INSIDE WEBOBJECTS

WebObjects Overview

Preliminary

December 2000

Contents

C H A P T E R 1

About This Book

Preliminary documentation: This version of *WebObjects System Overview* is in a preliminary stage of completion. The final document accompanying the WebObjects 5 for Java release will contain updated material.

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 System Overview is written for developers who want to start using WebObjects. However, anyone interested in WebObjects technology will get something out of 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 on the subject.

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

This book has the following chapters:

- **What Is WebObjects?** Introduces the technologies of WebObjects and how they fit together.
- **Enterprise Objects.** Describes the objects that lie at the heart of all WebObjects applications and encapsulate your application's business logic and data.
- **HTML-Based Applications.** Describes the approach that allows you to create HTML applications for the World Wide Web.
- **Direct To Web Applications.** Describes the rapid development version of the HTML-based application approach.
- **Java Client Applications.** Describes the approach with which you can produce a graphical user interface application that runs on a client machine.
- **Direct to Java Client Applications.** Describes the rapid development version of the Java Client approach.
- **Choosing Your Approach.** 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 particular WebObjects application.

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.

Other Apple Publications

Apple is planning a series of documents, of which this document is the first in the series. The rest of the documents are

- *Discovering WebObjects HTML*
- *Discovering Java Client*
- *Discovering Direct to Web*
- *Discovering Direct to Java Client*
- *Creating Enterprise Objects*

After you choose an approach, you should read the appropriate document (when it becomes available). Regardless of the approach you use, you should read *Creating Enterprise Objects* as the material pertains to all approaches.

These documents are in preparation at the time of this writing. You can obtain other documents in this series (as they become available) using the publish-on-demand arrangement Apple has with Fatbrain.com.

To obtain a printed copy of one of these documents, use your Web browser to access the page at `www1.fatbrain.com/documentation/apple`. Then follow the directions. The document should be delivered to you within a few business days.

Installed Developer Documentation

When you install the WebObjects Developer 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 and if it moves (or is copied) to another location. It also makes it possible to have localized versions of the documentation (although English currently is the only supported localization).
- Development applications. 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` (`$NEXT_ROOT/Developer/Examples/JavaWebObjects` on Windows 2000) showing you how to perform common tasks using WebObjects.
- All information that is not specific to frameworks or development applications is installed in `/Developer/Documentation` (`$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 use a special interface of the Help Viewer called the Developer Help Center. Use the following steps to access the Developer Help Center:

1. Choose Help Center from the Help menu in the Finder.
2. Click the Developer Center link on the first (home) page of the Help Center.
3. To return to the Help Center, click the Help Center link on the home page of the Developer Help Center.

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.

Information on the Web

Apple maintains several websites where developers can go for general and technical information on WebObjects.

- Apple Product Information (`www.apple.com/webobjects`). Provides general information on WebObjects.
- Apple Developer Connection—Developer Documentation (`developer.apple.com/techpubs/webobjects`). Features the same documentation that is installed with WebObjects, except that often the documentation is more up-to-date. This location also includes legacy documentation.
- AppleCare Tech Info Library (`til.info.apple.com`). Contains technical articles, tutorials, FAQs, technical notes, and other information.

C H A P T E R 2

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 World Wide 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.

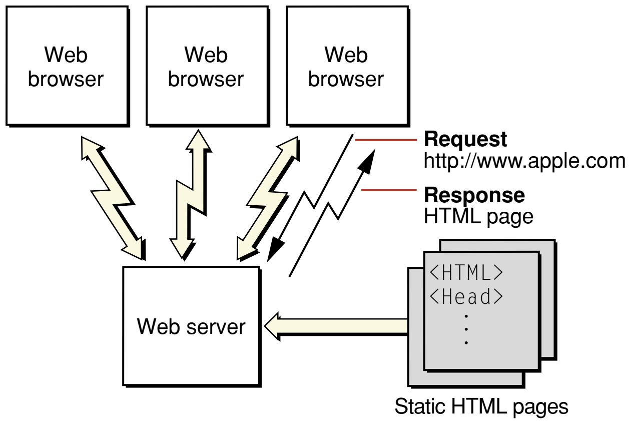
WebObjects allows you to take the next logical step. With it, you can produce full-fledged Web-accessible applications, for use either across the Internet or within a corporate intranet. These applications can be HTML-based, and thus accessible through a Web browser, or can have the full interactivity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

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 Web 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. Using the HTML tags embedded within the file received from the Web server, the browser renders the page.

Figure 2-1 A static publishing site

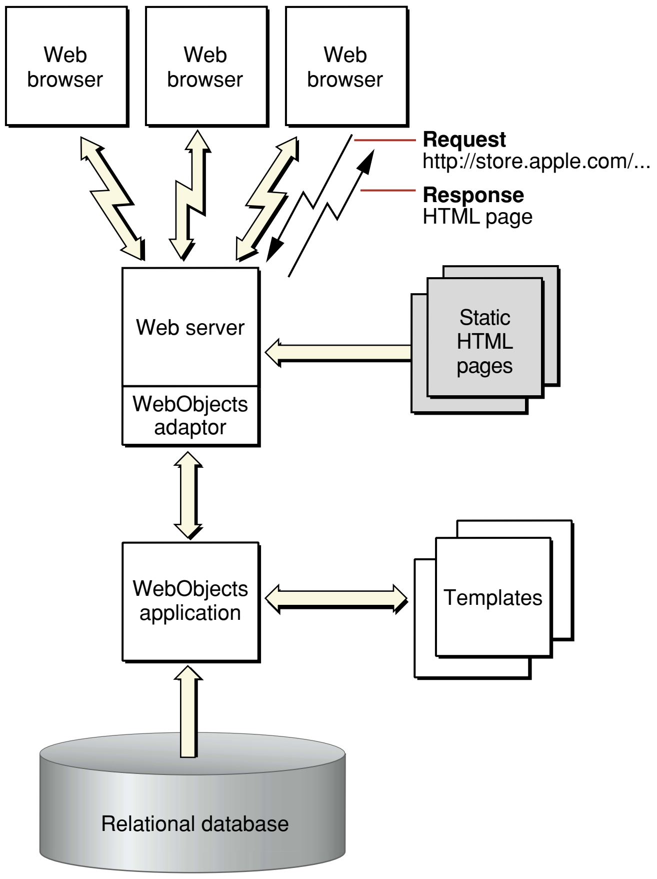


Static publishing sites are easy to maintain. There are a number of tools on the market that allow you to create HTML pages with a relatively small amount of effort, and as long as the page content doesn't change too often, it isn't that difficult to keep them up-to-date. Dynamic publishing sites, however, 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, it can reside in 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.

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 Web server detects that the request is to be handled by WebObjects, it passes the request to a WebObjects 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



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

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 a 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 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 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 13) applies to HTML-based Web applications as well.

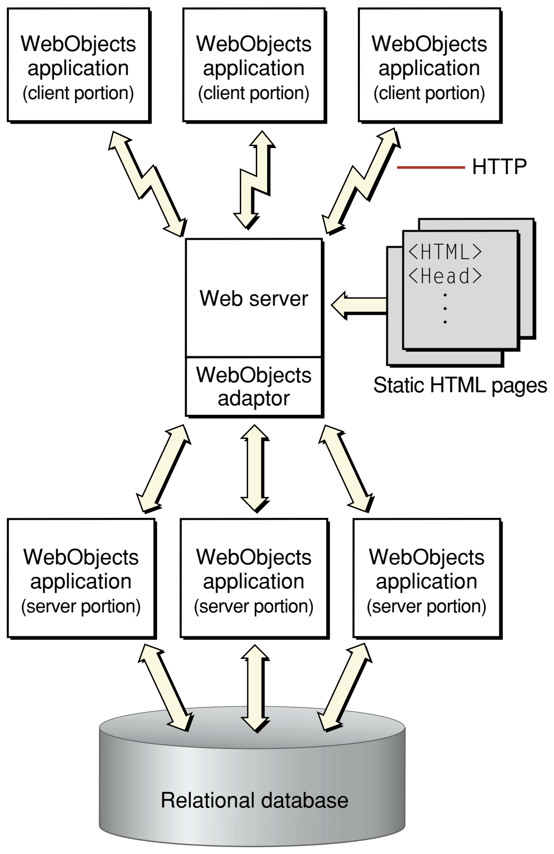
Java Client Applications

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 a Web 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 16) 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.

Figure 2-3 A website running Java Client applications


WebObjects application (client portion)
WebObjects application (client portion)
WebObjects application (client portion)
HTTP
Web server
WebObjects adaptor
<HTML>
<Head>
Static HTML pages
WebObjects application (server portion)
WebObjects application (server portion)
WebObjects application (server portion)
Relational database

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 "quick and dirty" 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 configurable system for creating HTML-based WebObjects applications that access a database. All Direct to Web needs to create the application is a model for the database, which you can build using EOModeler.

Direct to Web applications are not a set of static Web pages. Instead, Direct to Web uses information from the 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 the properties, and change the way they are displayed without recompiling or relaunching the application.

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 developer-configurable rules, 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 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 data source. 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 design and edit using standard Web authoring tools. A component can also implement custom behavior using a separate Java source file. Neither the template or 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's 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.

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 the Project Builder Integrated Development Environment (IDE), which 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 object-oriented Java APIs underlying all the different frameworks, allowing virtually unlimited customization and expandability.

Pure Java

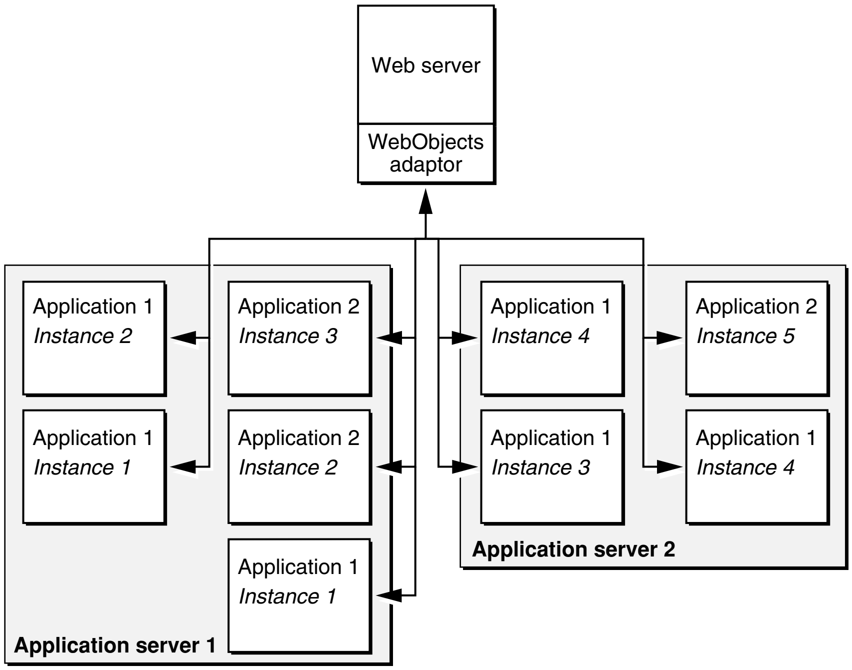
WebObjects applications are 100% Pure Java, which means they can be deployed on any platform with a certified Java 2 virtual machine.

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 21)). You can choose from one of several load-balancing algorithms (or create your own) that determine which 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.

Figure 2-4 Multiple instances, multiple applications


Web server
WebObjects adaptor
Application 1 *Instance 2*
Application 2 *Instance 3*
Application 1 *Instance 1*
Application 2 *Instance 2*
Application 1 *Instance 1*
Application server 1
Application 1 *Instance 4*
Application 2 *Instance 5*
Application 1 *Instance 3*
Application 1 *Instance 4*
Application server 2

C H A P T E R 3

Enterprise Objects

As mentioned in the previous chapter, WebObjects applications gain much of their usefulness by interacting with a persistent store, that is, a database. In WebObjects, databases are represented as collections of objects 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. The reader may wish to read the first section, "What Is an Enterprise Object?" (page 23), and the last section, "The Enterprise Objects Advantage" (page 32), 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:

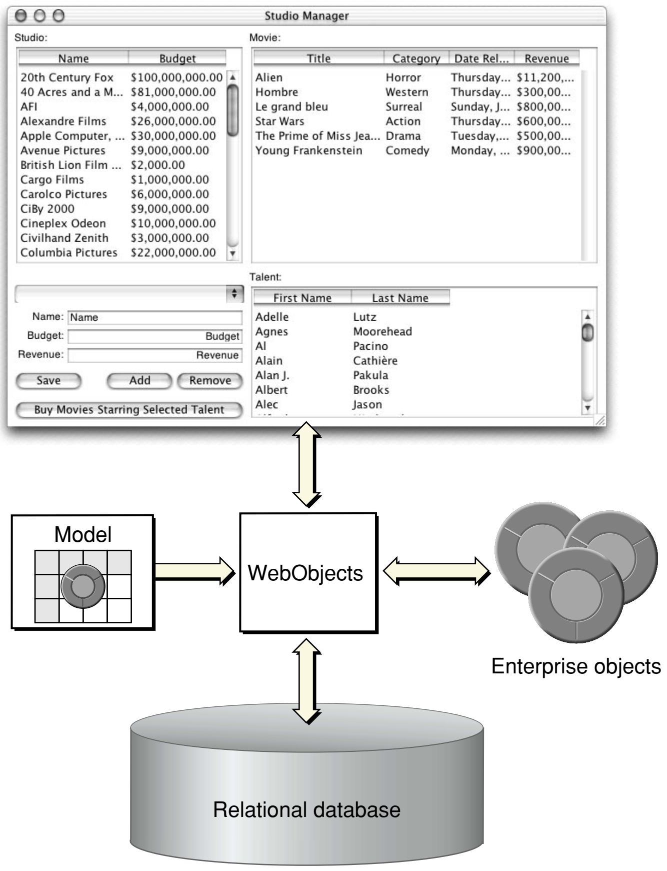
- It has properties that map to stored or persistent data; an enterprise object instance typically corresponds to a single row or record in a database.
- It knows how to interact with other parts of WebObjects to give and receive values for its properties. This is done through a mechanism known as **key-value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

In addition to providing classes that manage a set of enterprise objects 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.

If not in the business objects, then where does this knowledge go? It's handled by WebObjects as shown in Figure 3-1 (page 25).

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



Name	Budget
20th Century Fox	$100,000,000.00
40 Acres and a M...	$81,000,000.00
AFI	$4,000,000.00
Alexandre Films	$26,000,000.00
Apple Computer, ...	$30,000,000.00
Avenue Pictures	$9,000,000.00
British Lion Film ...	$2,000.00
Cargo Films	$1,000,000.00
Carolco Pictures	$6,000,000.00
CiBy 2000	$9,000,000.00
Cineplex Odeon	$10,000,000.00
Civilhand Zenith	$3,000,000.00
Columbia Pictures	$22,000,000.00

Title	Category	Date Rel...	Revenue
Alien	Horror	Thursday...	$11,200,...
Hombre	Western	Thursday...	$300,00...
Le grand bleu	Surreal	Sunday, J...	$800,00...
Star Wars	Action	Thursday...	$600,00...
The Prime of Miss Jea...	Drama	Tuesday,...	$500,00...
Young Frankenstein	Comedy	Monday, ...	$900,00...

Name:	Name
Budget:	Budget
Revenue:	Revenue

First Name	Last Name
Adelle	Lutz
Agnes	Moorehead
Al	Pacino
Alain	Cathière
Alan J.	Pakula
Albert	Brooks
Alec	Jason

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 enables you to create libraries of enterprise objects that can be used in as many applications as you need, with any user interface, and with any database server. You're able to concentrate on coding the logic of your business while WebObjects takes care of the rest.

For example, you could create a class of enterprise objects called Customer that defines such business rules as "customers must have a work or home phone number," or "the customer cannot spend more than his 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 (windows, for example, or buttons). A View object is "ignorant" of the data it displays. View objects tend to be very reusable and so provide consistency between applications.

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

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 the 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 formed from its data.

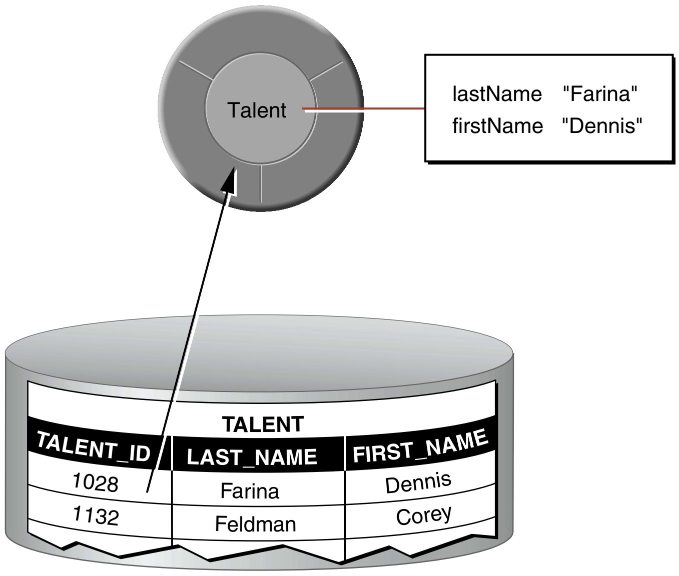
Mapping Your Enterprise Objects to Database Tables

Enterprise objects make use of a separate file, known as a **model**, to specify a mapping between tables in the database and your classes of enterprise objects. This is formally called an **entity-relationship** (E-R) model. You use the EOModeler application 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
- specify enterprise object classes for the tables in your database
- specify relationships between enterprise objects and referential integrity rules for these relationships
- generate source code files for the enterprise object classes you specify
- define fetch specifications (queries) that you can invoke by name in your applications
- create and delete databases and database tables

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 enterprise objects classes; frequently, database rows map to instances of the appropriate class as shown in Figure 3-2 (page 28).

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

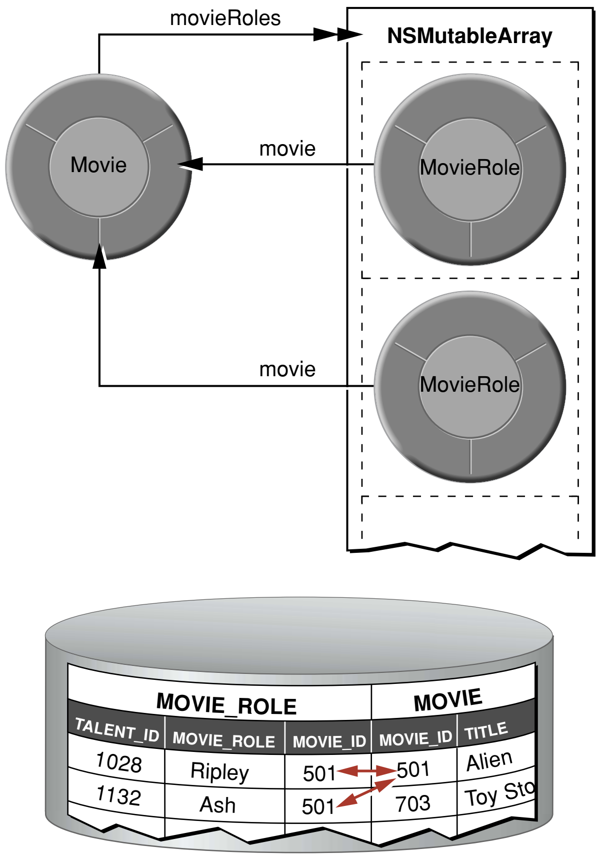


In actual practice, the mapping is more flexible than this. For example:

- You can map an enterprise object class to a single table, a subset of a table, or to more than one table. For instance, a Person object can get its first and last names from a PERSON table but get its street address, city, state and zip code from an ADDRESS table.
- Generally an enterprise object instance variable maps to a single column, but the column-to-instance variable correspondence is similarly flexible. You can map an instance variable to a derived column, such as “price * discount” or “salary * 12”.
- You can map an enterprise object class inheritance hierarchy to one or more database tables.

In addition to mapping tables to enterprise object classes and database columns to instance variables, WebObjects maps database primary and foreign keys 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



MOVIE_ROLE			MOVIE	
TALENT_ID	MOVIE_ROLE	MOVIE_ID	MOVIE_ID	TITLE
1028	Ripley	501	501	Alien
1132	Ash	501	703	Toy Sto

WebObjects Support for Enterprise Objects

After your program has accumulated changes to enterprise objects, WebObjects analyzes the objects for changes, generates corresponding database operations, and executes those operations to synchronize the database with in-memory enterprise objects. WebObjects has mechanisms for ensuring that the integrity of your data is maintained between your application and the database 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 database.

Referential integrity enforcement

In your model you can specify rules governing the relationships between objects, such as whether a to-one relationship is optional or mandatory. You can also specify delete rules—actions that should occur when an enterprise object is deleted. For example, if you have a "department" object you can specify that when it is deleted all the 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 needn't maintain database artifacts such as database primary and foreign key values in your application; WebObjects keeps track of them for you. Database 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 objects 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.

Locking

WebObjects offers three types of locking. "Pessimistic" 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 object at the same time. "Optimistic" doesn't detect update conflicts until you try to save an object's changes to the database; if the corresponding database row has changed since it was originally fetched, the save is aborted. "On-Demand" is a mixture of the other two: it locks an object after you fetch it but before you attempt to modify it. The lock can fail for one of two reasons: either the corresponding database row has changed since you fetched the object (optimistic locking), or because someone else already has a lock on the row (pessimistic locking).

Faulting

When WebObjects fetches an object, it creates objects representing the destinations of the fetched object'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 objects related to the one explicitly asked for, it would also have to fetch the objects 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.

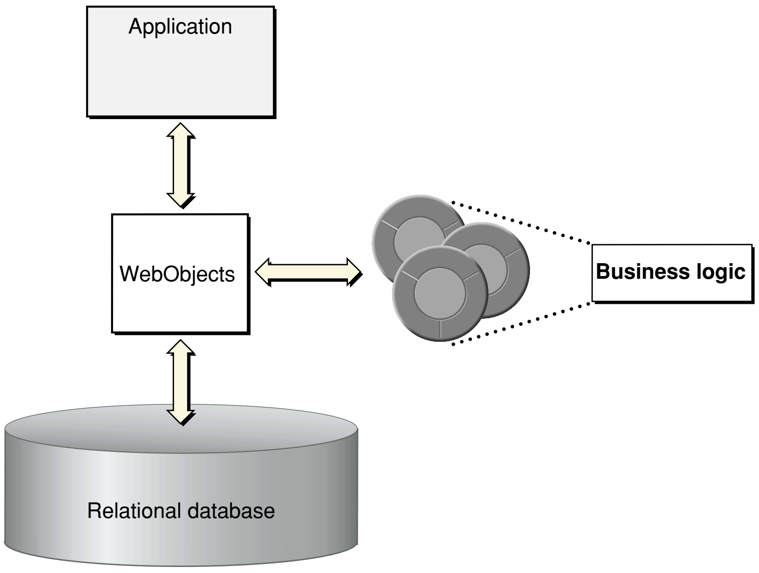
Uniquing

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 to its corresponding database 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 database row. Uniquing of enterprise objects, 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 database and the user interface. In WebObjects, you put the business logic in the enterprise objects (Figure 3-4).

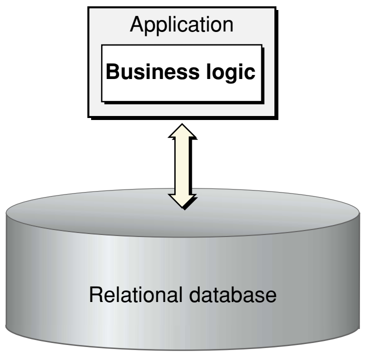
Figure 3-4 Implementing business logic in enterprise objects



Another approach (Figure 3-5) is to implement business logic in the web or desktop application. The WebObjects approach betters 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 recode your business logic into each screen or web page.
- **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 server machines.

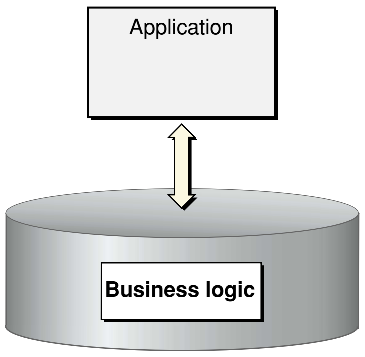
Figure 3-5 Implementing business logic in the user interface application



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 database, 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 objects.
- **It improves back-end portability.** Database vendors all have different ways to implement 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 object-oriented language. The programmable variants of SQL generally aren't object-oriented and suffer from weakly defined language specifications—the programming constructs are usually grafted on to standard SQL.

Figure 3-6 Implementing business logic in the database



C H A P T E R 4

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 extremely complex 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 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 Web interface code remains separate from your business logic.** Enterprise objects, discussed in Chapter 3, 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 HTML and Code

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

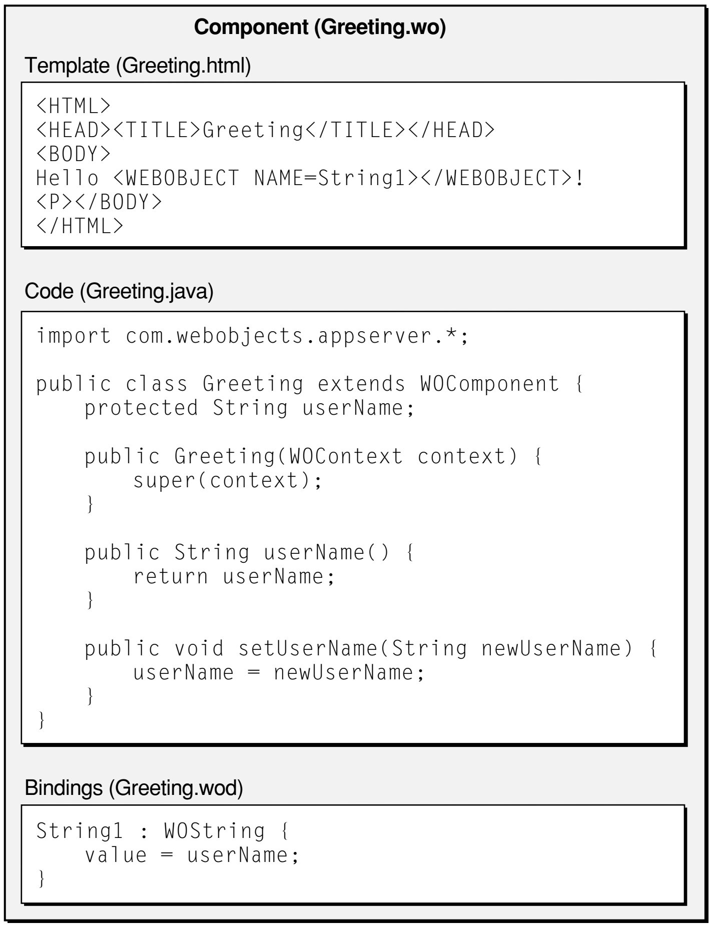
- **a template in HTML that specifies how the component looks.** This file can be edited by any HTML editor or text editor.
- **code that specifies how the component acts.** You specify this with a standard Java source file.
- **bindings that associate the component's template with its code.** These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

Separating the template, code, and bindings makes it much easier to maintain a website. A graphic artist can modify a template, thus modifying the appearance of the page, without breaking the code. A programmer can completely rearrange the code without accidentally changing the layout.

You do not need to edit all three files separately. WebObjects Builder, a graphical component editing tool provided with WebObjects, edits the template, bindings, and code files simultaneously, relieving you of having to manually synchronize them. WebObjects Builder is described in more detail in "WebObjects Builder" (page 44).

Figure 4-1 (page 37) shows the three files in an example component.

Figure 4-1 The files of a WebObjects component



```
<HTML>
<HEAD><TITLE>Greeting</TITLE></HEAD>
<BODY>
Hello <WEBOBJECT NAME=String1></WEBOBJECT>!
<P></BODY>
</HTML>
```

```
import com.webobjects.appserver.*;

public class Greeting extends WOComponent {
    protected String userName;

    public Greeting(WOContext context) {
        super(context);
    }

    public String userName() {
        return userName;
    }

    public void setUserName(String newUserName) {
        userName = newUserName;
    }
}
```

```
String1 : WOString {
    value = userName;
}
```

Dynamic HTML 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 basic building blocks of a WebObjects 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	A selection list that displays multiple items at a time.
WOCheckBox	A checkbox user interface control.
WOConditional	Controls whether a portion of the HTML page will be generated.
WOForm	A container element that generates a fill-in form.
WOHyperlink	Generates a hypertext link.
WOImage	Displays an image.
WORadioButton	Represents a toggle switch.
WORepetition	A 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	A button that clears a form.
WOString	A dynamically generated string.
WOSubmitButton	A submit button.
WOText	A multiline field for text input and display.
WOTextField	A 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 a 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 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' repertoire of dynamically generated HTML 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 objects.

Maintaining State

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

A **session** is a period during which a particular user is accessing your application. Because users on different clients may be accessing your application at the same time, a single application may host more than one session at a time. Session objects encapsulate the state of a single session. These objects persist beyond the HTTP request-response cycles, 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.

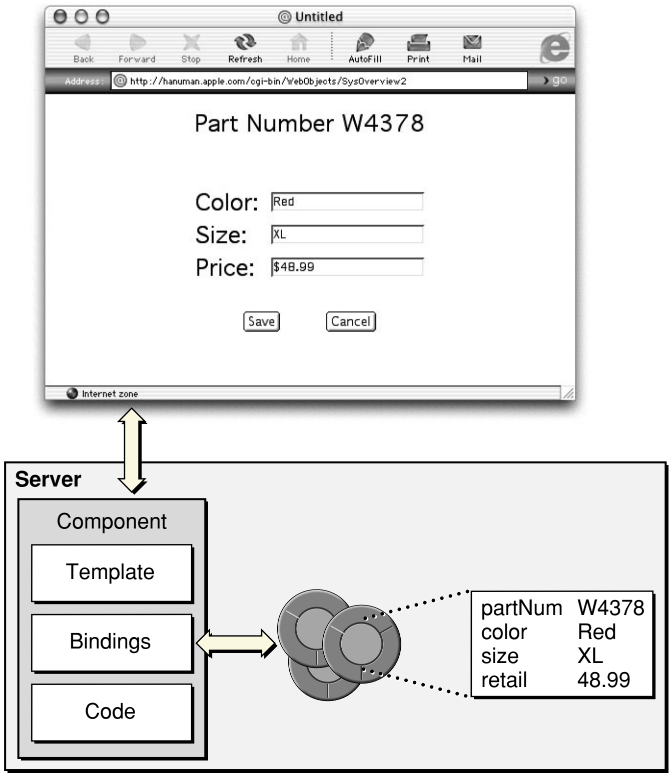
Session variables can be used in shopping cart applications to represent the items in the shopping cart. Email applications can use session variables to keep track of whether the user has logged in or not.

The application object is responsible for interfacing to an 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 adaptors, sessions, application resources, and components.

Separating Web Interface Code from Business Logic

In HTML-based WebObjects applications (as in all WebObjects applications), the 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 component's bindings file relates the component's enterprise objects to the attributes of its dynamic elements. Figure 4-2 shows how enterprise objects relate to a component in a WebObjects application.

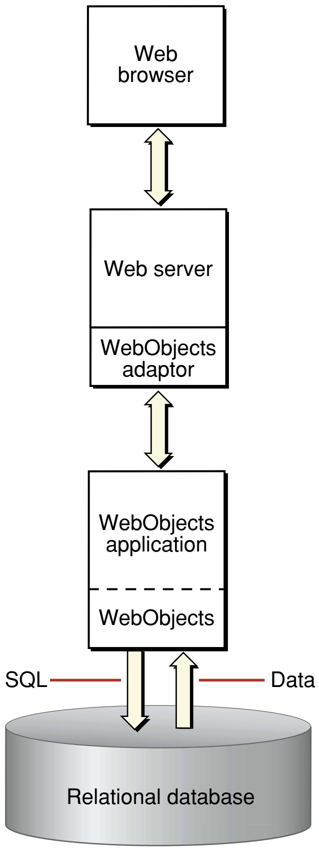
Figure 4-2 How enterprise objects relate to a WebObjects component



WebObjects Architecture

When you run a WebObjects application, it communicates with the Web browser through the chain of processes shown in Figure 4-3.

Figure 4-3 WebObjects HTML-based application communication chain



Here is a brief description of these processes:

- **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.
- **A WebObjects adaptor.** A WebObjects adaptor connects WebObjects applications to the Web by acting as an intermediary between Web applications and HTTP servers. Note that the WebObjects adaptor may not be a separate process but plug-in to the Web server.
- **A WebObjects application process.** The application process receives incoming requests and responds to them, usually by returning a dynamically generated HTML page. You can run multiple instances of this process if one instance is insufficient to handle the application load. The application process is made up of your 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 project using Project Builder.
- Create a model using EOModeler.
- Edit your components with WebObjects Builder.

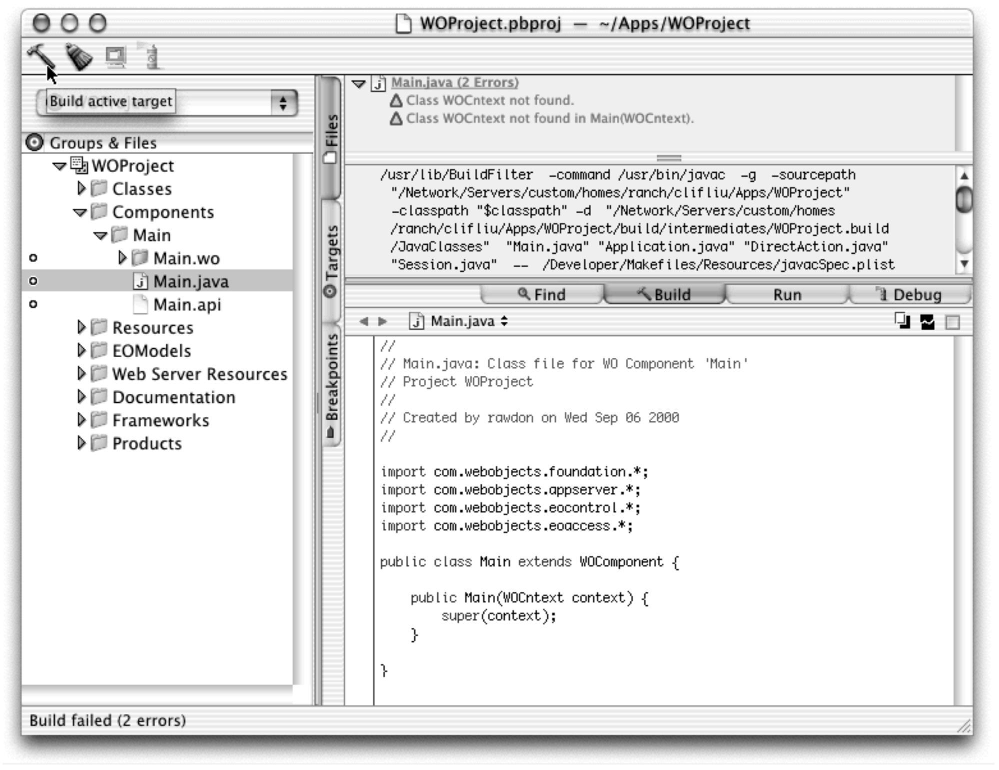
You have already been introduced to EOModeler. Project Builder and WebObjects Builder are discussed in the following sections.

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 assistants help you create new WebObjects components. You also can launch the other development tools from within Project Builder.

Figure 4-4 (page 43) shows Project Builder in use.

Figure 4-4 Project Builder



```
Main.java (2 Errors)
  Class WOCntext not found.
  Class WOCntext not found in Main(WOCntext).
```

```
/usr/lib/BuildFilter  -command /usr/bin/javac  -g  -sourcepath
 "/Network/Servers/custom/homes/ranch/cliftliu/Apps/WOProject"
 -classpath "$classpath" -d  "/Network/Servers/custom/homes
 /ranch/cliftliu/Apps/WOProject/build/intermediates/WOProject.build
 /JavaClasses"  "Main.java" "Application.java" "DirectAction.java"
 "Session.java"  --  /Developer/Makefiles/Resources/javacSpec.plist
```

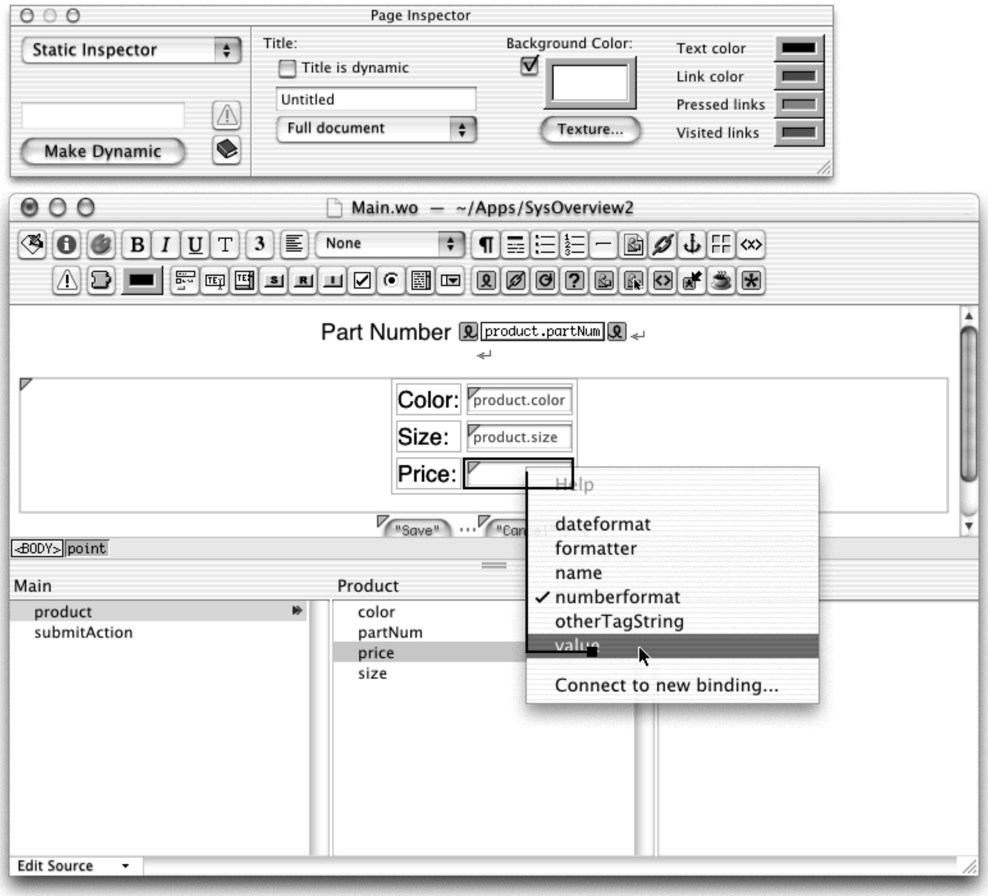
```
//
// Main.java: Class file for WO Component 'Main'
// Project WOProject
//
// Created by rawdon on Wed Sep 06 2000
//

import com.webobjects.foundation.*;
import com.webobjects.appserver.*;
import com.webobjects.eocontrol.*;
import com.webobjects.eoaccess.*;

public class Main extends WOComponent {

    public Main(WOCntext context) {
        super(context);
    }

}
```

Build failed (2 errors)

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 45).

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 extremely complex HTML applications with the WebObjects HTML approach.
- **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 each time, reducing the number of people necessary to keep the site up to date.

In some cases, you can use the Direct to Web rapid development system to create your HTML-based application for you. Direct to Web works particularly well for data-driven applications, prototypes, and internal applications. See "Direct to Web Applications" (page 47) for more information.

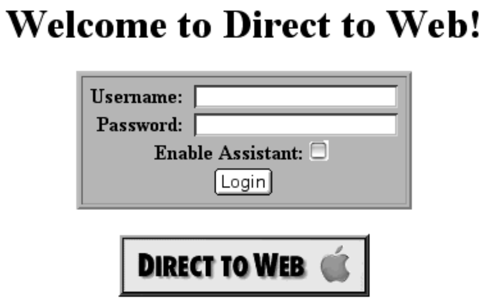
C H A P T E R 5

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.

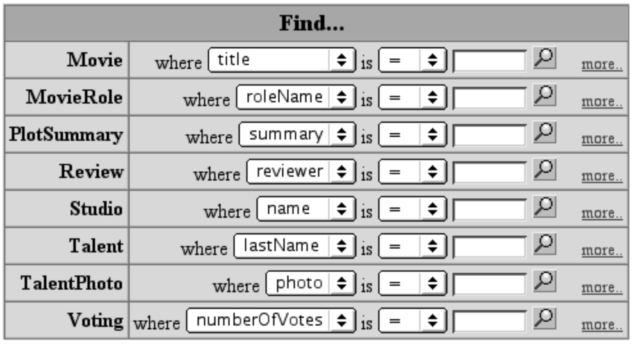
Direct to Web applications have a particular structure. Every Direct to Web application begins on a login page (Figure 5-1). By default, this page provides an interface to authenticate the user but does not actually perform any authentication. Because the login page is a standard WebObjects component,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

Figure 5-1 A login page



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).

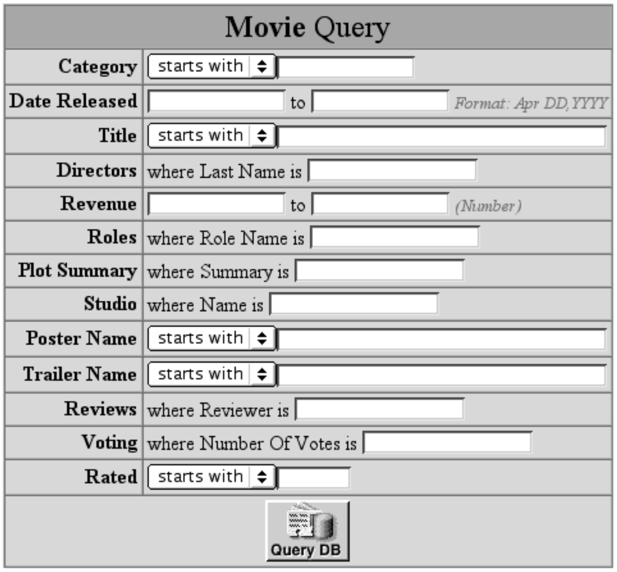
Figure 5-2 A query-all page



Find...						
Movie	where	title	is	=		more..
MovieRole	where	roleName	is	=		more..
PlotSummary	where	summary	is	=		more..
Review	where	reviewer	is	=		more..
Studio	where	name	is	=		more..
Talent	where	lastName	is	=		more..
TalentPhoto	where	photo	is	=		more..
Voting	where	numberOfVotes	is	=		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

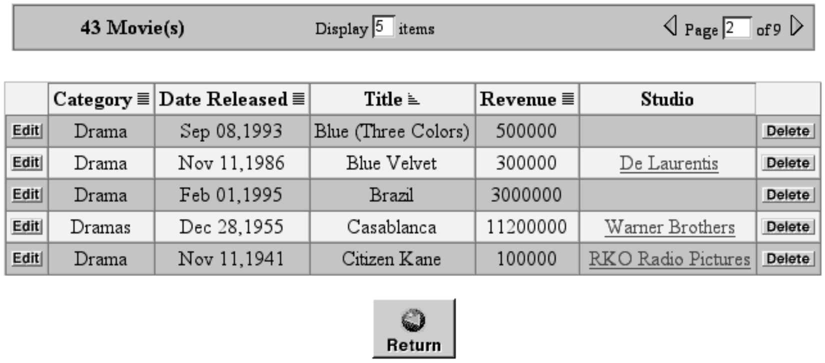


Movie Query

Category	starts with
Date Released	to Format: Apr DD, YYYY
Title	starts with
Directors	where Last Name is
Revenue	to (Number)
Roles	where Role Name is
Plot Summary	where Summary is
Studio	where Name is
Poster Name	starts with
Trailer Name	starts with
Reviews	where Reviewer is
Voting	where Number Of Votes is
Rated	starts with

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

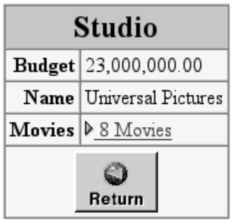

43 Movie(s) Display 5 items Page 2 of 9

	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.

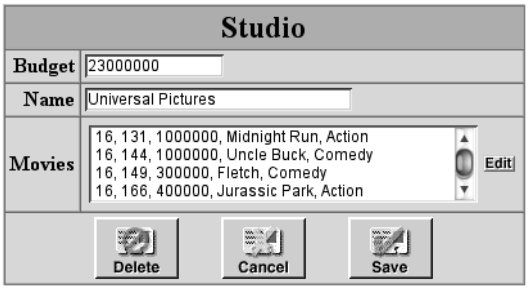
Figure 5-5 An inspect page


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.

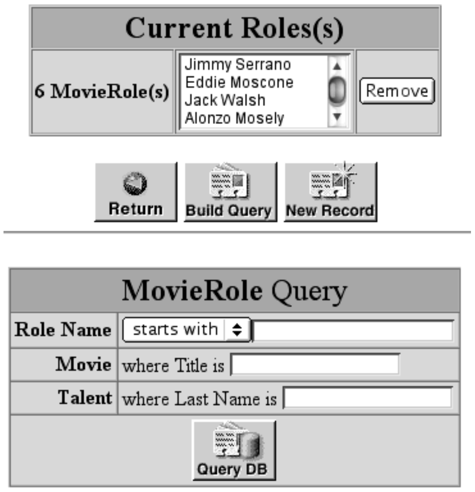
Figure 5-6 An edit page



Budget: 23000000
Name: Universal Pictures
Movies:
16, 131, 1000000, Midnight Run, Action
16, 144, 1000000, Uncle Buck, Comedy
16, 149, 300000, Fletch, Comedy
16, 166, 400000, Jurassic Park, Action

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

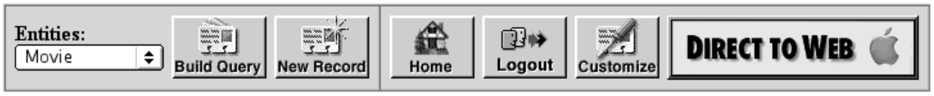


6 MovieRole(s):
Jimmy Serrano
Eddie Moscone
Jack Walsh
Alonzo Mosely

Role Name: starts with
Movie: where Title is
Talent: where Last Name is

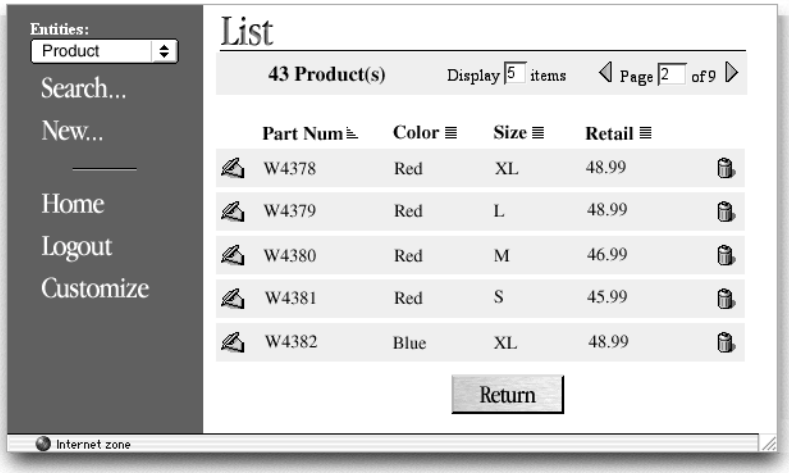
With the exception of the login page, every Direct to Web page has an area containing a menu and buttons that assist in navigating around the application (Figure 5-8). This is called the **menu header**.

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 53)).

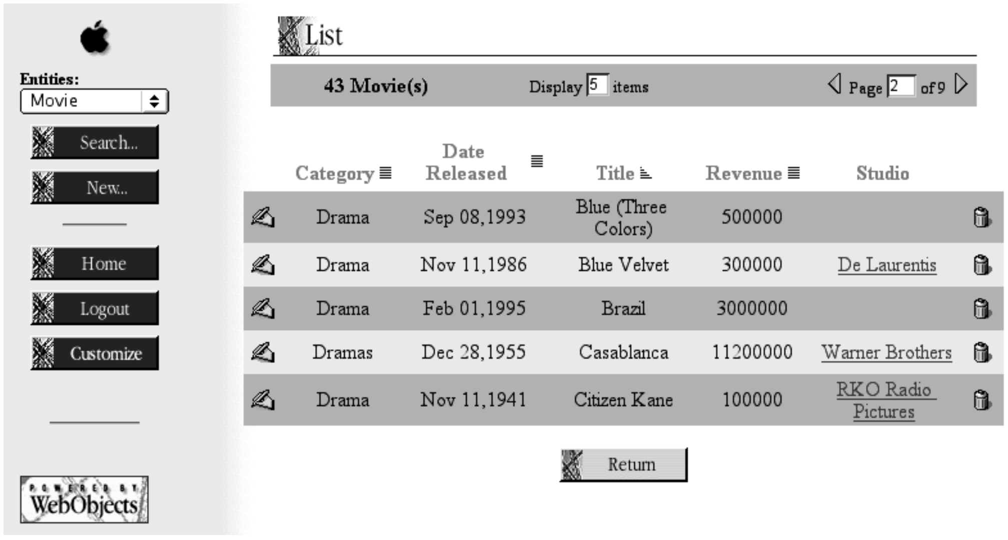
Figure 5-9 An example Neutral look page



43 Product(s)

Part Num	Color	Size	Retail
W4378	Red	XL	48.99
W4379	Red	L	48.99
W4380	Red	M	46.99
W4381	Red	S	45.99
W4382	Blue	XL	48.99

Figure 5-10 An example WebObjects look page



	Category	Date Released	Title	Revenue	Studio	
	Drama	Sep 08,1993	Blue (Three Colors)	500000		
	Drama	Nov 11,1986	Blue Velvet	300000	De Laurentis	
	Drama	Feb 01,1995	Brazil	3000000		
	Dramas	Dec 28,1955	Casablanca	11200000	Warner Brothers	
	Drama	Nov 11,1941	Citizen Kane	100000	RKO Radio Pictures	

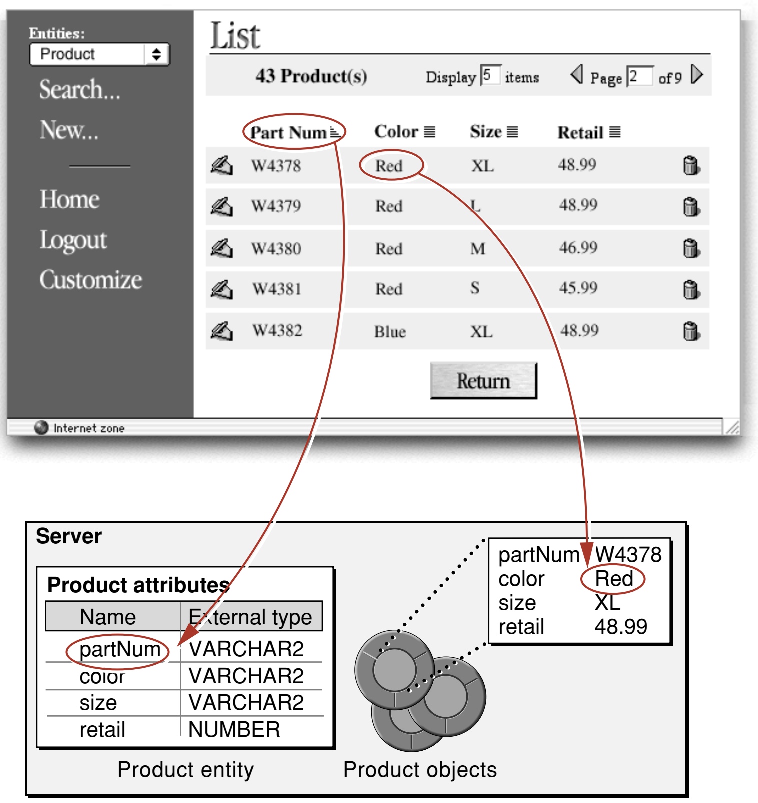
How Direct to Web Works

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 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



	Part Num	Color	Size	Retail	
	W4378	Red	XL	48.99	
	W4379	Red	L	48.99	
	W4380	Red	M	46.99	
	W4381	Red	S	45.99	
	W4382	Blue	XL	48.99	

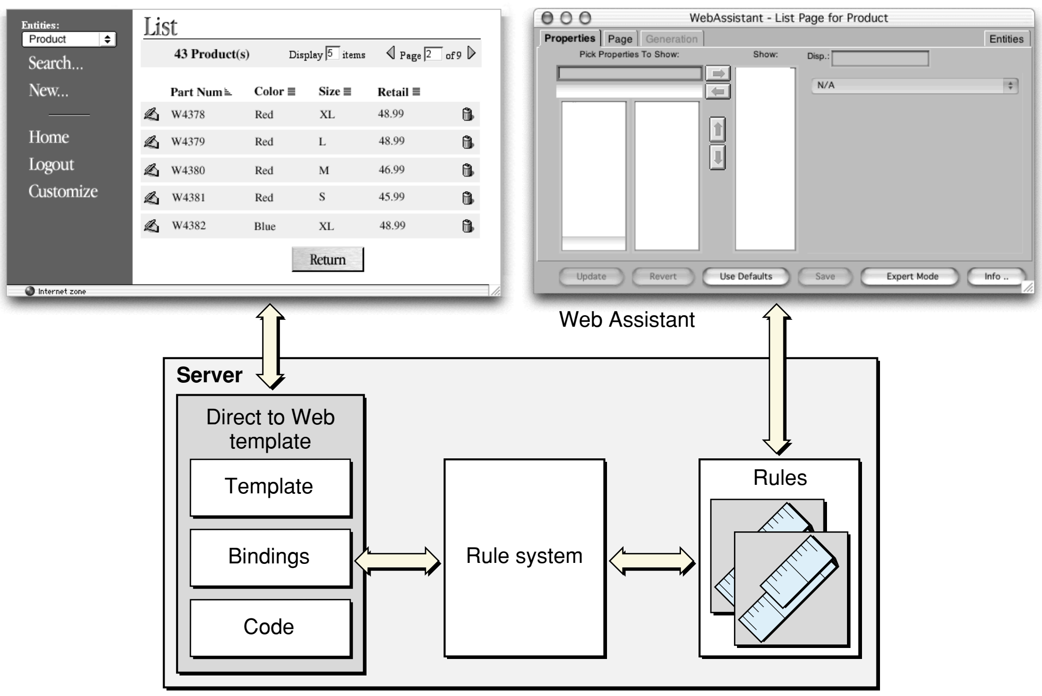
Product attributes

Name	External type
partNum	VARCHAR2
color	VARCHAR2
size	VARCHAR2
retail	NUMBER

partNum	W4378
color	Red
size	XL
retail	48.99

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 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



43 Product(s) Display 5 items Page 2 of 9

Part Num	Color	Size	Retail
W4378	Red	XL	48.99
W4379	Red	L	48.99
W4380	Red	M	46.99
W4381	Red	S	45.99
W4382	Blue	XL	48.99

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:

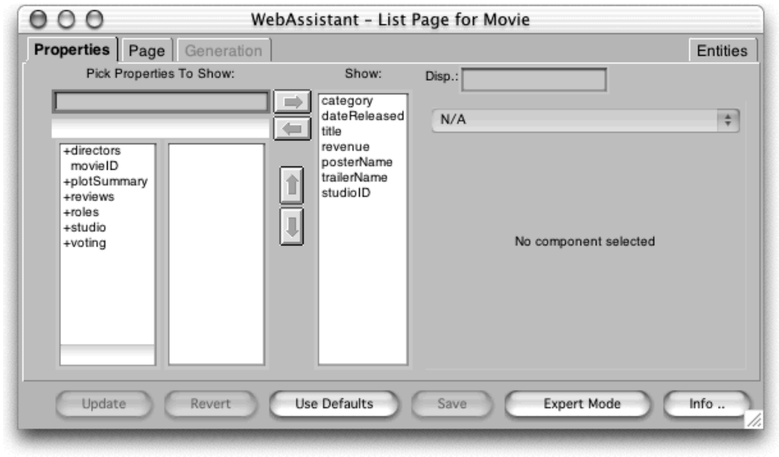
- Create a Direct to Web project using Project Builder.
- Create a model using EOModeler.
- Customize your Direct to Web application using the Direct to Web Assistant (optional).
- Further customize your Direct to Web application (optional).

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 5-13 (page 57) shows the Direct to Web Assistant in use.

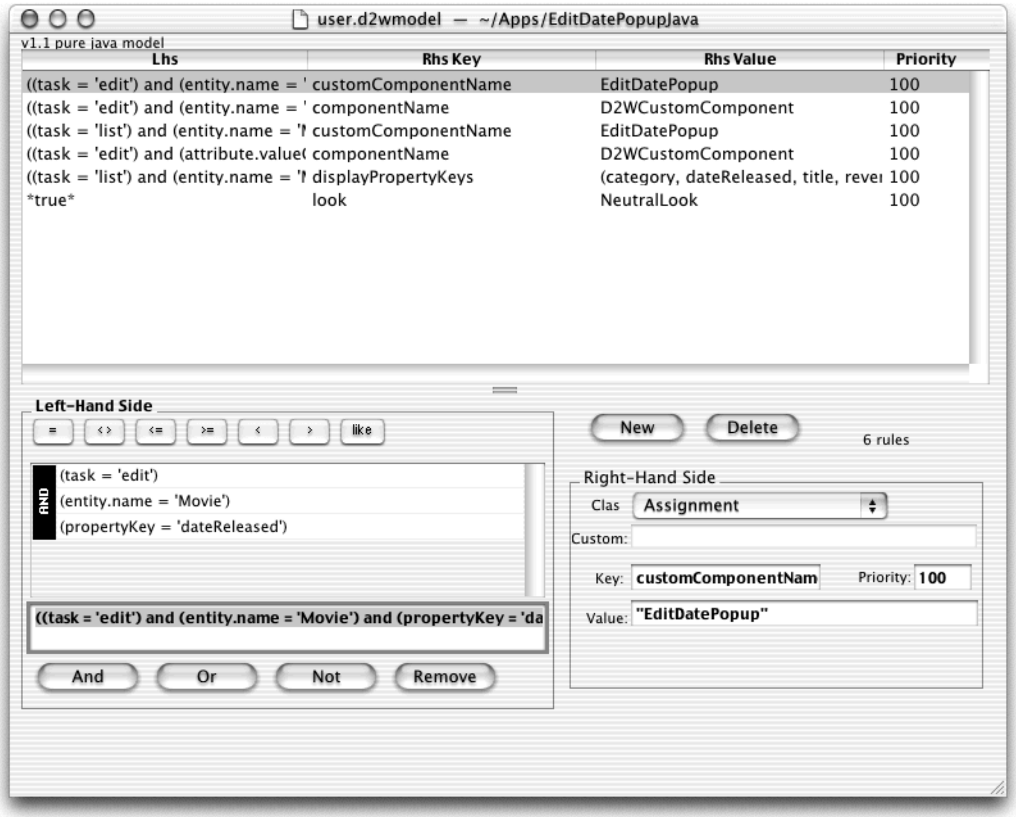
Figure 5-13 The Direct to Web Assistant



With the assistant, you can designate which entities are read-write, read-only, or hidden. 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.

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 5-14 shows the Rule Editor.

Figure 5-14 The Rule Editor


user.d2wmodel — ~/Apps/EditDatePopupJava

v1.1 pure java model

Lhs	Rhs Key	Rhs Value	Priority
((task = 'edit') and (entity.name = '	customComponentName	EditDatePopup	100
((task = 'edit') and (entity.name = '	componentName	D2WCustomComponent	100
((task = 'list') and (entity.name = 'I	customComponentName	EditDatePopup	100
((task = 'edit') and (attribute.value(	componentName	D2WCustomComponent	100
((task = 'list') and (entity.name = 'I	displayPropertyKeys	(category, dateReleased, title, reve	100
true	look	NeutralLook	100

Left-Hand Side

AND
(task = 'edit')
(entity.name = 'Movie')
(propertyKey = 'dateReleased')

((task = 'edit') and (entity.name = 'Movie') and (propertyKey = 'da

6 rules

Right-Hand Side

Clas: Assignment

Custom:

Key: customComponentNam Priority: 100

Value: "EditDatePopup"

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 a 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, 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 52) 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.
- 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.

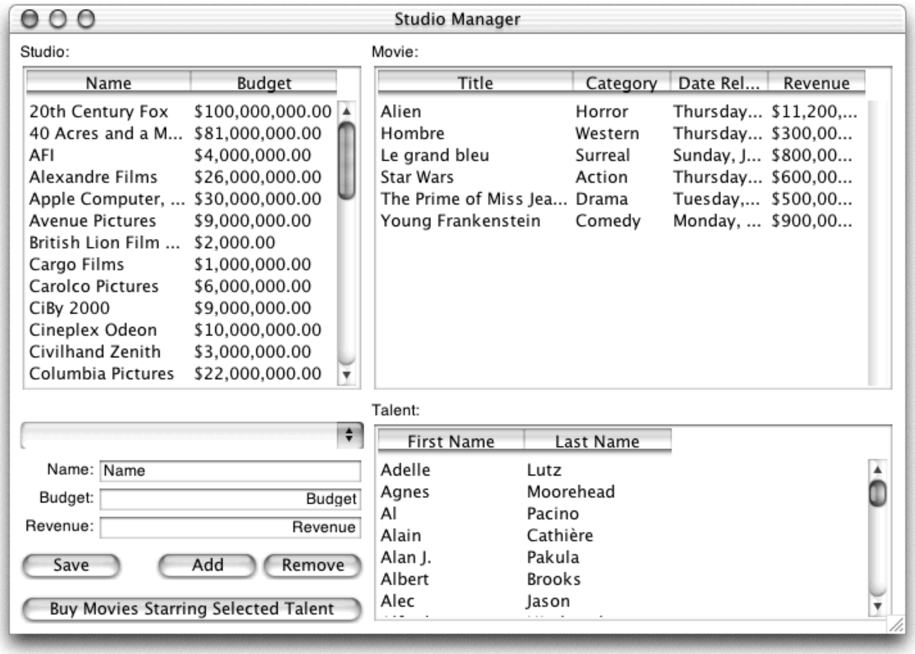
Direct to Web is particularly suited for mission-critical applications, prototypes, and intranet applications where development time is critical and the limitations that Direct to Web imposes on the flow and user interface are not an issue. Although you can customize the application, you first need to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

Java Client Applications

The Java Client approach allows you to replace the HTML user interface of a WebObjects application with a much more interactive and flexible one written in Java. 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



Name	Budget
20th Century Fox	$100,000,000.00
40 Acres and a M...	$81,000,000.00
AFI	$4,000,000.00
Alexandre Films	$26,000,000.00
Apple Computer, ...	$30,000,000.00
Avenue Pictures	$9,000,000.00
British Lion Film ...	$2,000.00
Cargo Films	$1,000,000.00
Carolco Pictures	$6,000,000.00
CiBy 2000	$9,000,000.00
Cineplex Odeon	$10,000,000.00
Civilhand Zenith	$3,000,000.00
Columbia Pictures	$22,000,000.00

Title	Category	Date Rel...	Revenue
Alien	Horror	Thursday...	$11,200,...
Hombre	Western	Thursday...	$300,00...
Le grand bleu	Surreal	Sunday, J...	$800,00...
Star Wars	Action	Thursday...	$600,00...
The Prime of Miss Jea...	Drama	Tuesday,...	$500,00...
Young Frankenstein	Comedy	Monday, ...	$900,00...

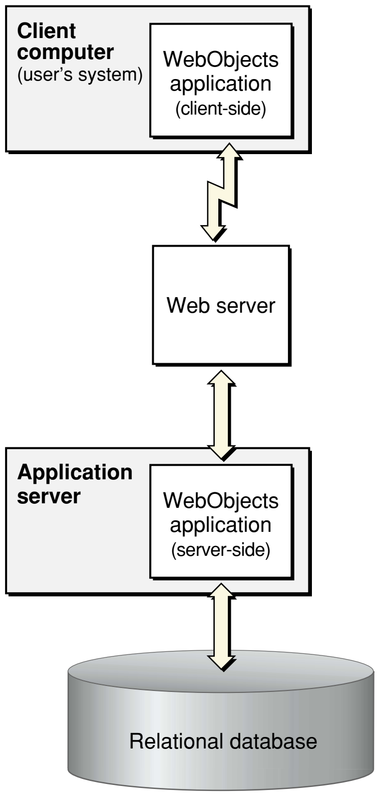
Name:	Name
Budget:	Budget
Revenue:	Revenue

First Name	Last Name
Adelle	Lutz
Agnes	Moorehead
Al	Pacino
Alain	Cathière
Alan J.	Pakula
Albert	Brooks
Alec	Jason

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 6-2. The server-side portion interacts with a database server and the client-side portion provides all or part of the application's user interface.

Figure 6-2 Java Client's distributed, multitier architecture

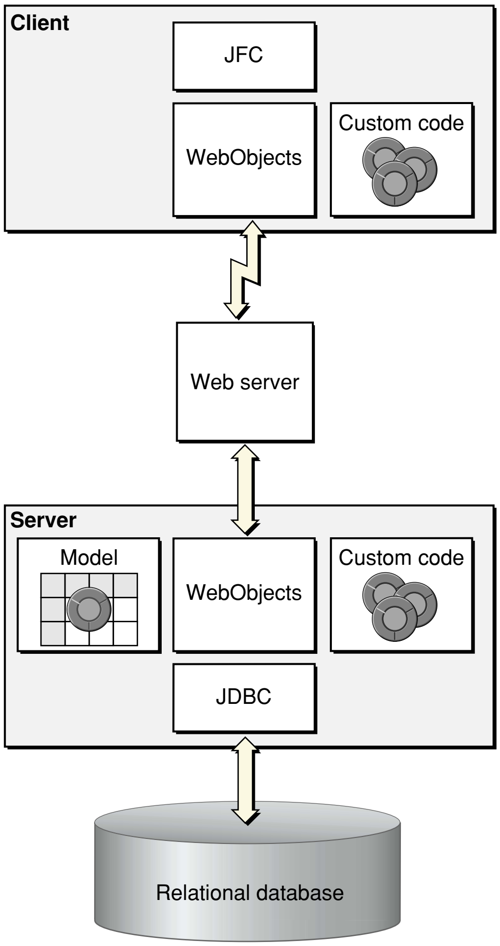


The client and server applications have duties other than merely providing the user interface and database access—for example, each can contain business logic and each communicates with the other through a Web server. However, the database access/user interface division is significant because it provides 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.

Managing the User Interface

Unlike in HTML-based applications, the WebObjects component of the server-side application doesn't have to provide any user interface management. Instead, the Java Client architecture moves part or all of the user interface management to a client application, as shown in Figure 6-3 (page 66). To support this division, 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.

Figure 6-3 Architecture of a Java Client application



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.

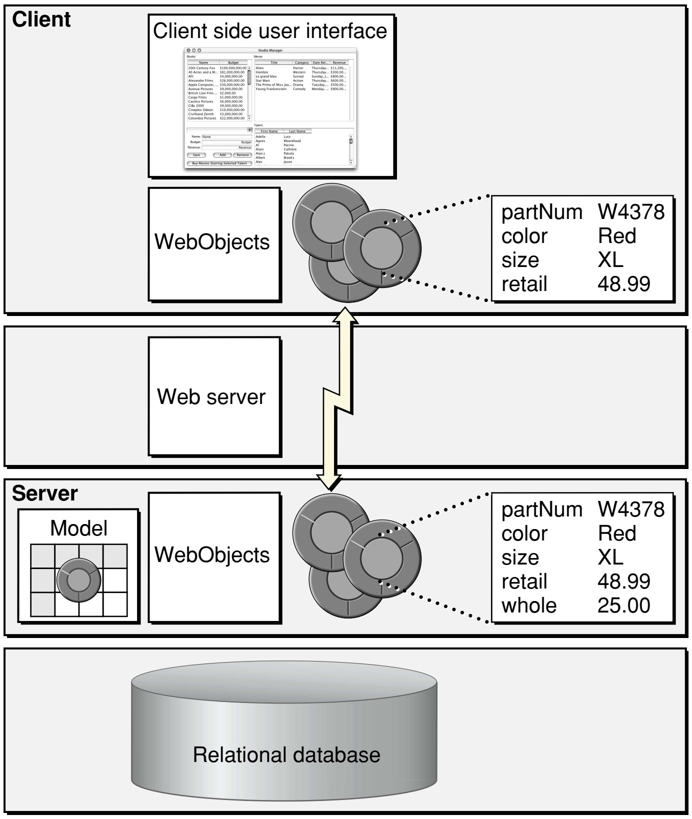
Data Synchronization Between Client and Server

Figure 6-4 (page 68) 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.

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 6-4 Data flow in a Java Client application



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 6-4, 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.

Other Architectures

There are other Java-based architectures besides Java Client that are also distributed and mutiltier. This section describes how Java Client compares with the Client JDBC and JDBC three-tier architectures.

Client JDBC Architecture

Client JDBC applications use the same "fat-client" architecture that desktop applications do. 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 shortcomings of this type of architecture are typical of all fat-client architectures. Security is a problem because the compiled Java on the client is easily decompiled, leaving both sensitive data and business rules at risk. The server has to be open to allow all client operations without being able to control what the client is doing. In addition, such an architecture doesn't scale; it is expensive to move data over the channel to the client.

Java Client has none of these problems. Sensitive data and business rules can be confined to the server, so the server doesn't have to allow indiscriminate access to data and operations. Additionally, because the client part of a Java Client application contains nonsensitive data and business logic, it doesn't make nearly as many round trips to the server or move as much data back and forth.

JDBC Three-Tier Architecture

A JDBC three-tier application (with CORBA as the transport) is a big improvement over Client JDBC. In this architecture the client can be thin because all that is required on the client side is the JFC, non-sensitive custom code (usually for managing the user interface), and CORBA stubs for communicating with the server. Sensitive business logic as well as logic related to database connection are stored on the server. In addition, the server handles all data-intensive computations.

Although the JDBC three-tier architecture is an improvement over Client JDBC, it has its own weaknesses.

- The JDBC three-tier architecture results in too much network traffic. Because this architecture uses "proxy" business objects on the client as handles to the real objects on the server, each client request for an attribute is forwarded to the server, causing a separate round trip and precipitating a "message storm."
- The JDBC three-tier architecture 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.
- The JDBC three-tier architecture does not provide much of the functionality associated with application servers, such as application monitoring and load balancing, nor does it provide HTML integration.

Java Client addresses these problems as well. First, instead of using proxy business objects on the client, the Java Client application makes use of "copies" of the objects. Second, WebObjects provides all of the database access and user interface synchronization with its enterprise objects technology. Finally, it provides application server functionality and HTML integration.

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.
- 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

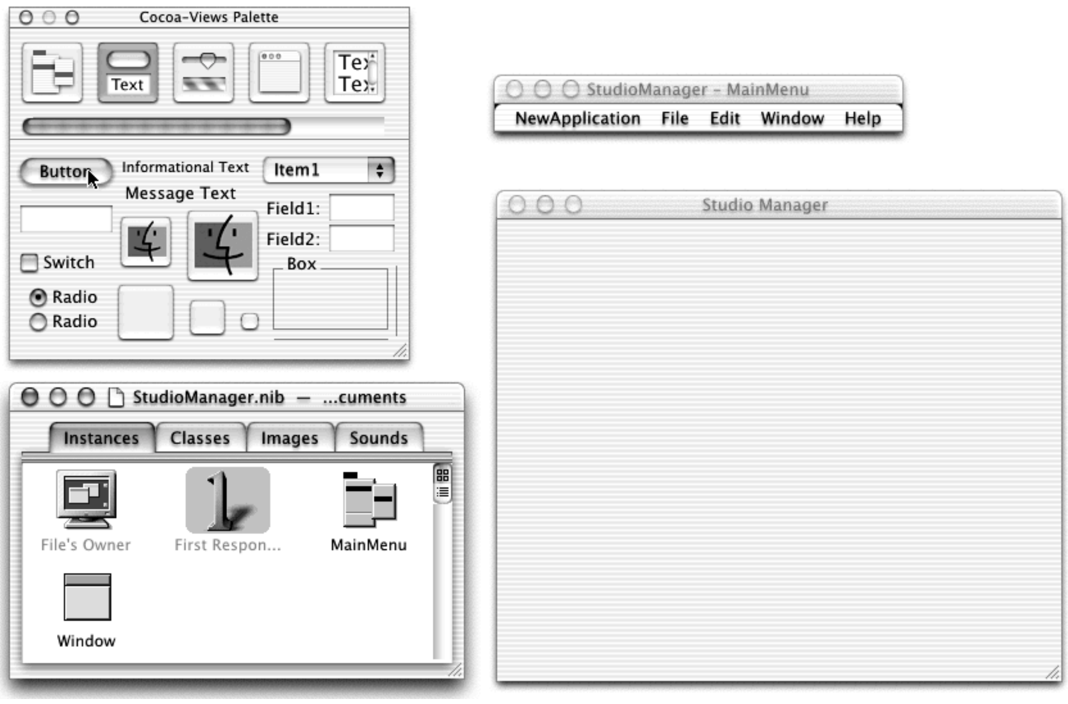
A Java Client WebObjects application gives you considerable flexibility in how you compose its user interface. Ideally you provide an application's entire user interface in a single Java application that runs on the client. But you can also combine Java Client applets and static and dynamic (WebObjects) 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 35)). 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 6-5. 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 6-5 Composing a user interface with Interface Builder



Advantages of the Java Client Approach

Java Client applications have much in common with traditional desktop applications. They also have much in common with HTML-based Web applications. Correspondingly, Java Client applications have some of the best features from each:

- **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.
- **Rich user interfaces**. Like a desktop application, a Java Client application can draw upon user-interface frameworks that provide endless possibilities for user interface widgets. Additionally, a Java Client application can have multiple windows open at once. HTML applications have a more rigid workflow, allowing only one page of the user interface to be available at a time.
- **Portability**. Like any Web application, a Java Client application is portable. A Web application can run on any client browser that implements certain standards and protocols. Because a Java Client application is written in Pure Java, it too runs almost anywhere. The client need only have a compatible Java virtual machine (VM), something that major operating systems include as a standard feature.
- **Security**. Because sensitive data and business logic are confined to the server in Web applications, they are more secure than traditional client-server applications.

Limitations

Java Client applications compare very favorably to traditional desktop applications and other distributed, multitier, Java-based architectures. They compare favorably to an HTML-based approach, too, but you trade some of the advantages of an HTML-based application for an improved user interface.

The client portion of a Java Client application can take the form of an application or an applet. The trade-offs are different with each. Providing the client portion as an application requires more system administration than using an HTML-based approach. HTML-based Web applications require no software installations on client systems. Users need only a Web browser to access the applications. With Java Client applications, however, the application should usually be pre-installed on client machines.

The alternative to providing the client portion as an application is providing it as an applet that runs in a browser. The system administration requirements of this approach are comparable to those of HTML-based Web applications: no client installations are required. Instead of running a preinstalled application, users transparently download an applet to a browser in which the applet runs. This download, however, is the disadvantage of using an applet. No matter how simple the Java Client user interface is, the size of the applet is considerable. Thus, it takes longer for a Java Client applet to start up than for a Java Client application; and the slower the client's connection is, the longer it takes for the applet to start up.

Guidelines for Choosing Java Client

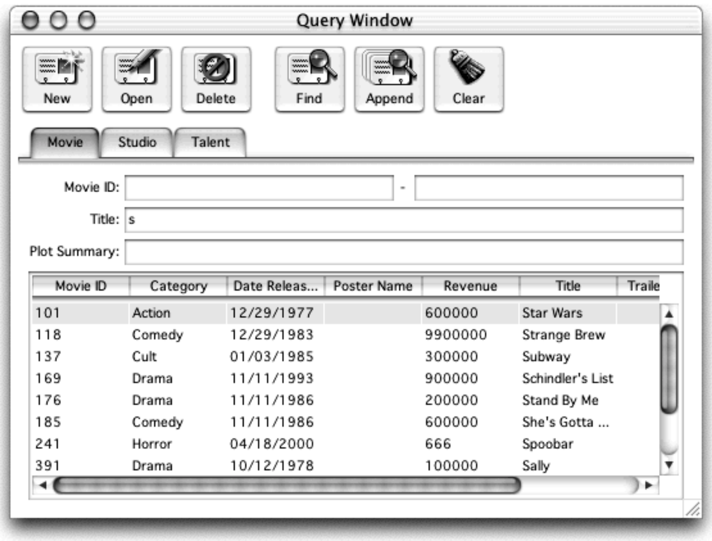
Some applications simply require the flexibility and interactivity of a Java Client user interface—an HTML-only solution just isn't acceptable. However, Java Client is rarely appropriate for mass markets and highly visible websites. Typically it's practical only in intranet environments.

As mentioned in the last section, Java Client applications have special deployment requirements because part of the application runs on the user's machine. 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, which can take a long time if the network is slow. If neither solution is acceptable, you need to use the WebObjects HTML-based approach or the Direct to Web approach.

Direct to Java Client Applications

The Direct to Java Client approach provides the richness of a Java Client application without as much work. Direct to Java Client is an add-on to Java Client that dynamically generates user interfaces. Instead of designing a user interface for your application, Direct to Java Client does it for you. Figure 7-1 shows the user interface of a typical Direct to Java Client application.

Figure 7-1 A sample Direct to Java Client application



Movie ID: -
Title: s
Plot Summary:

Movie ID	Category	Date Releas...	Poster Name	Revenue	Title	Traile
101	Action	12/29/1977		600000	Star Wars	
118	Comedy	12/29/1983		9900000	Strange Brew	
137	Cult	01/03/1985		300000	Subway	
169	Drama	11/11/1993		900000	Schindler's List	
176	Drama	11/11/1986		200000	Stand By Me	
185	Comedy	11/11/1986		600000	She's Gotta ...	
241	Horror	04/18/2000		666	Spoobar	
391	Drama	10/12/1978		100000	Sally	

The Basics

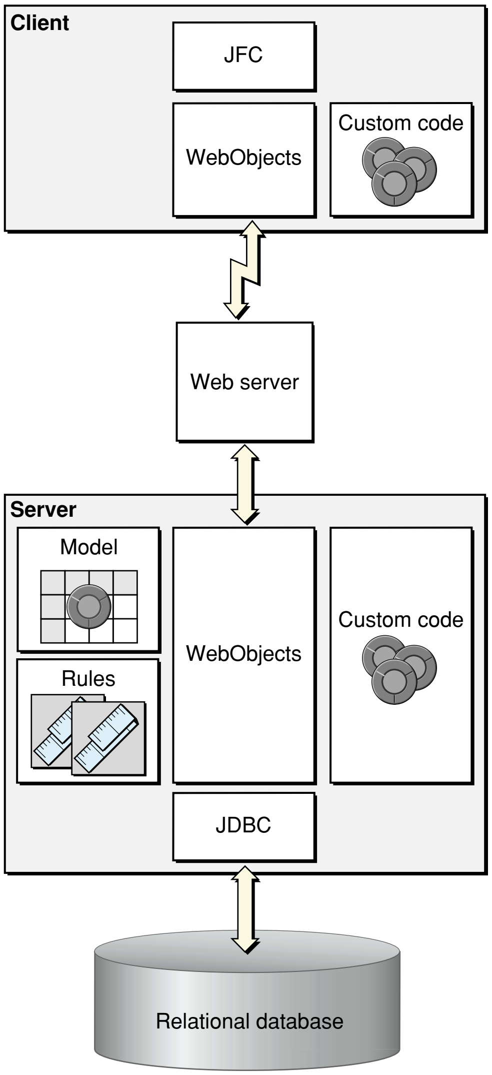
Direct to Java Client dynamically generates an application's user interface from model files. It creates windows for finding, creating, updating, and deleting objects that correspond to entities in the models. Based on model information, Direct to Java Client chooses appropriate widgets for representing the properties of objects.

There are numerous ways to customize the default user interface Direct to Java Client generates, so you can achieve almost any user interface you need.

Direct to Java Client Architecture

Direct to Java Client applications have the same basic architecture as Java Client applications do. The main difference is the addition of rule files that determine how Direct to Java Client assembles a user interface for your application.

Figure 7-2 The components of a Direct to Java Client application



Development Tasks and Tools

The tasks of creating a Direct to Java Client application are similar to writing a Direct to Web application. There are four steps:

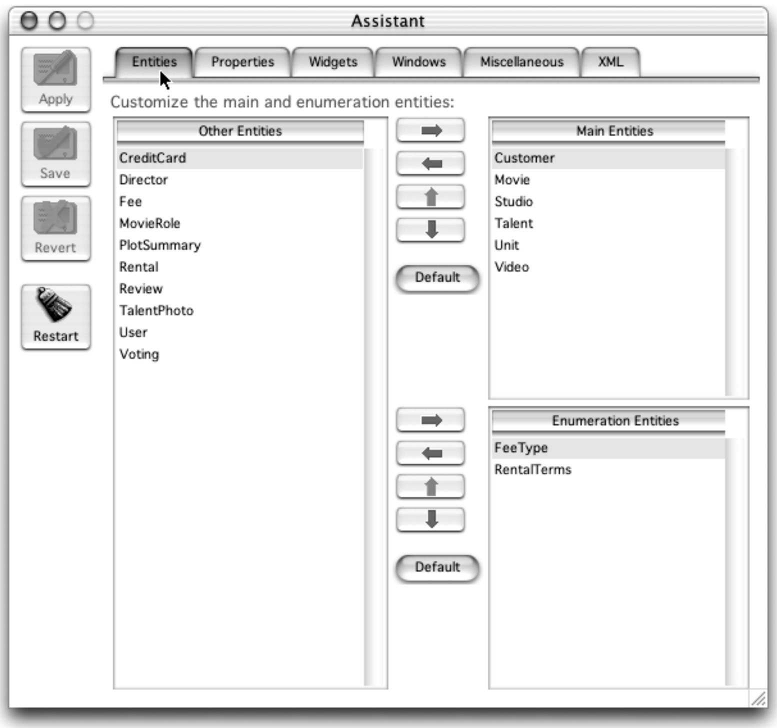
- Create a Direct to Java Client project using Project Builder.
- Create a model using EOModeler.
- Customize your application using the Direct to Java Client Assistant (optional).
- Further customize your application (optional).

Further information about the last two optional steps is given below.

The Direct to Java Client Assistant

The Direct to Java Client Assistant is the easiest way to customize a Direct to Java Client application. It is an easy-to-use tool that is integrated into a running client application. It allows you to perform the most common customizations, directly test them while the application is running, and save them in your project. The Direct to Java Client Assistant is very similar in concept to the Direct to Web Assistant even though the two assistants look and behave differently.

Figure 7-3 The Direct to Java Client Assistant



Further Customizing Your Application

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 56).

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.

Advantages of the Direct to Java Client Approach

A Direct to Java Client application is really just a specialized Java Client application, so it has the same advantages: a rich user interface, portability, and security.

In addition, Direct to Java Client creates your application like Direct to Web does. Thus you get the Direct to Web's advantages as well: reduced time to develop applications, increased stability, improved prototyping capabilities, reduced testing requirements, and the freedom to focus on the business logic instead of the user interface.

Limitations

Direct to Java Client applications share both the advantages and limitations of Java Client applications. Thus, Direct to Java Client applications have the disadvantage of requiring more system administration than HTML-based Web applications.

In addition, Direct to Java Client applications have the advantages and limitations of Direct to Web. The learning curve is flat until you need to customize your application; then it becomes very steep. You need to understand the layer of Direct to Java Client that generates the user interface in addition to the Java Client technology. Also, detailed widget placement (moving a text field three pixels to the right, for example) is harder to do with Direct to Java Client than it is with regular Java Client and Interface Builder.

Finally, Direct to Java Client applications are a little slower than static user interfaces. When you start up the application or open a new type of window, Direct to Java Client has to make round trips to the server to get the information it needs to assemble the user interface. It has numerous optimizations to make user interface generation fast, but it does incur a performance hit.

Guidelines for Choosing Direct to Java Client

Once you've decided an application will have a Java Client user interface, try to use Direct to Java Client to build the user interface. Direct to Java Client's advantages generally outweigh its disadvantages. At least prototype new applications with Direct to Java Client, and slowly transition to regular Java Client only as needed.

The only time you might choose not to use Direct to Java Client is when you know that your final application will have a significantly different user interface paradigm than the one Direct to Java Client uses by default.

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 83), "User Interface Requirements" (page 84), and "Rapid Development Considerations" (page 85) 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 86).

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 86) for more information.

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 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 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.

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 HTML*
- *Discovering Java Client*
- *Discovering Direct to Web*
- *Discovering Direct to Java Client*

These books are in preparation at the time of this writing.

Finally, because the creation of enterprise objects is independent of the approach you choose, and because there are numerous subtleties in the way you can implement your enterprise objects, a fifth companion volume (also in preparation) is dedicated to this subject: *Creating Enterprise Objects*.

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.

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.

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.

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.

Index