15 The Rule Editor

Further Customizing Your Direct to Web Application

If you need to customize your application beyond what you can do with the Direct to Web Assistant, you can use these methods:

- Freeze a page.** When you want to change the appearance or function of a single page in your Direct to Web application, you can freeze the page with the Direct to Web Assistant. The page becomes a WebObjects component in your project with an HTML template, a Java source file, and a bindings file. You can edit it with WebObjects Builder just as you would any other component. The downside is that you can't customize frozen pages with the Direct to Web Assistant.
- Generate a Direct to Web template.** Sometimes you need to change the way every page for a particular task appears in your application. For example, you might want to put an extra hyperlink at the bottom of every list page. To do so,

Direct to Web Applications

you instruct the Direct to Web Assistant to generate a Direct to Web template, modify the template, and tell the assistant to use your customized template instead of the standard one. As mentioned earlier, a Direct to Web template is an ordinary WebObjects component and can be edited using WebObjects Builder. Unlike frozen pages, Direct to Web pages based your custom template can be customized with the assistant.

- **Modify the page wrapper and menu header.** The page wrapper component is included in your project and determines the text and elements that are common to every page in your application except the login page. It contains the menu header appropriate for the look. [Figure 5-8](#) (page 54) shows the menu header for the Basic look. The menu header is another component in your project.
- **Mix WebObjects and Direct to Web pages.** You can navigate to a Direct to Web page from a WebObjects page and vice versa. You can also embed certain Direct to Web functions within a WebObjects page. These capabilities extend the flexibility of Direct to Web considerably.
- **Perform other customizations.** You can change almost anything in a Direct to Web application because it is just a WebObjects application with some extra functionality. However, you need to know the details of the Direct to Web architecture.

Advantages of the Direct to Web Approach

Direct to Web applications are just specialized HTML-based WebObjects applications and so they have the same advantages: portability and reduced system administration. What Direct to Web adds to the HTML-based WebObjects approach is the ability to dynamically generate all of the Web pages, relieving you of designing and coding them yourself. As a consequence, Direct to Web has the following advantages over the HTML-based WebObjects approach:

- It flattens the learning curve for developing applications.
- It reduces the time required to develop applications.
- It reduces the likelihood of errors.
- It increases the maintainability and adaptability of applications.
- It increases prototyping capabilities.

Direct to Web Applications

- It allows you to focus on business logic instead of on the user interface.

Also, Direct to Web applications are constructed using well-tested Apple technology, which increases the stability of applications and reduces the time required to test applications before deploying them.

Limitations

Direct to Web is an HTML-based technology. As a result, Direct to Web user interfaces are highly portable but suffer the limited interactivity provided by HTML forms.

Because Direct to Web generates your applications for you, the applications have a number of additional limitations.

First, the programming model is indirect. You provide a model and Direct to Web assembles the application for you. The Web page generation is performed by a “magic box.” You don’t have to know what’s going on. This makes it really easy to get started programming with Direct to Web. But for certain customizations, the learning curve gets very steep very fast.

The machinery for generating Web pages is an entirely new layer on top of the WebObjects HTML-based application technology. This layer adds complexity that regular HTML-based applications don’t have, and you might have to learn the details of it to get certain results. In fact, making fundamental changes to a Direct to Web application can be a lot of work. Note, however, that you can typically reuse this work in later applications.

Another disadvantage is that modifying the layout of a Direct to Web template is more involved and harder to do than laying out a WebObjects component because Direct to Web templates are more complex than most WebObjects components.

Guidelines for Choosing Direct to Web

If your application requires a fast graphical user interface similar to that of a desktop application, you need to use one of the Java Client approaches (Java Client or Direct to Java Client). Direct to Web produces HTML-based applications.

Direct to Web is particularly suited for mission-critical applications, prototypes, and intranet applications where development time is critical and the limitations that Direct to Web imposes on the flow and user interface are not an issue. Although you can customize the application, you first need to familiarize yourself with the WebObjects HTML-based application approach and possibly Direct to Web.

Once you are familiar with how Direct to Web works, you can also use the Direct to Web reusable components in a standard WebObjects HTML-based website. This technique can dramatically reduce the development time for certain types of pages like forms and list pages.

WebObjects Desktop Applications

WebObjects recognizes the need for distributed, three-tier application solutions with more complex, rich, and responsive user interfaces than HTML allows. So, in addition to HTML-based WebObjects applications, you can also write Java-based WebObjects desktop applications which use Swing for the user interface. These applications can be deployed either as applets running in a web browser or as real desktop applications running in the client's Java Virtual Machine (though deploying as desktop applications is highly recommended). This feature of WebObjects is called *Java Client*.

In the sample Java Client application shown in [Figure 6-1](#), the user interface is like the interfaces you see in traditional desktop applications.

Figure 6-1 A sample Java Client application

WebObjects Java Client is a three-tier network application solution that allows you to develop platform-agnostic desktop applications with database access and rich user interfaces. Java Client applications are WebObjects applications: they share much of their API with traditional HTML-based WebObjects applications, such as Enterprise Objects for database access and the Foundation APIs that supply data structures and other core functionality to applications.

Java Client Features

If you're looking for a three-tier Java application platform with robust data access, rapid development tools, and powerful, innovative customization capabilities, Java Client is the perfect solution. Consider the features it offers.

Better User Experience

Java Client applications differ from HTML-based WebObjects applications in that the user interface is built on Sun's JFC/Swing classes, rather than on HTML. This allows Java Client applications to take advantage of the rich user interface elements the Swing toolkit offers. This is perhaps the primary reason why you'd want to choose to build a WebObjects application using Java Client: the need for a rich, more interactive user interface.

Rich user interfaces allow you to build more complex and interactive applications than HTML allows. As the user interface becomes more robust, it is easier to display and manipulate complex data. The more active feel of desktop applications gives users the ability to work more efficiently: desktop applications feel like they are closer to the data store.

Object Distribution

Java Client is built on the paradigm of object distribution. It distributes Enterprise Objects between an application server and one or more clients—Java applications or applets. It is up to the developer to control how this distribution occurs.

In all multi-tier intranet applications, it's vitally important that the developer has control over where the business logic sits. Some information, such as credit card numbers and passwords, are important elements of business logic but should never be sent to the client. Likewise, certain algorithms represent confidential business logic and should live only on the application server. By partitioning your business logic into a client side and a server side, you can improve performance and secure business rules and legacy data.

In pure Java applications, object distribution is crucial in protecting business rules. Since Java byte code can quite easily be decompiled, it's important that you have control over the objects which live on the client. Object distribution, coupled with remote method invocation, lets you build secure, high-performance applications.

The Best of WebObjects

As with any type of WebObjects application, Java Client gives you a lot for free. Its tight integration with Enterprise Objects takes care of many basic database access tasks for you. Without writing a single line of code, Java Client allows you to connect user interface widgets to database actions such as saving, retrieving,

WebObjects Desktop Applications

reverting, undoing, adding objects, editing objects, and more. Furthermore, Java Client's integration with Enterprise Objects abstracts development above the need to ever write a line of SQL. And the development tools you use to build Java Client applications let you build complex user interfaces in Swing without writing any code.

It is the WebObjects philosophy that the technology should take care of all the tasks fundamental to three-tier applications: database access, user interface coding, deployment, and client-server communication. That way, you can focus on writing business logic that best leverages the powerful data access mechanisms all WebObjects applications offer.

Deployment Options

WebObjects Java Client offers two deployment options—you can deploy as applets in a web browser or as real Java desktop applications. Since the Java Client architecture isolates the application from any particular data access mechanisms, you are not tied to any particular deployment scheme.

Rapid Application Development

In addition to the powerful data modeling, project development, and interface building tools, Java Client includes a sophisticated Rapid Development Environment based on the WebObjects Rule System.

The Java Client Rapid Development Environment, called Direct to Java Client, generates application user interfaces by analyzing your application's data model. Direct to Java Client allows you to immediately see how changes in your data model affect your application's user interface.

Direct to Java Client lets you focus on writing custom business logic and provides customization techniques that allow you to build sophisticated user interfaces without writing any code. Best of all, Direct to Java Client applications are completely integrated with Enterprise Objects technology, so they take full advantage of the rich data access and persistence mechanisms that technology offers.

When To Choose Java Client

Java Client is a great technology for developing and deploying desktop applications with powerful database access in controlled network environments where the end users are known and are willing to install parts of the client application. It is not suitable, however, for use in uncontrolled Internet environments, or for mass markets. Typically, Java Client applications, when deployed as desktop applications, are practical only in intranet environments.

Consider the case of a software company's bug-tracking system. Perhaps the company wants to give premium support customers access to the system through a Java Client application. These customers are assumed to be knowledgeable users and would have no problem downloading and installing certain parts of the client application. However, providing the client application as a desktop application to a large number of novice end-users would be impractical due to the support those users would need installing and maintaining a current version of the client application

When deployed as desktop applications, Java Client applications have special deployment requirements because part of the application runs on the user's machine. Unlike HTML-based applications, it is not enough to have a browser application to run a Java Client application as a desktop application. You either need to install the client-side application on the user's machine, which requires system administration, or the user needs to download the client-side application. This makes Java Client applications too complex for the average Internet application user who expects to type a URL in a browser and enter an application within seconds of hitting the web site.

However, you can also deploy Java Client applications as applets that run in browsers. Deploying as applets alleviates many of the issues encountered when running Java Client applications as desktop applications since the user doesn't need to download or install the client application. However, applets introduce other usability and deployment issues.

WebObjects Desktop Applications

In WebObjects 5.1, the Java Client Class Loader eases applet deployment and improves usability. Likewise, future enhancements to the JDK such as Java Web Start will ease application deployment and usability by providing caching and other mechanisms to ease client-side class management.

In deciding to use Java Client, you should evaluate the technology with these criteria in mind: portability, performance, network environment, administration, security, and user experience.

- **Portability:** Java Client applications are 100% Pure Java applications, requiring a JRE 1.3 or higher system. Java Client applications running on Mac OS X take advantage of platform-specific interface features such as the global menu and the dirty window marker.
- **Performance:** After the initial download of Java classes to the client, Java Client applications don't exchange large chunks of data between client and server. Rather, compact business objects are exchanged over the network. Also, the Java Client architecture separates the user interface layer from the data exchange layer, so data flows across the network independent of user interface data. This allows Java Client applications to scale well, and a WebObjects application server should scale just as well serving Java Client applications or HTML-based applications.
- **Network environment:** Java Client applications can be deployed across the Internet; they are not inherently constrained to intranet environments. However, they are not appropriate for high-volume, high-visibility applications because of the long initial download and other system administration requirements (presence of JRE 1.3 or higher).
- **System administration:** The presence of JRE 1.3 or higher is not ubiquitous amongst desktop operating systems. Mac OS X includes JRE 1.3 out-of-the-box; JRE 1.3 is not available for Mac OS 9 or earlier; Sun provides the JRE for all Windows platforms, but it does not ship in the box; JRE 1.3 is available for many UNIX platforms. So, while the JRE is widely available, it must often be downloaded and installed by the end user. You should evaluate your target market, keeping in mind that some customers will be put off by the proposition of installing the JRE.
- **Security:** If you take careful steps to partition your business logic, Java Client applications offer security equal to that of HTML-based applications. By default, Java Client uses HTTP as the transport protocol between client and server, but it can be replaced with another, more secure protocol such as SSL.

WebObjects Desktop Applications

- **Client-side processing:** Web applications do the majority of their processing on the server, while Java Client moves much of an application's processing to the client. This reduces the amount of client-server communication considerably, making Java Client applications much snappier than their Web counterparts.
- **User experience:** These criteria all affect user experience in some way. If your application demands a rich user interface, the manipulation of complex data, and long sessions, Java Client is an excellent choice.

Two Approaches to Java Client

For both HTML and Java Client applications, WebObjects offers rapid development environments that are useful both for prototyping applications and for building full-featured, usable applications. The Java Client rapid development environment is called Direct to Java Client.

Direct to Java Client and non-direct Java Client are very similar in that they use the Enterprise Objects technology the same way to access data stores. They only differ at the user interface level.

Think of the relationship this way: a Java Client application is a completely customized Direct to Java Client application. Whereas the user interface in Direct to Java Client applications is generated dynamically at run time, the user interface in Java Client applications is built by hand.

If you need the precise user interface customization that the non-direct approach allows, it's much easier to integrate a custom interface file in a Direct to Java Client application than to develop a completely custom Java Client application (though this is possible and supported). That way, you get the best of both worlds: the advantages of Direct to Java Client and the advantages of custom interfaces built with the non-direct approach.

The primary advantage of Direct to Java Client is that it's not necessary to write source code to generate or manage an application's user interface. This allows you to focus on writing business logic instead. Although the direct approach lets you manage user interfaces without writing much source code, the approach offers a number of mechanisms to customize user interfaces:

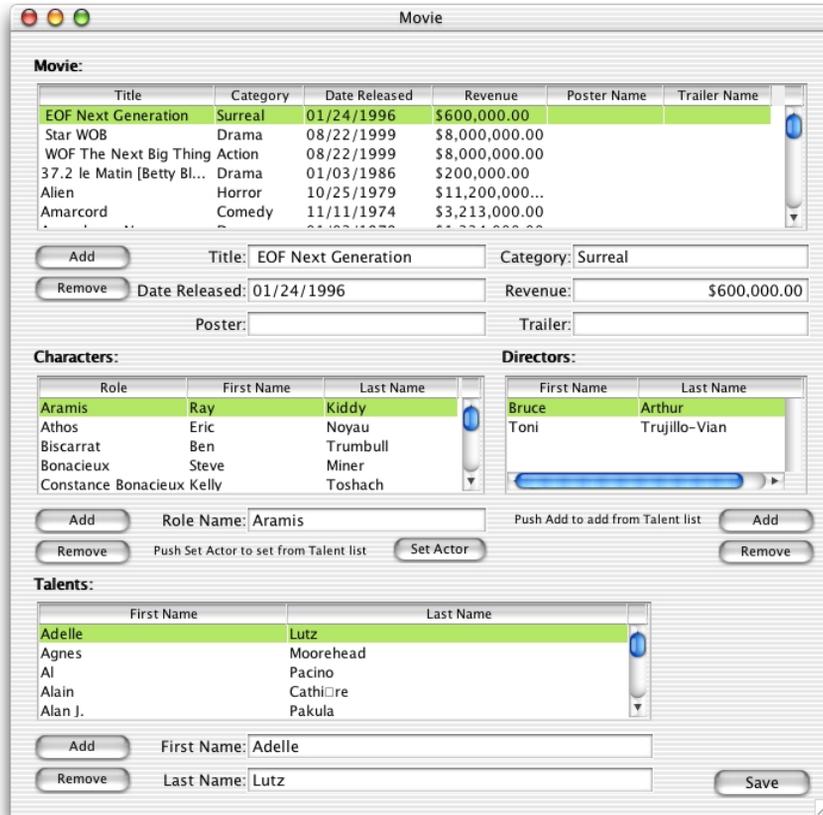
- Java Client Assistant

WebObjects Desktop Applications

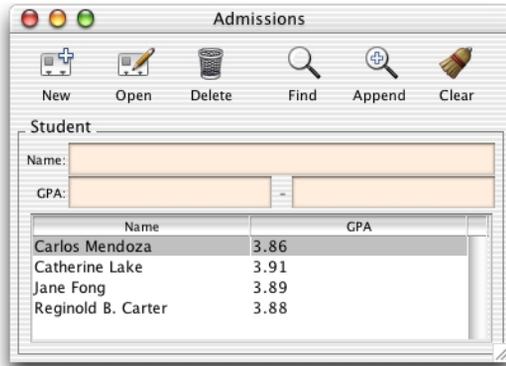
- Rule Editor
- freezing XML
- freezing .nib files
- using custom controller classes
- using factory delegates

The user interfaces for the two approaches to Java Client development each have a particular character. However, keep in mind that it's possible to customize each type of interface to look like the other.

Typically, user interfaces built in Interface Builder for non-direct Java Client applications or for use as frozen interface files in Direct to Java Client applications resemble (page 68).

Figure 6-2 A typical Java Client application

The dynamic user interface generation provided in Direct to Java Client applications yields interfaces which resemble (page 69). However, advanced Direct to Java Client applications will likely include other, non-dynamically generated user interfaces such as custom controller classes or frozen interface files (either built in Interface Builder or in raw Swing).

Figure 6-3 A typical Direct to Java Client application

Choosing an Approach

Direct to Java Client simplifies many parts of the development process, and facilitates the addition of features such as localization, data access, and data model synchronization. The direct approach to Java Client is a great way to start developing Java Client applications.

However, you will most likely want to customize your Direct to Java Client application, which requires learning some of the customization methods listed above. Although this most likely requires you to learn about the Rule System, the controller hierarchy, and XML user interface generation, the customization

WebObjects Desktop Applications

techniques work at a higher level than raw Swing. You'll also save time by learning the Direct to Java Client customization techniques rather than rolling your own Java Client interfaces in Interface Builder or in raw Swing.

Table 6-1 Comparison of Java Client and Direct to Java Client

Task	Java Client	Direct to Java Client
Customization options	Interface Builder and raw Swing.	Assistant, XML freezing, Interface Builder files, custom controller classes, controller factory delegates.
Development time	Rapid. User interfaces are automatically generated but are also easily customizable.	Moderate to heavy depending on user interface design.
User interface synchronization with data model	Difficult. User interface not synchronized with data model once user interface building begins.	Synchronization happens throughout much of the customization process.
Localization	Must use different interface files.	More automatic using localizable strings tables.

Although it is possible to build and use completely custom Java Client applications (thus not using the `eogeneration` package or XML user interface generation), there is little benefit in doing so. Using custom interfaces (`.nib` files) in Direct to Java Client applications is the most sensible way to integrate completely custom interfaces in your Java Client applications.

Java Client Architecture

The Java Client architecture differs from the HTML-based WebObjects architecture in that it's distributed across client and server systems as shown in [Figure 6-4](#). The server-side portion interacts with a database server as in HTML-based WebObjects applications, and the client-side portion, in addition to providing the application's user interface, can also contain non-sensitive business logic.

Figure 6-4 Java Client's distributed, multitier architecture

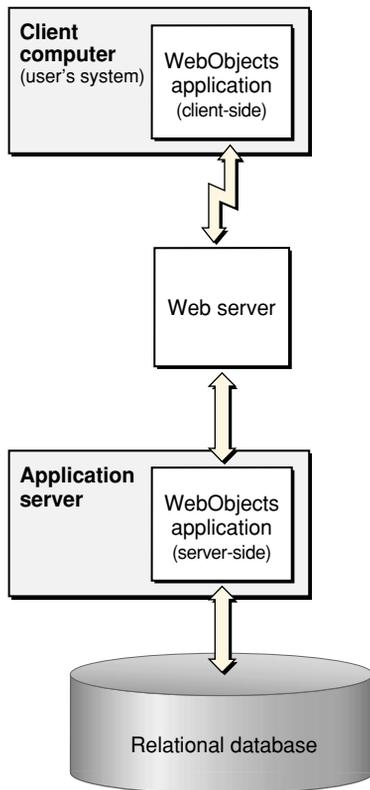
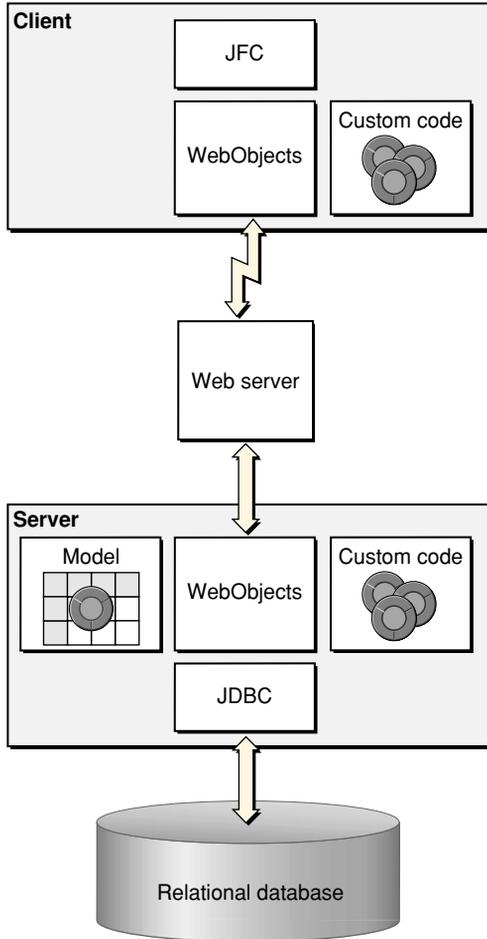


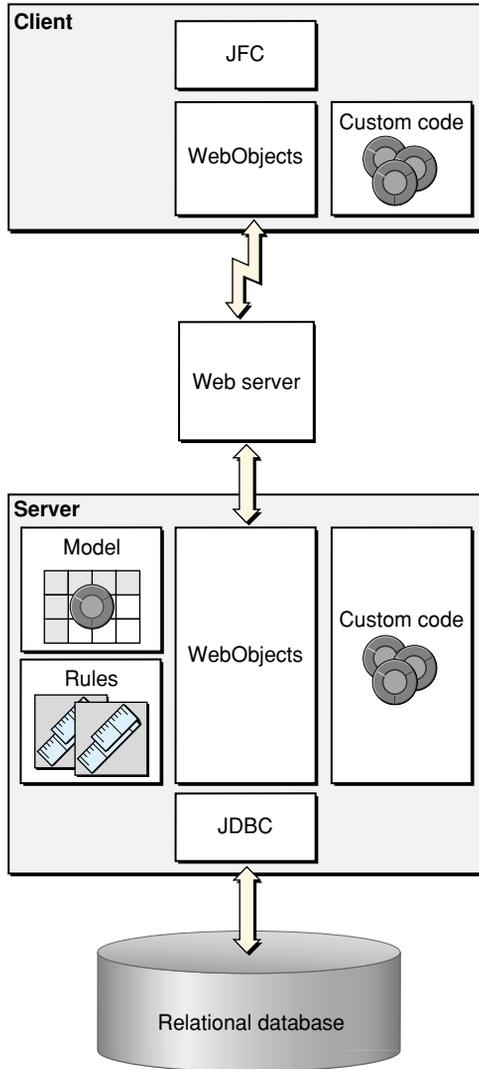
Figure 6-5 (page 77) elaborates the architecture of Java Client applications.

Figure 6-5 Architecture of a Java Client application



The architecture of Direct to Java Client applications is slightly more complex than that of non-direct Java Client applications, as illustrated in Figure 6-6 (page 78). It includes the Rule System—the part of the server-side application responsible for dynamically generating the user interface and defining its behaviors.

Figure 6-6 Architecture of a Direct to Java Client application



WebObjects Desktop Applications

The client and server applications have duties other than merely providing the user interface and database access—for example, each can contain business logic and each communicates with the other through a Web server. However, the database access/user interface division is significant because it provides a richness of user interface without compromising security or performance.

Sensitive business logic and database connection logic is provided only by the server application. Because compiled Java on the client side can be decompiled, the client application is limited to user interface code and nonsensitive business logic. At the same time, the ability to put some of the business logic on the client (any nonsensitive logic) improves performance. By performing as much processing as possible on the client (such as field validation), round trips to the server are limited.

To be clear, the Java Client architecture duplicates the graph of enterprise objects on the client application so the graph and its management occur on both server and client. WebObjects handles all client-server communication and distributes the object graph across client and server.

Managing the User Interface

The user interface itself is implemented using Java Foundation Classes (JFC). This is what gives a Java Client application the appearance and functionality of a traditional desktop application. WebObjects maps data between the application's user interface and the graph of enterprise objects. Changes to the object graph are automatically synchronized with the user interface and user-entered data is automatically reflected in the object graph.

Data Synchronization Between Client and Server

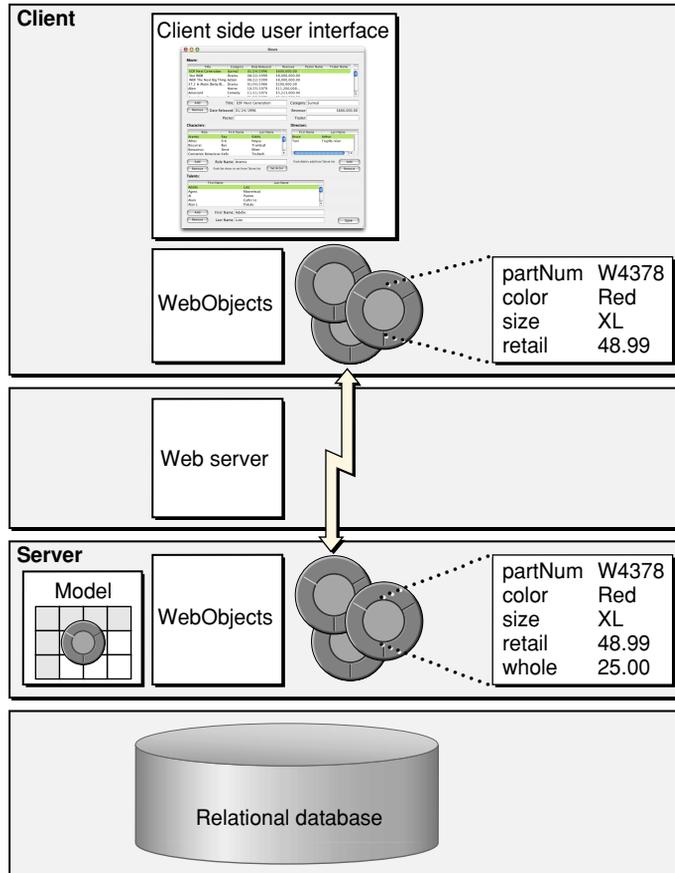
Figure 6-7 (page 80) shows the flow of data between the client and server applications for the Java Client architecture.

Starting in the upper left of the diagram and working down, when the client application initiates a fetch, the client application forwards the corresponding fetch specification to the server application. From there the normal mechanisms take over and an SQL query is performed in the database server.

WebObjects Desktop Applications

Working back up the diagram on the right side, the database server returns the rows of requested data and, as usual, this data is converted to enterprise objects. The server then sends *copies* of the requested objects to the client. When the client receives the objects, it updates its user interface with values from the requested objects.

Figure 6-7 Data flow in a Java Client application



WebObjects Desktop Applications

Although requested objects are copied from the server to the client, and these objects exist in parallel object graphs on both server and client, the enterprise objects on the client usually do not exactly mirror the enterprise objects on the server. The objects on the client usually have a subset of the properties of the objects on the server. You can partition your application's enterprise objects so the objects that exist on the client have a restricted set of data and behaviors. This ability allows you to restrict sensitive data and business logic to the server. For example, in [Figure 6-7](#), the client side enterprise objects don't have the "whole" property, the price the seller paid to the manufacturer.

Once the client has fetched data, this data is cached and is represented internally by the client's object graph. As users modify the data (or delete or add "rows" of data), the client application updates the client's object graph to reflect the new state. When the client application initiates a save, the changed objects are "pushed" to the server. If the business logic on the server validates these changes, the changes are committed to the database.

Note that Java Client automatically updates the client with changes that have occurred on the server. Whenever the client makes a request, the server passes updates along to the client with whatever information the client requested. Similarly, Java Client has the opportunity to update the client before client-side objects remotely invoke methods on server-side objects.

Dynamic User Interface Generation

Using the Direct to Java Client approach, the Swing user interface is generated dynamically by the Rule System on the server. The Rule System analyzes an application's data models (EOModel files) and generates XML descriptions of the user interface. These XML descriptions are sent to the client where they are parsed into Swing widgets to create the user interface.

Java Client and Other Three-Tier Systems

There are many distributed multi-tier Java-based architectures on the market today. So how do they compare to WebObjects Java Client?

WebObjects Desktop Applications

Client JDBC applications use a fat-client architecture. Custom code invokes JDBC on the client, which in turn goes through a driver to communicate with a JDBC proxy on the server. This proxy makes the necessary client-library calls on the server.

The shortcoming of this architecture are typical of all fat-client architectures. Security is a problem because the bytecodes on the client are easily decompiled, leaving both sensitive data and business rules at risk. In addition, this architecture doesn't scale; it is expensive to move data over the channel to the client. Also, client JDBC applications access the datasource directly—there is server layer to validate data or control access to the datasource.

JDBC three-tier applications (with CORBA as the transport) are a big improvement over client JDBC applications. In this architecture the client can be thin since all that is required on the client side are the Java Foundation Classes, nonsensitive custom code (usually for managing the user interface), and CORBA stubs for communicating with the server. Sensitive business logic and database connection logic are stored on the server. In addition, the server handles all data-intensive computations.

The JDBC three-tier architecture has its own weaknesses. First, it results in too much network traffic. Because this architecture uses proxy business objects on the client as handles to real objects on the server, each client request for an attribute is forwarded to the server, causing a separate round trip. Second, JDBC three-tier requires developers to write much of the code themselves, from code for database access and data packaging, to code for user-interface synchronization and change tracking. Finally JDBC three-tier does not provide much of the functionality associated with application servers, such as application monitoring and load balancing, nor does it provide HTML integration.

The Java Client architecture, however, scales well since real data objects live on the client and roundtrips are made to the server only for database commits and new data fetches. Also, Java Client applications are designed to leverage custom business logic which lets you to control which business objects are sent to the client and to validate data from the client before committing it to the data store.

Development Tasks and Tools

The most basic tasks of creating a Java Client application are as follows:

- Create a project using Project Builder.
- Create a model using EOModeler.
- Write source code for enterprise object classes.
- Create your application's user interface with Interface Builder (non-direct approach).
- Customize your application's user interface (Direct to Java Client approach)
- Write source code for any application-level logic.

The tasks have much in common with those for developing HTML-based WebObjects applications. The major differences are the way you design your enterprise object classes and the way you create your application's user interface.

Designing Enterprise Objects for Java Client

Java Client allows you to specify two enterprise object classes for each entity: one for the server and one for the client. The client and server classes can have different sets of properties and business logic. This means that programming a Java Client WebObjects application requires a specific design technique that isn't necessary in HTML-based development: object partitioning. Simply put, you have to determine whether you need different enterprise object classes for the client and the server and also what data and business logic to put in each class.

Usually, client objects have the more restricted set of data and behaviors, but it is really up to you to decide based on the requirements of the application and your business. As noted earlier, the primary criteria for partitioning are performance and security.

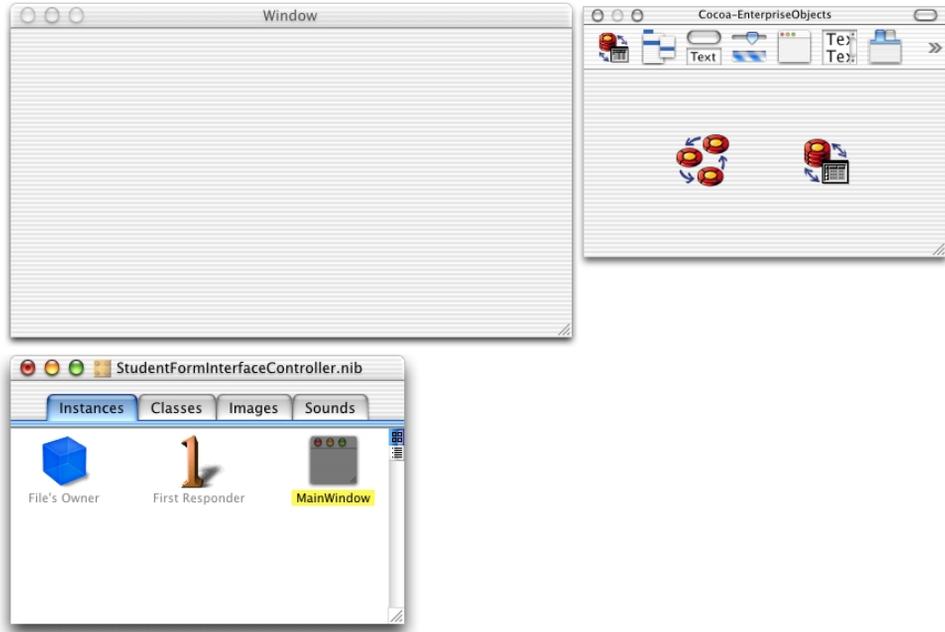
Creating the User Interface (Non-Direct)

A Java Client WebObjects application gives you considerable flexibility in how you compose its user interface. Ideally you provide an application's entire user interface in a single Java application that runs on the client. But you can also combine Java Client applets and static and dynamic (WebObjects) HTML elements in various ways. You can have pages with or without Java Client applets or pages with multiple Java Client applets. For example, you could have a login page that takes the user to one of many Java Client pages based on some piece of account data. In addition, Java Client applets are not limited to the downloaded JFC components; as with any applet, they can create dialogs and secondary windows on the fly.

If your application's user interface uses static and dynamically generated HTML, you create those parts of the user interface in the normal way with WebObjects Builder (as described in "HTML-Based Applications" (page 37)). The process is different for creating a Java Client application or applet. Instead of using WebObjects Builder to create the user interface, you use an application called Interface Builder.

Note: If you're familiar with Cocoa development, the process for creating a Java Client user interface is nearly the same as the one for creating a Cocoa user interface for Mac OS X applications.

In Interface Builder, you typically construct a user interface by dragging widgets from a palette and dropping them into a window, as shown in [Figure 6-8](#). It does more, however, than simple user interface layout. Interface Builder also lets you create, edit, and connect objects so they can communicate with one another at runtime.

Figure 6-8 Composing a user interface with Interface Builder

Customizing the User Interface (Direct Approach)

Writing custom rules is another way to customize your Direct to Java Client application. It's very similar to writing custom rules for Direct to Web applications. As with Direct to Web applications, all the information about how to configure a Direct to Java Client application is stored in the form of rules. The default rules generate the default Direct to Java Client application. Adding new rules that override or supplement the default rules is an easy-to-maintain approach that doesn't interfere with your use of the assistant. You write rules with the Rule Editor, the same application used for writing rules in Direct to Web applications. For more information, see *"Developing a Direct to Web Application"* (page 58).

There are also some more specialized ways to change the way Direct to Java Client works. For example, you can get the precise user interface layout for a particular window by freezing the interface and supplying a nib file (created in Interface Builder the way you do for regular Java Client applications). As another example, Direct to Java Client provides hooks you can use to introduce custom commands

into an application's main menu. Additionally, you can also subclass Direct to Java Client classes to change the way an application performs a particular task or to add new functionality to the default set.

Direct to Java Client was designed to be flexible and extensible, so there are numerous customization approaches. There are simple approaches that are code-free and maintainable (using the assistant and writing custom rules), there are more specialized approaches that are complex and require a lot of work, and there's everything in between. You can achieve almost any effect that you need. It's generally simply a question of which technique to use and what trade-offs you're willing to make.

J2EE Integration

Sun's J2EE (Java 2, Enterprise Edition) platform aims at laying out an infrastructure on which solutions (components or applications) from different vendors can work together and share resources. In addition, it provides a framework on which n-tier applications can be seamlessly deployed.

Enterprise JavaBeans (EJB) is an important part of J2EE. It provides an environment in which components from several manufacturers can be assembled into a working application. The application assembler, with deep knowledge of the requirements of the business, can choose the component that best matches the task at hand. For instance, she could use transaction-processing beans from one company; customer, order, and product beans from another vendor; and shipping beans from a third provider. She would then end up with an application capable of accepting orders, charging the customer, and process shipments without having to write code.

JavaServer Pages (JSP) and servlets are also part of the J2EE architecture. JSP is a specification that defines interfaces that vendors can implement to provide developers the ability to create dynamic Web pages, which are files with the extension `.jsp`. HTTP servers that support JSP, interpret these files and create servlets to process HTTP requests and produce responses. Servlets are server plugins (specialized applications) that extend the capabilities of your HTTP server. They provide a straightforward deployment mechanism for your applications.

Java Naming and Directory Service (JNDI) provides a mechanism through which components and applications can locate shared resources. For example, it allows you to place a server-side resource anywhere in your system, while the application that uses the resource only needs to know its name, not the name or IP address of the host in which it resides.

The following sections give a more detailed explanation of each of these technologies.

Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a specification that provides an infrastructure through which vendors can develop solutions that can be used by other vendors. The major part of these solutions are enterprise beans. Enterprise beans are business objects that contain logic used to perform specific tasks. They are similar to enterprise objects in WebObjects, but can be used in application servers by multiple vendors.

When an application uses enterprise beans, it's said that the application is a *client* of the bean. (Beans can themselves be clients of other beans.) Client applications don't access enterprise-bean instances directly. Instead, they interact with bean *stubs*. These stubs contain only the bean's methods that are to be accessed by client applications. Other implementation-specific methods are hidden from the client, facilitating changes and updates to the bean's business logic.

Enterprise beans are deployed in an **EJB container** (or bean container). The EJB container manages the lifecycle of enterprise-bean instances. In addition, the bean container can perform any database work required by the bean, allowing the bean developer to concentrate on business problems. When necessary, however, enterprise beans can execute database transactions themselves. Because client applications interact with bean stubs, not the bean instances themselves, bean containers are free to implement the EJB specification in a way that maximizes efficiency and performance, without affecting the functionality of client applications or the enterprise beans they contain.

To make enterprise beans available to WebObjects applications, you need to create a bean framework. Project Builder can assist you in creating such frameworks. You can create a bean framework using third-party enterprise beans, either from source code or JAR (Java archive) files, or you can write your beans from scratch.

For more information on Enterprise JavaBeans in WebObjects, see *Developing EJB Applications*.

JSP and Servlets

Servlets are generic server extensions that expand the functionality of a server. By deploying WebObjects applications as servlets running inside servlet containers, you can take advantage of the features that your servlet container offers. Alternatively, you can deploy your applications using an HTTP adaptor that talks to your HTTP server.

Deploying applications as servlets can be more efficient than using HTTP adaptors. A servlet is loaded once inside a servlet container. Concurrent requests are handled by separate threads, providing you with a high degree of scalability.

JSP is a technology that allows you to write dynamic Web pages that use JavaBeans or WebObjects components. JSPs are compiled into servlets when users access them. The servlets then process the request and typically return a Web page to the user's browser.

For more information on JSP and servlets, see *JavaServer Pages and Servlet Integration*.

JNDI

WebObjects implements the Java Naming and Directory Interface (JNDI) API. Through it WebObjects applications can access multiple naming and directory services, including Lightweight Directory Access Protocol (LDAP), Netware Directory Services (NDS), Domain Name System (DNS), and Network Information Service (NIS).

JNDI is used in EJB applications to locate beans, datasources, and other resources.

Choosing Your Approach

Choosing between the four WebObjects approaches is the first task you face as a WebObjects programmer. To make the choice you need to consider the following issues:

- Are you planning to deploy over the Internet or an intranet?
- What are your user interface requirements?
- How quickly do you need to develop the application?

The following sections, “[Internet and Intranet Deployment](#)” (page 91), “[User Interface Requirements](#)” (page 92), and “[Rapid Development Considerations](#)” (page 93) explore the approaches in more detail from the perspective of each of these issues. You can also combine approaches as described in “[Combining Approaches](#)” (page 94).

Internet and Intranet Deployment

The WebObjects HTML approach is the best approach for deploying internet applications, especially those for highly visible websites. A user on any internet-enabled computer with a Web browser can access a WebObjects HTML application.

You can also use Direct to Web to create your application, but the user interface is generally not flexible enough for public websites. If you’re familiar with WebObjects and Direct to Web, you can use Direct to Web reusable components to accelerate the development process. See “[Combining Approaches](#)” (page 94) for more information.

Choosing Your Approach

Java Client and Direct to Java Client applications are generally unsuitable for public websites because they contain client code that runs on the user's computer. Not only must the user wait for this code to download, but also the quality of the user's Java Virtual Machine determines whether the application runs correctly, efficiently, and attractively.

User Interface Requirements

The WebObjects development approaches differ in the richness and response times of the user interfaces and the ease in which you can make user interfaces with specific layout and flow requirements.

Rich Widget Selection and Fast Response Times

The Java Client and Direct to Java Client approaches offer user interfaces with multiple windows and a large selection of widgets, features commonly found in client-server applications. If your application needs these features, you should use one of these approaches. The HTML user interface used by the WebObjects HTML-based and Direct to Web approaches offers much more limited possibilities.

When you need fast response times from your user interface (if you're displaying and updating real time data, for example), you should use the Java Client or Direct to Java Client approaches. The user's computer manages the highly interactive user interface.

The drawback of the Java Client approaches is you need to be sure the client code is on the user's computer when the user runs your application. You can either install it on the user's computer in advance like a standalone application, which can be inconvenient, or download it as an applet, which can take a long time.

Specific Layout and Flow Requirements

If you plan to create an HTML application with specific page design and flow requirements, you should use the WebObjects HTML approach. The alternative, Direct to Web, creates applications with a predefined structure that limits the user

Choosing Your Approach

interface's flexibility. Direct to Web is highly customizable, but you need to have a strong understanding of WebObjects before you can effectively customize the flow of a Direct to Web application.

Your decision is similar if your application needs the rich and fast user interface the Java Client approaches offer. If the user interface has specific layout and flow requirements, you should use the Java Client approach over the Direct to Java Client approach.

Keep in mind that the Direct to Java Client approach—including the user interface it generates—is designed expressly for viewing and editing databases, especially large ones. If your application requires this capability, you will probably find Direct to Java Client's user interface extremely well-suited for the task.

Rapid Development Considerations

Using the Direct to Web and Direct to Java Client approaches, you can build an application with far less time and effort than the WebObjects HTML and Java Client approaches. You only need to provide the database-to-enterprise objects mapping (the model) and WebObjects creates your application from it. However, the rapid development approaches also impose a user interface on your application and you must be adept at WebObjects and Direct to Web to override it.

There are several types of applications at which the rapid development approaches excel because their user interface limitations aren't an issue:

- **Database maintenance tools.** These approaches create user interfaces optimized for administering databases and are therefore well-suited for this type of application.
- **Prototypes.** You can quickly and easily test a model by creating a Direct to Web or Direct to Java Client application based on the model. Using this application, you can test whether the relationships and database integrity rules are correct.
- **Internal data driven applications.** Direct to Web has been used to develop in-house applications for bug and feature tracking, customer account management, and writing online help. Direct to Java Client can be used for such

Choosing Your Approach

applications as well. For internal applications, the user interface polish is not as important as development time, making these applications ideal candidates for the WebObjects rapid development approaches.

Combining Approaches

WebObjects does not confine you to a single approach. You can switch your approach as you develop your application or combine it with another approach. This is possible in WebObjects because the business logic is encapsulated in enterprise objects and not in the application.

Combining HTML-based and Java Client Approaches

In general, you shouldn't combine a HTML-based approach (WebObjects HTML-based or Direct to Web) with a Java Client approach (Java Client or Direct to Java Client) because the combination has none of the advantages and all of the drawbacks of the individual approaches. The speed and interactivity of their user interfaces are major advantages of Java Client applications. These advantage are lost when the applications also use inherently less-interactive HTML-based interfaces.

Similarly, a major advantage of HTML-based applications is that any computer with a Web browser can use them. When combined with Java Client, these applications depend on the quality of the browser's Java Virtual Machine, if the browser even implements one. In addition, you must install the client code on the user's computer or force the user to wait for it to download. The extra interactivity Java Client adds to the HTML-based approaches is usually outweighed by the concomitant loss of portability.

Adding Rapid Development

The WebObjects HTML-based and Direct to Web approaches can be combined in many ways. You can start with a Direct to Web application, freeze and customize pages, and add your own pages. You can also start with a HTML-based application and link its components with Direct to Web pages.

Direct to Web also provides reusable components, of which the edit and list components are used the most. If your application employs forms and lists that work with enterprise objects, these components can save you a tremendous amount of time.

You can also mix Java Client and Direct to Java Client applications. If you're developing a Java Client application and you need a Direct to Java Client controller (a window that edits an enterprise object, for example), you can easily instantiate one. Also, you can freeze an interface in Direct to Java Client and edit it with Interface Builder.

Summary

The advantages and disadvantages of the four WebObjects development approaches are summarized in the following paragraphs.

The primary advantage of the WebObjects HTML-based approach is its portability. Any user with a Web-enabled computer and a Web browser can use the application. Its disadvantages are the limitations of the HTML-based user interface—delays due to round trips to the server and a limited widget set.

Direct to Web has the same advantages and limitations of the WebObjects HTML-based approach. However, it also allows you to develop data-driven applications extremely quickly. The downside of Direct to Web is that it generates a particular user interface that may not be suitable for your application.

Java Client applications provide the rich and fast user interfaces of client-server desktop applications. The disadvantage of this approach is portability. You need to install or download the application on the user's computer.

Choosing Your Approach

Direct to Java Client allows you to quickly develop data-driven Java Client applications and therefore has the advantages and disadvantages of Java Client. However, like Direct to Web, Direct to Java Client imposes a particular user interface that may not be suitable for your application.

Where to Go From Here

Once you have decided upon an approach, you can go to companion documents that cover the creation of WebObjects applications for each approach. These documents are

- *Discovering WebObjects for HTML*
- *WebObjects Desktop Applications*

Glossary

adaptor, WebObjects A process (or a part of one) that connects WebObjects applications to an HTTP server.

application object An object (of the WOApplication class) that represents a single instance of a WebObjects application. The application object's main role is to coordinate the handling of HTTP requests, but it can also maintain application-wide state information.

attribute In Entity-Relationship modeling, an identifiable characteristic of an entity. For example, lastName can be an attribute of an Employee entity. An attribute typically corresponds to a column in a database table. See also **entity**; **relationship**.

business logic The rules associated with the data in a database that typically encode business policies. An example is automatically adding late fees for overdue items.

CGI A standard for interfacing external applications with information servers, such as HTTP or Web servers. Short for Common Gateway Interface.

class In object-oriented languages such as Java, a prototype for a particular kind of object. A class definition declares instance variables and defines methods for all members of the class. Objects that have the

same types of instance variables and have access to the same methods belong to the same class.

column In a relational database, the dimension of a table that holds values for a particular attribute. For example, a table that contains employee records might have a column titled "LAST_NAME" that contains the values for each employee's last name. See also **attribute**.

component An object (of the WOComponent class) that represents a web page or a reusable portion of one.

database server A data storage and retrieval system. Database servers typically run on a dedicated computer and are accessed by client applications over a network.

Direct to Java Client A WebObjects development approach that can generate a Java Client application from a model.

Direct to Java Client Assistant A tool used to customize a Direct to Java Client application.

Direct to Web A WebObjects development approach that can generate a HTML-based Web applications from a model.

Direct to Web Assistant A tool that used to customize a Direct to Web application.

Direct to Web template A component used in Direct to Web applications that can generate a web page for a particular task (for example, a list page) for any entity.

dynamic element A dynamic version of an HTML element. WebObjects includes a list of dynamic elements with which you can build your component.

EJB container The execution environment of EJB components. It's managed by an EJB server.

enterprise object A Java object that conforms to the key-value coding protocol and whose properties (instance data) can map to stored data. An enterprise object brings together stored data with methods for operating on that data. See also **key-value coding**; **property**.

entity In Entity-Relationship modeling, a distinguishable object about which data is kept. For example, you can have an Employee entity with attributes such as lastName, firstName, address, and so on. An entity typically corresponds to a table in a relational database; an entity's attributes, in turn, correspond to a table's columns. See also **attribute**; **table**.

Entity-Relationship modeling A Discipline for examining and representing the components and interrelationships in a database system. Also known as E-R modeling, this discipline factors a database system into entities, attributes, and relationships.

EOModeler A tool used to create and edit models.

faulting A mechanism used by WebObjects to increase performance whereby destination objects of relationships are not fetched until they are explicitly accessed.

fetch In Enterprise Objects Framework applications, to retrieve data from the database server into the client application, usually into enterprise objects.

foreign key An attribute in an entity that gives it access to rows in another entity. This attribute must be the primary key of the related entity. For example, an Employee entity can contain the foreign key deptID, which matches the primary key in the entity Department. You can then use deptID as the source attribute in Employee and as the destination attribute in Department to form a relationship between the entities. See also **primary key**; **relationship**.

HTML-based application approach A WebObjects development approach that allows you to create HTML-based Web applications.

inheritance In object-oriented programming, the ability of a superclass to pass its characteristics (methods and instance variables) on to its subclasses.

instance In object-oriented languages such as Java, an object that belongs to (is a member of) a particular class. Instances are created at runtime according to the specification in the class definition.

Interface Builder A tool used to create and edit graphical user interfaces like those used in Java Client applications.

Java Client A WebObjects development approach that allows you to create graphical user interface applications that run on the user's computer and communicate with a WebObjects server.

Java Foundation Classes A set of graphical user interface components and services written in Java. The component set is known as Swing.

JDBC Informally stands for "Java Database Connectivity." An interface between Java platforms and databases.

JNDI (Java Naming and Directory Service) Protocol that provides a standard API to naming and directory services.

key An arbitrary value (usually a string) used to locate a datum in a data structure such as a dictionary.

key-value coding The mechanism that allows the properties in enterprise objects to be accessed by name (that is, as key-value pairs) by other parts of the application.

locking A mechanism to ensure that data isn't modified by more than one user at a time and that data isn't read as it is being modified.

look In Direct to Web applications, one of three user interface styles. The looks differ in both layout and appearance.

method In object-oriented programming, a procedure that can be executed by an object.

model An object (of the EOModel class) that defines, in Entity-Relationship terms, the mapping between enterprise object classes and the database schema. This definition is typically stored in a file created with the EOModeler application. A model also includes the information needed to connect to a particular database server.

Model-View-Controller An object-oriented programming paradigm in which the functions of an application are separated into the special knowledge (Model objects), user interface elements (View objects), and the interface that connects them (the Controller object).

object A programming unit that groups together a data structure (instance variables) and the operations (methods) that can use or affect that data. Objects are the principal building blocks of object-oriented programs.

primary key An attribute in an entity that uniquely identifies rows of that entity. For example, the Employee entity can contain an EmpID attribute that uniquely identifies each employee.

Project Builder A tool used to manage the development of a WebObjects application or framework.

property In Entity-Relationship modeling, an attribute or relationship. See also **attribute; relationship**.

record The set of values that describes a single instance of an entity; in a relational database, a record is equivalent to a row.

referential integrity The rules governing the consistency of relationships.

relational database A database designed according to the relational model, which uses the discipline of Entity-Relationship modeling and the data design standards called normal forms.

relationship A link between two entities that's based on attributes of the entities. For example, the Department and Employee entities can have a relationship based on the deptID attribute as a foreign key in Employee, and as the primary key in Department (note that although the join attribute deptID is the same for the source and destination entities in this example, it doesn't have to be). This relationship would make it possible to find the employees for a given department. See also **to-one**; **to-many**; **many-to-many**; **primary key**; **foreign key**.

reusable component A component that can be nested within other components and acts like a dynamic element. Reusable components allow you to extend the WebObject's selection of dynamically generated HTML elements.

request A message conforming to the Hypertext Transfer Protocol (HTTP) sent from the user's Web browser to a Web server that asks for a resource like a Web page. See also **response**.

response A message conforming to the Hypertext Transfer Protocol (HTTP) sent from the Web server to the user's Web browser that contains the resource specified by the corresponding request. The response is typically a web page. See also **request**.

row In a relational database, the dimension of a table that groups attributes into records.

rule In the Direct to Web and Direct to Java Client approaches, a specification used to customize the user interfaces of applications developed with these approaches.

Rule Editor A tool used to edit the rules in Direct to Web and Direct to Java Client applications.

session A period during which access to a WebObjects application and its resources is granted to a particular client (typically a browser). Also an object (of the WOSession class) representing a session.

table A two-dimensional set of values corresponding to an entity. The columns of a table represent characteristics of the entity and the rows represent instances of the entity.

template In a WebObjects component, a file containing HTML that specifies the overall appearance of a web page generated from the component.

to-many relationship A relationship in which each source record has zero to many corresponding destination records. For example, a department has many employees.

to-one relationship A relationship in which each source record has exactly one corresponding destination record. For example, each employee has one job title.

transaction A set of actions that is treated as a single operation.

G L O S S A R Y

Glossary

uniquing A mechanism to ensure that, within a given context, only one object is associated with each row in the database.

validation A mechanism to ensure that user-entered data lies within specified limits.

WebObjects Builder A tool used to graphically edit WebObjects components.

G L O S S A R Y

Glossary

Index

A

adaptor (WebObjects) 15, 41, 44
Apache API 44
Apple documentation 11
applet 56, 83
applets 69
application object 41
application process 44
attribute
 editing 52

B

batch 51
bean
 framework 88
 stub 88
beans 88
bindings file 38, 42
business logic 25, 67
 in enterprise object 34
 implementing 34–36
 separating from interface code 41
business object 21
business objects 69
business rules 67
bytecodes 81

C

client 88
client code 92
Client JDBC applications 81
client-server communication 17, 70, 78

Client-side processing 70
code file 38–39
Common Gateway Interface (CGI) 16, 44
component 21, 37–42 *See also* page
 creating 45
 editing of 38, 46
 persistence of 41
 reusable 40–41, 64, 91
 separating 38
 task-specific 20
controlled network environments 68
Controller object 28
CORBA 81
CORBA stubs 81
custom business logic 68, 81
customize 73

D

data access mechanisms 67
data model 68
database
 access to 20–21
 maintenance tools 93
 modifying 21
 synchronizing Java Client data 78
 tasks 20
database table
 mapping to enterprise object class 29–30
database-to-objects mapping 49
database-to-objects mapping. *See* model
decompiled 81
deploying 89
Deployment 67
deployment requirements 69
developer documentation 11
Direct to Java Client 20, 64

INDEX

Direct to Java Client application
 advantages of 91–96
 combining with HTML-based approach 94
 creating 68–85
 disadvantages of 91–96
 extending classes of 85

Direct to Web 19–20
 reusable components of 41

Direct to Web application 49–64
 advantages of 62, 91–96
 combining with WebObjects 95
 configuring 56
 customizing 61
 developing 58–60
 disadvantages of 91–96
 dynamically generated pages of 50
 fixed structure 55
 guidelines for choosing 64
 limitations of 63
 modifying single page of 61
 overview 49
 reusable components of 64
 rules 60

Direct to Web Assistant 56–62

Direct to Web template 55, 62

directory 89

DNS 89

dynamic element 37, 39–40, 46

dynamic HTML publishing 13–16

E

Edit button 52

edit page 53

edit-relationship page 53

EJB 88

EJB container 88

enterprise object 25–36, 41–42
 advantages of 34–36
 in batch 51
 configuring on list page 52
 designing for Java Client 82
 entity of 55

 extensibility of 26
 mapping to database table 29–30
 overview 25–26
 reusability of 26
 WebObjects support for 31–33

Enterprise Objects 20, 66, 67, 68

entity 55

entity-relationship (E-R) model 29

eogeneration 74

EOModeler application 29, 82

F, G

fat-client 81

faulting 33

fetching 21, 33

foreign key 32

freeze (a page) 61

H

HTTP 70

HTTP adaptor 89

HTTP protocol 21

HTTP server 44

I

inspect page 52

Interface Builder 82–83

internal application 94

Internet Server API (ISAPI) 44

intranet application 64

J

J2EE
 implementing with WebObjects 87–89

INDEX

Java Client 17, 64
 user interface of 20
Java Client application 16–18, 65–85
 advantages of 91–96
 applets 83
 architecture of 75–80
 combining with HTML-based approach 94
 creating user interface of 83
 data flow in 78–79
 database access 78
 designing enterprise objects for 82
 developing 82–83
 disadvantages of 91–96
 managing user interface of 78
Java Client Class Loader 69
Java Foundation Classes (JFC) 16, 78
Java source file 38
Java Virtual Machine 65
Java virtual machine (JVM) 94
Java Web Start 69
JavaBeans 88
JDBC 2.0 driver 21
JDBC three-tier applications 81
JDK 69
JFC 66
JNDI 89
JRE 69, 70
JSP 89

K

key-value coding 25

L

LDAP 89
list page 52
load balancing 22
localization 73
locking 33
logic. *See* business logic

login page 49
look 54–55

M

Mac OS X 69
mapping 29–30
memory usage 33
menu header 53, 62
model 27, 29
Model object 28
Model-View-Controller (MVC) 28

N

naming 89
NDS 89
Netscape Server API (NSAPI) 44
Network environment 70
Neutral look page 54
nib file 84
NIS 89

O

Object Distribution 66
object graph 78
on-demand locking 33
optimistic locking 33

P

page 38 *See also* component
page wrapper 62
partitioning 67
Performance 69
pessimistic locking 33
Portability 69

INDEX

primary key 32
Project Builder 22, 45, 82, 88
properties 25
 displaying 55
 values for 50
prototype 64, 93
proxy 81
Pure Java 22

Q

query page 51
query-all page 50, 51

R

referential integrity 32
relationship
 editing 53
 referential integrity of 32
request-response cycle 41
reusability 26, 28
reusable component 40–41, 64, 91
Rule Editor 60, 84
Rule System 68, 76
rules 56–57, 60, 84

S

scalability 22
scale 81
scales 81
Security 70
server performance 22
servlet 89
servlet container 89
session 41
 maintaining state of 41
SQL (Structured Query Language) 20
SSL 70

state management 21
static element 37
static HTML publishing 14
Swing 65, 66, 74

T

task 20, 61
template 20
template file 38–39, 46
thread 89
Three-tier 66
transaction management 32

U

uniquing 33
UNIX 70
URL (Uniform Resource Locator) 14
user interface 20
 requirements of 92–93

V

validation 32
variable 41, 46
View object 28

W

Web browser 43, 94
Web server 78
Web-enabled client-server application 16–18
WebObjects adaptor 15, 41, 44
WebObjects application 37–47
 advantages of 20–23, 91–96
 application object of 41
 application process of 44

- architecture of 43–44
- combining with Direct to Web 95
- combining with Java Client 94
- developing 44–46
- disadvantages of 91–96
- HTML-based 16–17, 64
- instances of 22
- overview 13–23
- processes of 43
- server 22
- sessions of 41
- WebObjects Builder 38, 46, 62
- WebObjects framework 44
- WebObjects look page 55
- WebStart 69
- Windows 70
- WOExtensions Framework 41

X, Y, Z

XML 74

