Inside WebObjects

WebObjects Overview

For WebObjects 5.2

November 2002

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

Simultaneously published in the United States and Canada.

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

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

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

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

Contents

Figures, Listings, and Tables

About This Document

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 Web components. WebObjects is extremely scalable and supports high transaction volumes.

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

Why Read This Document

This document is written for developers who want to start using WebObjects. However, anyone interested in WebObjects technology will benefit from reading this document.

This document 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.

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 document doesn't require it.

Because WebObjects provides several distinct approaches to developing applications, this document discusses them one by one, and compares their pros and cons to help you decide which approach is appropriate for your application. In addition, you can take advantage of Web services using these approaches, enabling a new and exiting avenue for interapplication communication.

This document has the following chapters:

- Chapter 2, "Introduction" (page 15), introduces the technologies of WebObjects and how they fit together.
- Chapter 3, "Enterprise Objects" (page 29), explains how Enterprise Object technology allows you to think of your data as entities with customizable behavior instead of database tables with rows and columns.
- Chapter 4, "Web Applications" (page 41), covers the approach that allows you to create applications with a Web browser–based user interface.
- Chapter 5, "Desktop Applications" (page 69), discusses the approach with which you can produce a graphical user interface application that runs on a client computer.
- Chapter 6, "Web Services Applications" (page 91), addresses the addition of Web service support in your applications, both by providing Web services and consuming them.
- Chapter 7, "J2EE Support" (page 115), explains how WebObjects implements some aspects of Sun's Java 2 Platform, Enterprise Edition (J2EE), including Enterprise JavaBeans, JavaServer Pages (JSP), servlets, and Java Naming and Directory Interface (JNDI).
- Chapter 8, "Choosing Your Approach" (page 119), summarizes the pros and cons of these approaches and outlines the process you should go through when deciding which approach or combination of approaches is appropriate for a particular application.
- Appendix A, "Document Revision History" (page 125), lists the revisions made to this document.

Further Investigations

This document serves as a starting point. It surveys the technologies of WebObjects without providing the details. This section lists sources of WebObjects information available to you.

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

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

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

To access the developer documentation on Windows 2000, you use WOInfoCenter. To access the WOInfoCenter, choose Start > Program > WebObjects > WOInfoCenter.

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

CHAPTER 1

About This Document

Introduction

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 applications that integrate existing data and systems. WebObjects allows you to build applications that leverage the connectivity that the Internet or an intranet provide using the client-server paradigm.

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.

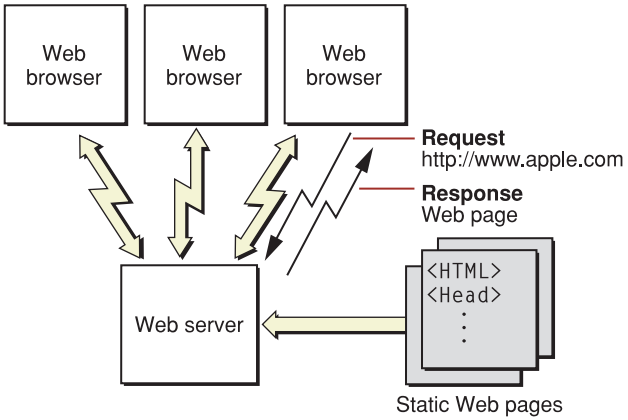
WebObjects allows you to take the next logical step. With it, you can produce full-fledged applications for use either across the Internet or within a corporate intranet. These applications can be Web-based, and thus accessible through a Web browser, or can have the full interactivity of a stand-alone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

Dynamic Web 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 the one in Figure 2-1. A user's Web browser requests pages using Uniform Resource Locators (URLs). 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 code. Using the HTML tags embedded within the file received from the HTTP server, the browser renders the page.

Figure 2-1 A static website

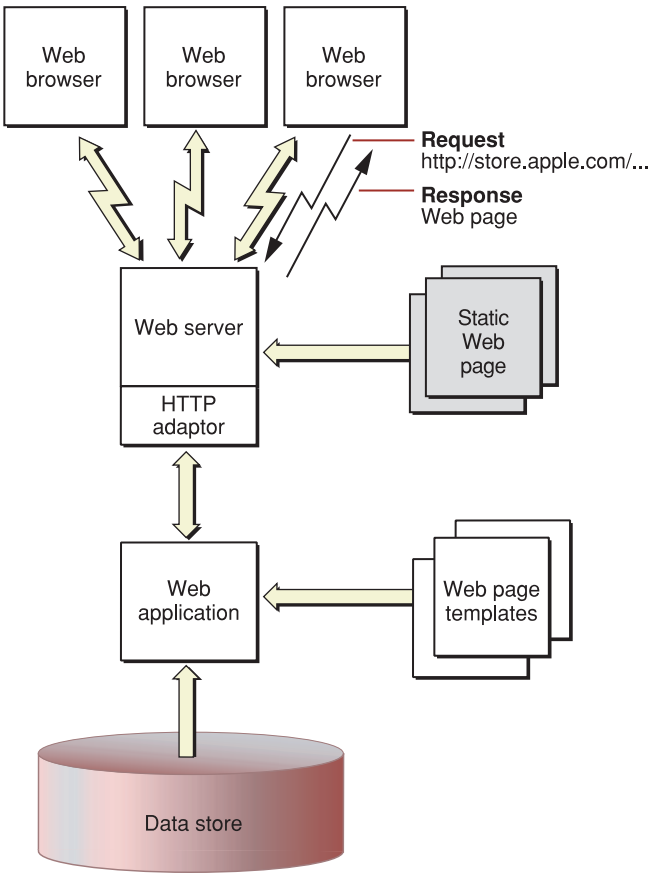


Static websites are easy to maintain. There are a number of tools in the market that allow you to create Web pages with a relatively small amount of effort. And, as long as the content of your pages doesn't change too often, it isn't difficult to keep them up-to-date. Dynamic websites, however, are a different situation. Without WebObjects, it would require many resources to stay up-to-date.

WebObjects is designed to allow you to quickly and easily publish dynamic data over the Web. You create Web page templates that indicate where on the Web page the dynamic data is to be placed. WebObjects fills in the content when the page needs to be generated in response to a request. The process is much like a mail merge. The information your applications publish can reside in a database or other data-storage medium or it can be 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 website. Again, the request (in the form of a URL) originates with a Web browser. If the Web server detects that the request is to be handled by a WebObjects application, it passes the request to an HTTP adaptor. The adaptor packages the incoming request in a form the WebObjects application can understand and forwards it to the application. Based upon Web page templates you define and the relevant data from the data store, the application generates a Web page that it passes back through the adaptor to the Web server. The Web server sends the page to the Web browser, which renders it.

Figure 2-2 A dynamic publishing website



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

Instead of using an HTTP adaptor, you can deploy applications through servlet containers. This approach allows you to take advantage of your servlet container's application deployment facilities. For more information on this approach, see "JavaServer Pages and Servlets" (page 117).

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 applications and data. Online stores selling books, music, or computers are examples of an Internet client-server application.

Client-server applications offer huge advantages over traditional applications. Users don't have to install the application on a client computer, which not only saves client disk space but ensures that the user always has the most up-to-date version of the application. Also, the client computers can be Macs, PCs, or anything that can run a Web browser with the necessary capabilities.

WebObjects allows you to develop two different types of Internet applications: Web applications and Java Client applications. Web applications are analogous to Common Gateway Interface (CGI) applications and consist of dynamically generated Web 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 complete user interface found in a more traditional desktop application.

Web Applications

When you need to develop a Web application, you can create it quickly and easily with the WebObjects development tools. WebObjects provides many elements that you can use to build your Web application's interface. These elements range from simple user interface widgets (for example, submit buttons, checkboxes, and tables) to elements that provide for the conditional or iterative display of content.

You can also define Web components. These are Web page templates that you can use to define your Web applications' design. Web components can contain any of the layout elements mentioned earlier as well as other Web components. For example, you can create a toolbar component that provides a link to your main website and to a search Web page. Then, as you create other components, you include the toolbar component in them. When you develop a support page for your website that you want all the other other components to use, the toolbar component is the only place you need to add a link to it.

Web components encapsulate more than the layout of a Web page. They also encompass a Java file that links the component's elements and subcomponents into a coherent entity. You put application-specific business logic in the Java class of a Web component.

For more information on Web applications, see "Web Applications" (page 41).

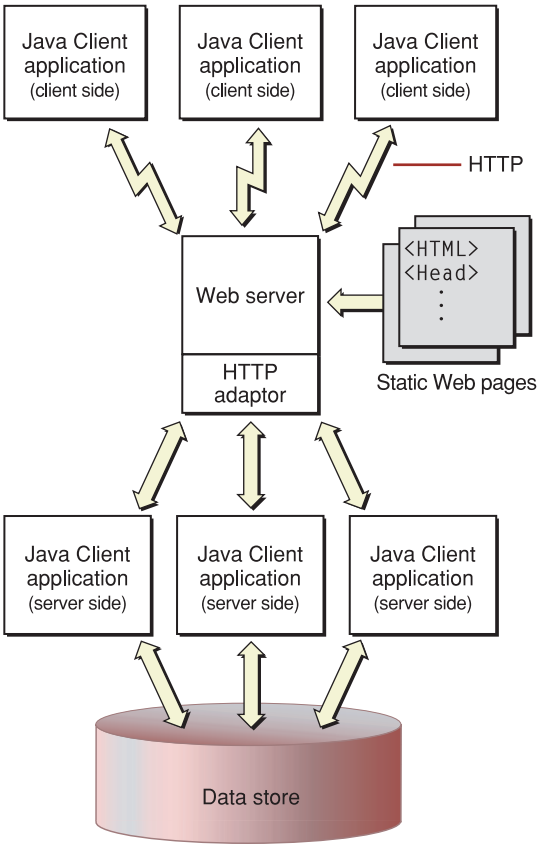
Desktop Applications

When you need the fast and full-featured 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 logic—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 multitier 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 illustrates a Java Client application in which the client portion is running as an application installed on the user's computer. Java Client applications, just like Web applications, can communicate with the application server using HTTP requests. 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 Java Client applications in action



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

For more information on desktop applications, see "Desktop Applications" (page 69).

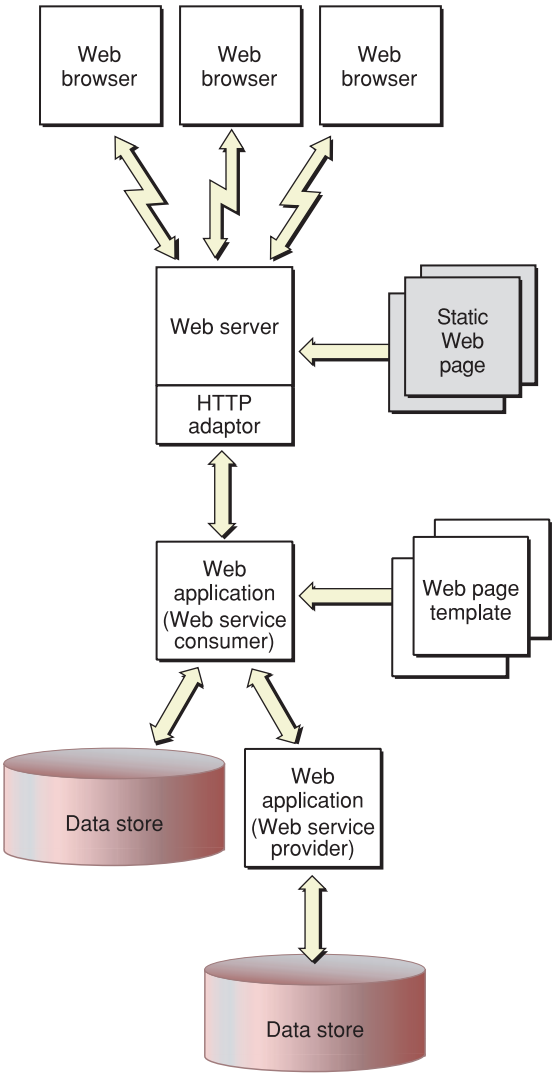
Web Services

Web services is an innovative implementation of distributed computing. WebObjects allows you to expose class methods as Web service operations. Web services provide an efficient way for applications to communicate with each other. Based on Simple Object Access Protocol (SOAP) messages that wrap Extensible Markup Language (XML) documents, Web services provide a flexible infrastructure that leverages the ubiquitous HTTP (or HTTPS) over TCP/IP. This means that your organization probably has all the hardware and software infrastructure needed to deploy Web services.

But Web services provide more than an information-exchange system. When an application implements some of its functionality using Web services, it becomes more than the sum of its parts. For example, you can create a Web service operation that uses a Web service operation from another provider to give its consumers (also known as *service requestors*) information tailored to their needs. Web service operations are similar to the methods of a Java class; a provider is an entity that publishes a Web service, while the entities that use the Web service are called consumers.

Web applications as well as Java Client applications can take advantage of Web services. Figure 2-4 shows a dynamic-publishing website that uses Web services.

Figure 2-4 A dynamic publishing website using Web services



For more information on Web service applications, see "Web Services Applications" (page 91).

Rapid Development

WebObjects provides power and flexibility. A certain degree of complexity, however, accompanies these features. For many applications, whether Web-based or Java Client–based, it's more important to develop the application quickly than strive for maximum customization. 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 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, Direct to Java Client, and Direct to Web Services.

These three technologies are similar in approach. Their primary difference is the type of application that gets generated by each: a Web application by Direct to Web, a Java Client application by Direct to Java Client, and a Web services application by Direct to Web Services. These technologies use a data model as the base upon which an application is created (in the case of Direct to Web Services, an application capable of providing or consuming Web services). In addition, they are useful not only for creating simple data-browsing applications or Web services, but in many situations can also serve as rapid prototyping tools. Because they allow customization on various levels, they are well-suited for creating your critical applications.

Direct to Web

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

Direct to Web uses information from a data model to dynamically generate Web pages. Consequently, you can modify your application's configuration at runtime—using the Web Assistant—to hide objects of a particular class, hide their properties, reorder properties, and change the way they are displayed without recompiling or relaunching the application.

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

A Direct to Web application is highly customizable. For example, you can change the appearance of the standard templates, mix Web components with Web pages generated by Direct to Web, and create custom Web components and Direct to Web templates that implement specialized behavior.

For more information on Direct to Web, see "The Direct to Web Approach" (page 51).

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 provides an assistant that allows you to modify your applications at runtime. But the applications produced by Direct to Java Client have rich desktop-class user interfaces. In addition, Java Client applications can take advantage of the processing power of the client computer to perform operations such as sorting a list of items received from the server.

For more information on Direct to Java Client, see "Two Approaches to Java Client" (page 75).

Direct to Web Services

Direct to Web Services allows you to create a Web service that lets its clients access data in your data store by invoking Web service operations. Although this approach is similar to Direct to Web and Direct to Java Client in its use of a data model and rule sets, the target users for Web service applications are other applications, not people.

You use the Web Services Assistant to determine which data entities are accessible by your Web service clients and the type of operations they can execute on them, such as search, insert, delete, and update. You accomplish this without writing a single line of code.

For more information on Direct to Web Services, see "Direct to Web Services" (page 104).

The WebObjects Advantage

WebObjects provides 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, accessing the data itself could be difficult. Some products rely on manually created or assistant-generated Structured Query Language (SQL) code, 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. Enterprise Objects 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 store. You can modify schemas, add or change databases, or even use a totally different storage mechanism without needing to rewrite your application.

JDBC is an interface between Java platforms and databases. WebObjects applications can access any database with a JDBC Type 2 or JDBC Type 4 driver.

Separation of Presentation Logic, Business Logic, and Data

An ideal Web application–development system simplifies maintenance and encourages code reuse by enforcing a clean separation of presentation (Web pages), logic (Java code), and data (data store). This modularity is inherent in the WebObjects programming model, which uses reusable Web components to generate Web pages directly from enterprise-object instances without the need to embed scripts or Java code inside the Web pages themselves. A Web component

contains a Web page template, which you—or a professional Web page designer—can design and edit using standard Web page authoring tools. A component can also implement custom behavior using a separate Java source file. Neither the template nor the Java source file includes data-model–specific information.

State Management

The HTTP protocol used on the Web is inherently stateless; that is, each HTTP request arrives independently of earlier requests, and it is up to Web applications to determine which of the active users sent it. 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 cart, associated with each user session.

Without using cookies, WebObjects provides objects that allow you to maintain information for the life of a particular user 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 customer buying patterns and then highlight items a particular buyer is likely to be interested in the next time she visits 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 Project Builder, an integrated development environment (IDE) that manages your Java business logic, tracks data models, Web components, and supporting files. As mentioned earlier, WebObjects also includes powerful assistants and frameworks that allow the rapid creation of Web or Java Client applications directly from the database. Advanced developers can tap into the WebObjects API, allowing virtually unlimited customization and expandability.

Pure Java

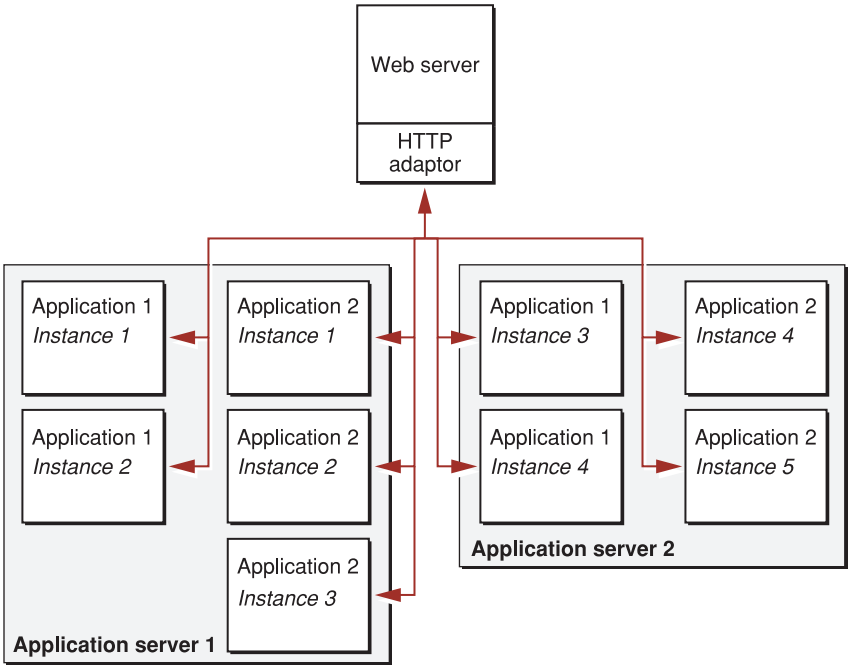
WebObjects applications are 100% Pure Java, which means they can be deployed on any platform with a certified Java 2 virtual machine.

Scalability and Performance

Static websites and traditional client-server applications have one advantage: 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 a set of static Web pages. Dynamic applications, although a tremendous advance over static websites, require additional server power to quickly access the changing data and construct the Web pages or Java Client user interface.

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 2-5). You can choose from one of several load-balancing algorithms (or create your own) to determine which application instance each new user should connect to. And, either locally or from a remote location, you can analyze site loads and usage patterns and then start or stop additional application instances as necessary. Load balancing is a very powerful feature of WebObjects that allows you to add more server capacity as the need arises without needing to implement a load-balancing algorithm yourself.

Figure 2-5 Multiple instances of two applications



CHAPTER 3

Enterprise Objects

As mentioned in "Introduction" (page 15), WebObjects applications gain much of their usefulness by interacting with a persistent data store, which is usually a database. In WebObjects, database tables are represented as collections of Java classes called **enterprise-object** classes. Enterprise-object classes contain the bulk of your application's **business logic**, the part of the application you write regardless of which application-development approach you take. For details on the WebObjects application-development approaches, see "Choosing Your Approach" (page 119).

This chapter introduces enterprise-object classes, describes how they map to a database, outlines how WebObjects supports and interacts with them, and presents the advantages of the Enterprise Objects approach over other techniques. If you are not acquainted with relational-database operations, you may read the sections "What Is an Enterprise Object Class?" (page 29), and "The Enterprise Objects Advantage" (page 38), and skip the rest of the chapter, which is written for those familiar with relational databases.

For detailed information on Enterprise Object technology, see *Inside WebObjects: Enterprise Objects*.

What Is an Enterprise Object Class?

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

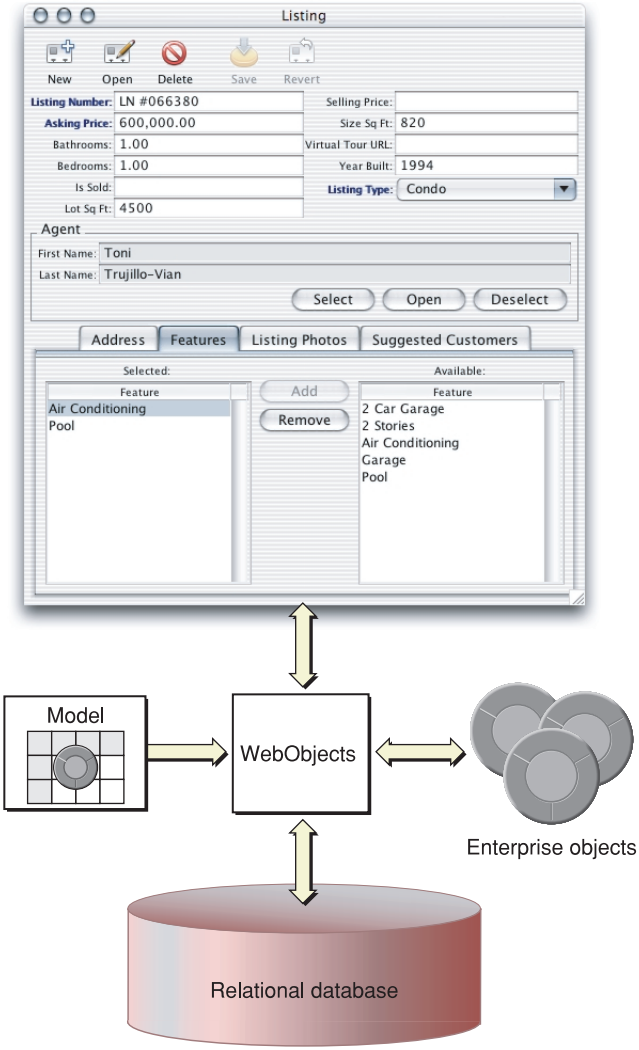
- It has properties that map to stored or persistent data; an enterprise-object instance typically corresponds to a single row or record in a database table.
- It knows how to interact with other parts of WebObjects to give and receive values for its properties. This is done through a mechanism known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

In addition to providing classes that manage a set of enterprise-object instances in memory, Enterprise Objects defines an API to which enterprise-object classes must conform, as well as default implementations of essential methods. 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 enterprise-object classes, they shouldn't embed knowledge of the user interface or database. For example, if you embed knowledge of your user interface, you can't reuse the classes because each application's user interface is different. Similarly, if you embed knowledge of your database, you have to update your classes every time you modify the database's schema.

If not in the enterprise-object classes, where does this knowledge go? It's handled by using a data model, as shown in Figure 3-1.

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



Enterprise Objects provides a database-to-objects mapping, called a **model**, which allows enterprise-object classes to be database-engine agnostic. WebObjects has facilities that let you map the properties of enterprise-object classes to user interface elements, providing these classes independence from user interface details.

This approach allows you to create libraries of enterprise-object classes that can be used in as many applications as you need, with any user interface, and with any database engine. Therefore, you can concentrate on coding the logic of your business while WebObjects takes care of the rest.

For example, you could create an entity called Customer that defines such business rules as *customers must have a work or home phone number*, or *customers cannot spend more than their credit limit*. Without rewriting your business logic, you could use these objects in a publicly available application and an internal customer service application. You could also switch the database that hosts 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 are reusable, distributed, persistent, and portable to a variety of platforms.

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

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

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

From the perspective of this paradigm, enterprise-object classes are model objects. However, WebObjects also extends the MVC paradigm. Enterprise-object classes are also independent of the persistent storage mechanism used to store instances of them. Enterprise-object classes do not need to know anything about the database that holds their data, and the database doesn't need to be aware of them, either.

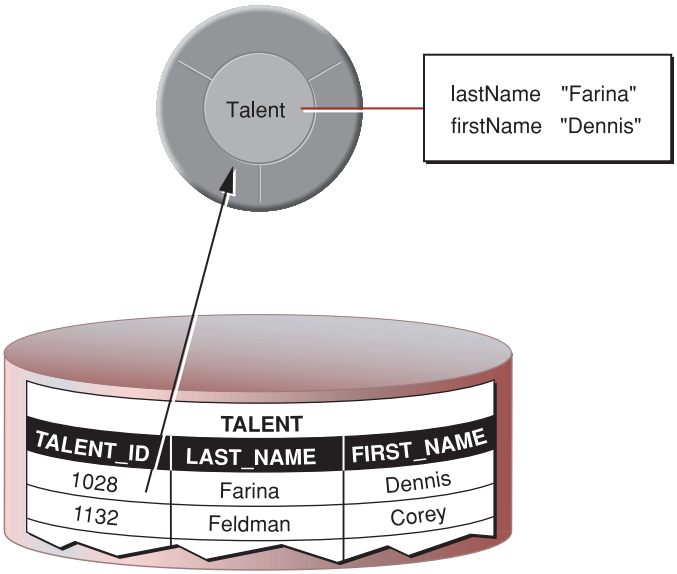
Mapping Data Entities to Database Tables

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

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

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

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

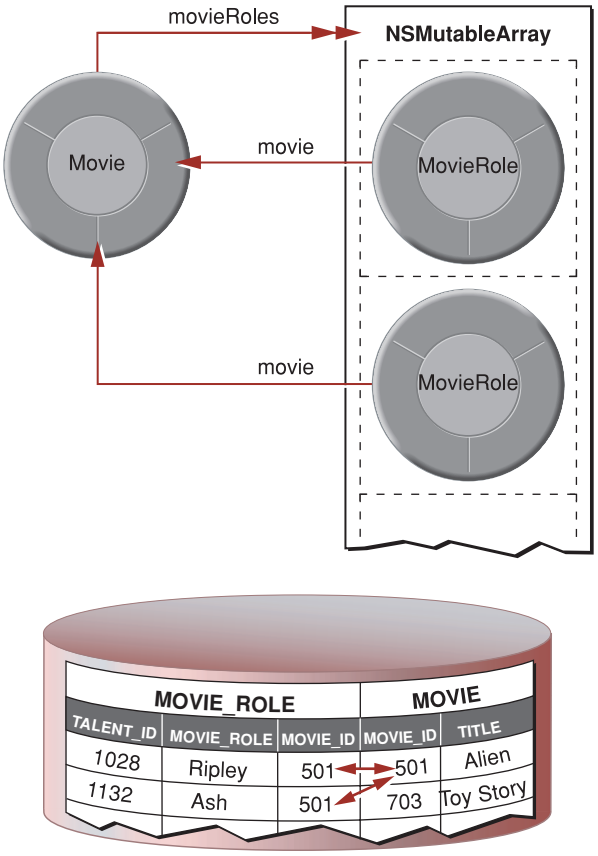


In practice, the mapping is more flexible than this. For example:

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

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

Figure 3-3 Mapping relationships



WebObjects Support for Enterprise-Object Instances

After an application has accumulated changes to enterprise-object instances and invokes the `saveChanges` method, WebObjects analyzes the instances, generates the necessary database operations (SQL statements), and executes those operations to synchronize the database with the enterprise-object instances held in memory.

Enterprise Objects maintains the integrity of your data as it is transferred between your application and the data store, without sacrificing performance or flexibility, by using the following techniques:

Validation

A substantial part of your application's business logic is usually devoted to data validation (for example, verifying that customers don't exceed their credit limits, the return dates of DVDs 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 data store.

Referential-integrity enforcement

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

Automatic primary-key and foreign-key generation

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

Transaction management

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

Locking

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

Faulting

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

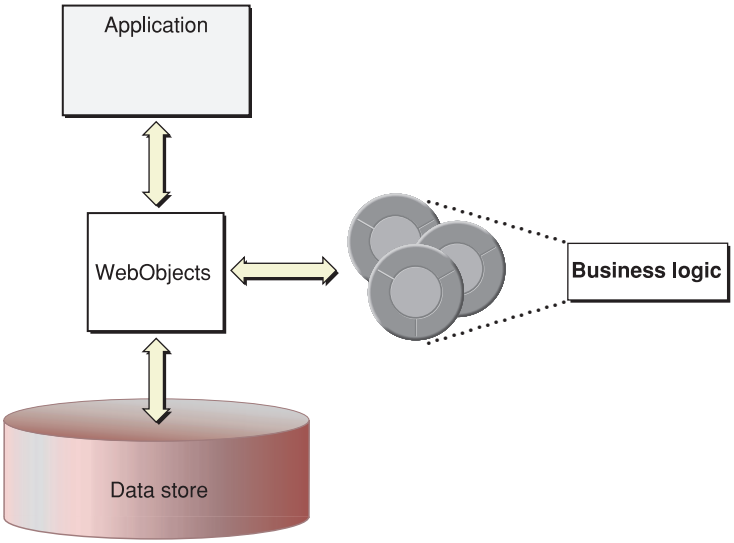
Uniquing

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

The Enterprise Objects Advantage

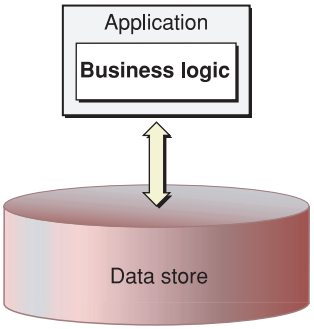
A hallmark feature of WebObjects, especially in comparison to other solutions, is the separation of the business logic from the database and the user interface. In WebObjects, you put the business logic in the enterprise-object classes, as shown in Figure 3-4.

Figure 3-4 Implementing business logic in enterprise-object classes



Another approach (Figure 3-5) is to implement business logic in the application.

Figure 3-5 Implementing business logic in the application

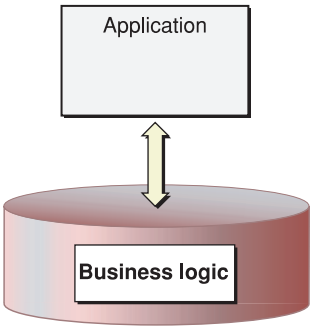


The Enterprise Objects approach betters this approach in the following ways:

- **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 mix the business logic code that deals with data entities with the code that manages an application's user interface.
- **It's more maintainable.** With WebObjects, you don't have to duplicate your business logic. Thus, you can make substantial changes to your business rules easily 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.** With 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.** With WebObjects, you can improve your application's performance without having to provide its users with faster systems. Instead, you can simply move some computation-intensive processing to fast computers.

A different approach (Figure 3-6) is to implement your business logic in the database—with stored procedures, rules, constraints, and triggers, for example.

Figure 3-6 Implementing business logic in the database



The Enterprise Objects 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 database changes, which prevents the user from receiving immediate feedback. WebObjects applications show changes to the user immediately, but the database is accessed only when saving these changes or fetching additional data.
- **It improves back-end portability.** Database vendors have different ways of implementing logic. If you have to support more than one database and you're using WebObjects, you don't have to implement the logic multiple times and, thus, suffer maintenance problems.
- **Java is a good development language.** With WebObjects, you program in Java, an industrial-strength, object-oriented language. The programmable variants of SQL usually have some object-oriented features, but are essentially procedural languages.

CHAPTER 4

Web Applications

The Web application approach allows you to create applications that dynamically generate Web pages based on Web page templates. WebObjects provides graphical tools, user interface elements, and a set of extensible frameworks with which you can develop elaborate applications. This chapter describes how you develop a Web application project, the advantages of using this approach, and what the development process is like. It also covers Direct to Web, the rule-based application-development approach that generates Web pages based on a data model and the tasks the user is allowed to perform.

For an in-depth discussion of Web application development, see *Inside WebObjects: Web Applications* and *Inside WebObjects: Developing Applications With Direct to Web*.

Web Applications—A Programmer's View

The following features of WebObjects ease the development of Web applications:

- **The presentation code and the business logic of Web components reside in separate files.** A **Web component** represents a Web page and consists of separate files for presentation logic (HTML file) and business logic (Java class).
- **Your presentation code remains separate from your business logic.** Enterprise-object classes, discussed in "Enterprise Objects" (page 29), contain all your business logic. This allows you to reuse business logic in multiple Web pages and even different applications.
- **WebObjects provides dynamic versions of static HTML elements.** These are called **dynamic elements**.

- **You can reuse HTML and Java code.** Web components can be embedded within other Web 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).

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

Separating Presentation Code From Event Handling Logic

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

A Web component consists of these files:

- **A Web page template that specifies how the component looks.** This file can be edited by any HTML editor or text editor.
- **Event-handling logic that specifies how the component acts.** You specify this with a standard Java source file.
- **Bindings that associate the component's presentation (HTML code) with its event-handling methods.** These bindings are stored in a WebObjects data (WOD) file, which uses a simple, text-based format.

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

You do not need to edit all three files separately. WebObjects Builder, a graphical Web component-editing application, allows you to edit the HTML, WOD, and Java files simultaneously, relieving you of having to manually synchronize them. WebObjects Builder is described in more detail in "WebObjects Builder" (page 49).

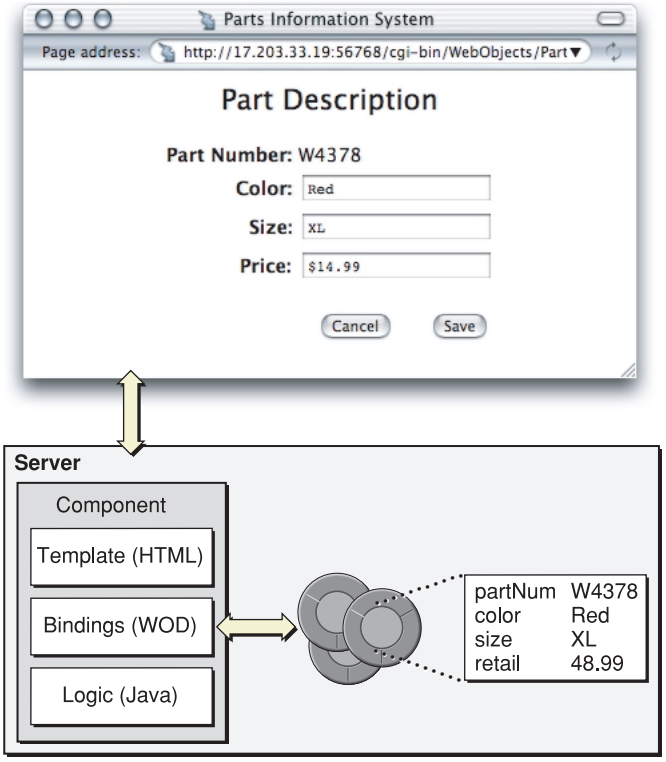
Figure 4-1 shows the three files in an example Web component.

Figure 4-1 The files of a Web component

Separating Presentation Code From Business Logic

Enterprise-object classes encapsulate an application's business logic and provide an interface to a data store. Since enterprise-object instances are regular objects, they can appear as variables in Web components, sessions, or the application object. A component's bindings (WOD) file relates the component's enterprise-object instances to its dynamic elements. Figure 4-2 shows how an enterprise-object instance is related to a Web component in a Web application.

Figure 4-2 How enterprise-object instances relate to a Web component



Dynamic Elements

The Web page template file in Figure 4-1 looks like any other HTML file except for the `WEBOBJECT` element. In this example, the element represents a **dynamic element**. Dynamic elements are the essential building blocks of a Web application. They link user interface elements with data and behavior. A dynamic element appears in the

template as a <WEBOBJECT> tag with a corresponding </WEBOBJECT> tag. Some dynamic elements have no HTML counterpart; WORepetition and WOConditional are examples. Table 4-1 lists some of the commonly used dynamic elements.

Table 4-1 Some dynamic elements

Element Name	Description
WOBrowser	Selection list that displays multiple items.
WOCheckBox	Checkbox.
WOConditional	Determines whether a portion of the Web component (or Web page) is generated.
WOForm	Container element that generates a fill-in form.
WOHyperlink	Generates a hypertext link.
WOImage	Displays an image.
WORadioButton	Toggle switch.
WORepetition	Container element that repeats its contents (that is, everything between the <WEBOBJECT...> and </WEBOBJECT...> tags in the Web page–template file) a given number of times.
WOResetButton	Button that clears a form.
WOString	Dynamically generated string.
WOSubmitButton	Submit button.
WOText	Multiline field for text input and display.
WOTextField	Single-line field for text input and display.

Reusing Web Components

You can embed a Web component within another Web component. For example, a component might represent only the header or the 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**, a shared

component, or a subcomponent. Like dynamic elements, reusable components appear in the Web page template as a `WEBOBJECT` element, allowing you to extend WebObjects's repertoire of dynamic elements.

WebObjects provides several reusable Web components like tables, radio button matrices, tabbed panes, and collapsible content. In addition, Direct to Web provides reusable components for editing, listing, selecting, inspecting, and querying enterprise-object instances.

Maintaining State

In addition to the Web components, a running Web application can have a number of sessions and an application object.

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

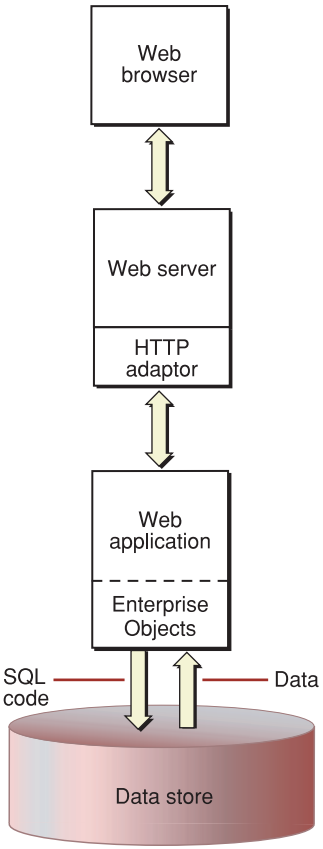
Session variables can be used in shopping cart applications to represent the items in the shopping cart. Email applications can use session variables to keep track of whether the user has logged in.

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

Web Application Architecture

When you run a Web application, it communicates with the Web browser using the process illustrated in Figure 4-3.

Figure 4-3 Web application communication chain



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

- **A Web browser.** WebObjects supports all Web browsers that conform to HTML 3.2. Of course, if your application uses more advanced features like JavaScript or QuickTime, the users' browsers must support those features.
- **A Web server.** WebObjects supports any Web 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 applications. In addition, you can deploy applications as servlets inside a servlet container. See "JavaServer Pages and Servlets" (page 117) for more information.

- **An HTTP adaptor.** The HTTP adaptor connects applications to the Internet or an intranet by acting as an intermediary between application instances and Web servers. Note that the HTTP adaptor may not be a separate process but a Web server plug-in. An HTTP adaptor is not needed when an application is deployed as a servlet.
- **An application instance.** The application instance receives incoming requests and responds to them, usually by returning a dynamically generated Web page. You can run multiple instances of an application when one instance is insufficient to handle the application's user load.

Developing a Web Application

Developing a Web application is a matter of creating your Web page 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 Web application with these tools is as follows:

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

You have already been introduced to EOModeler. The following sections introduce Project Builder and WebObjects Builder.

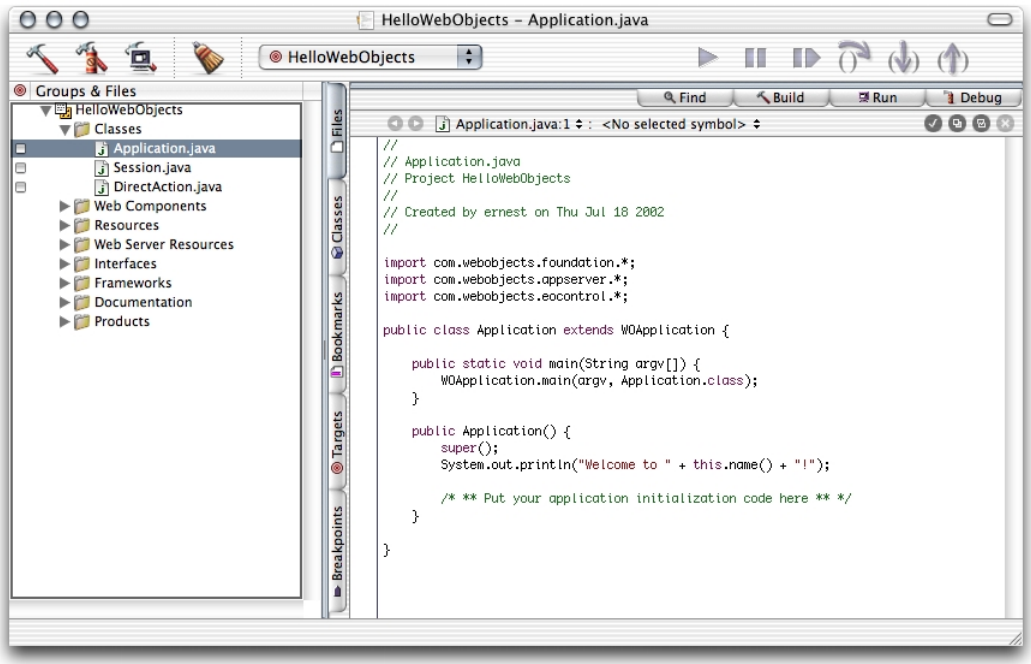
Project Builder

As its name implies, Project Builder manages the constituent parts of an application project, including source-code files, Web components, frameworks, 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. The Project Builder Assistant helps you create new Web components. You also can launch the other development tools from within Project Builder.

Figure 4-4 shows Project Builder in use.

Figure 4-4 Project Builder

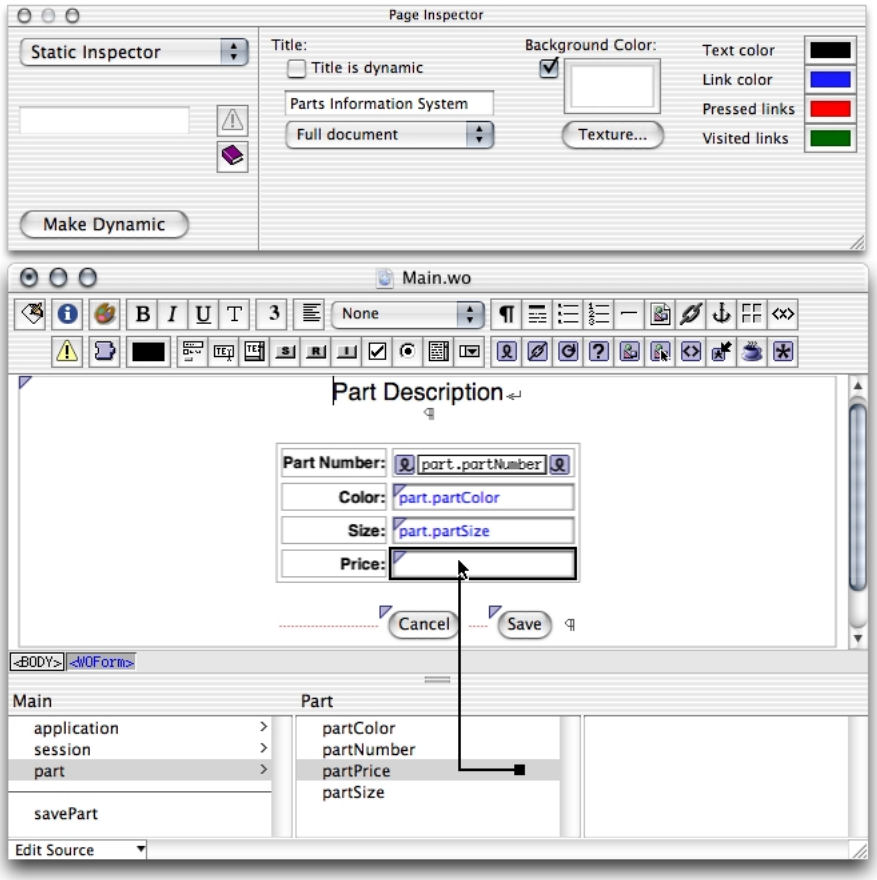


WebObjects Builder

You use WebObjects Builder to edit your application's Web components. WebObjects Builder allows you to graphically edit a component's Web page 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. As you define bindings, WebObjects Builder adds them to the component's WOD file.

Figure 4-5 WebObjects Builder



Guidelines for Choosing the Web Application Approach

The Web application approach has the following advantages:

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

In some cases, you can use the Direct to Web rapid-development system to create a Web application. Direct to Web works particularly well for data-driven applications, prototypes, and internal applications. See "The Direct to Web Approach" (page 51) for more information.

The Direct to Web Approach

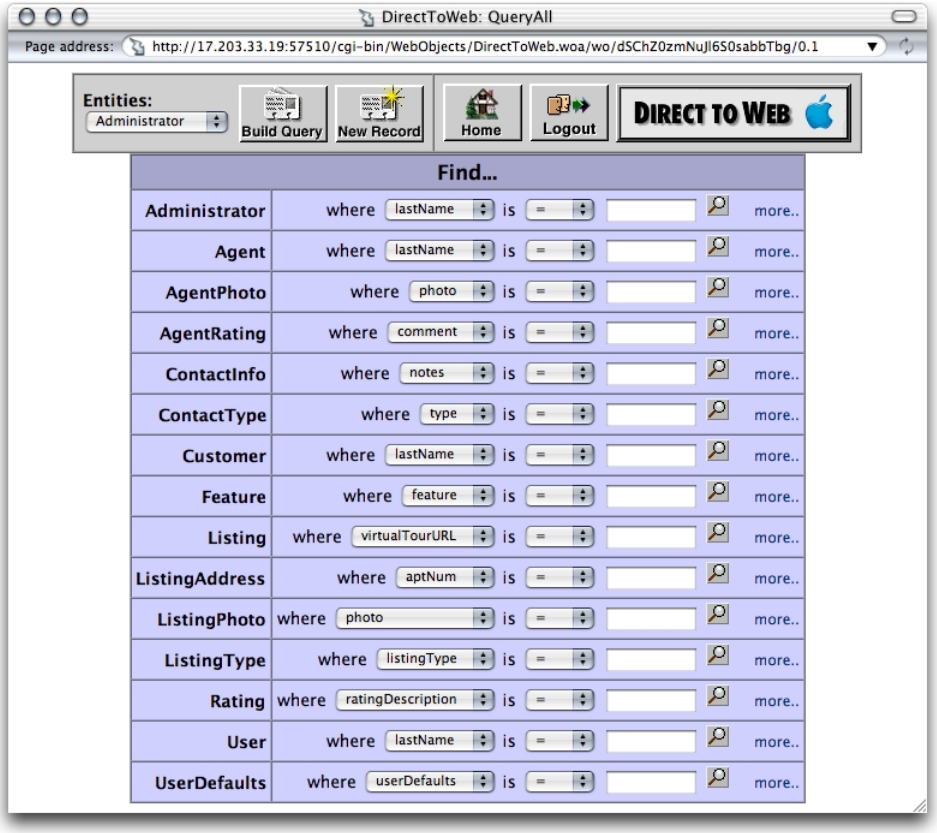
Direct to Web is a technology that creates Web applications whose main purpose is to provide a user interface that can be used to access and modify data in a data store. All you need to do is create a Direct to Web project in Project Builder and provide a data model and additional business logic, which can be contained in a custom framework. The Project Builder Assistant then generates a ready-to-test Direct to Web application.

By default, Direct to Web applications display a login page when a user accesses them for the first time during a session (Figure 4-6). By default, this page provides an interface to authenticate the user but does not actually perform any authentication. Because the login page is a regular Web component, you can customize its behavior.

Figure 4-6 A login page

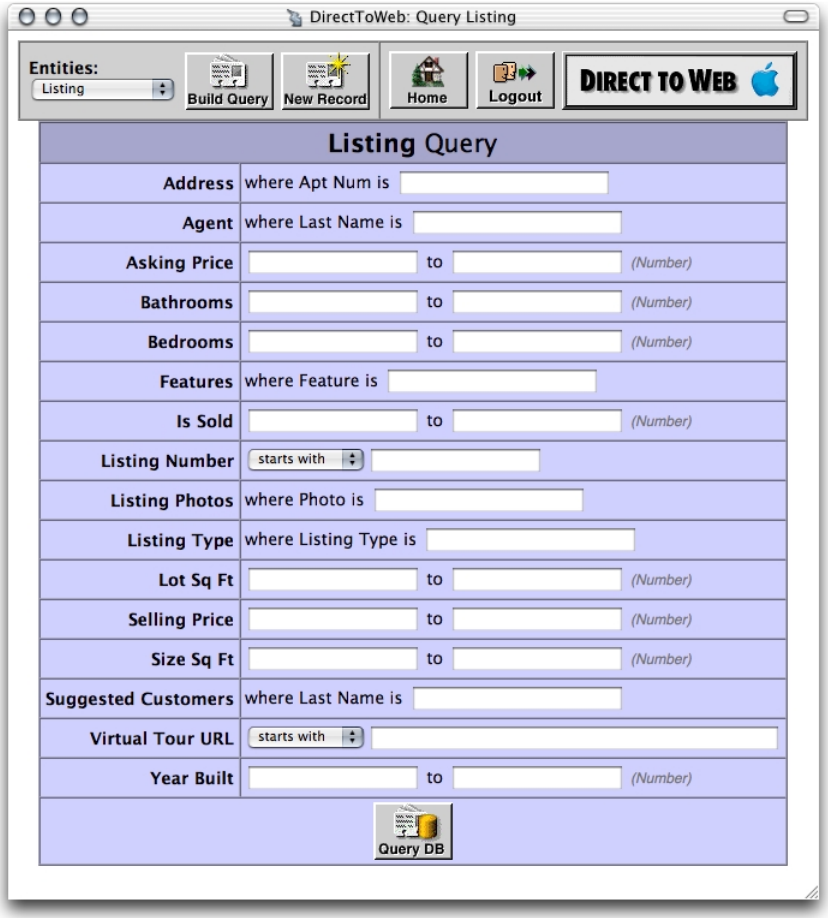
After the user logs in, Direct to Web displays its first dynamically generated page: a query-all page (Figure 4-7). This page allows the user to specify the type of records she wants to work with. She can define a query for any type of record that is visible in the page (the developer decides which entities are visible and which are not).

Figure 4-7 A query-all page



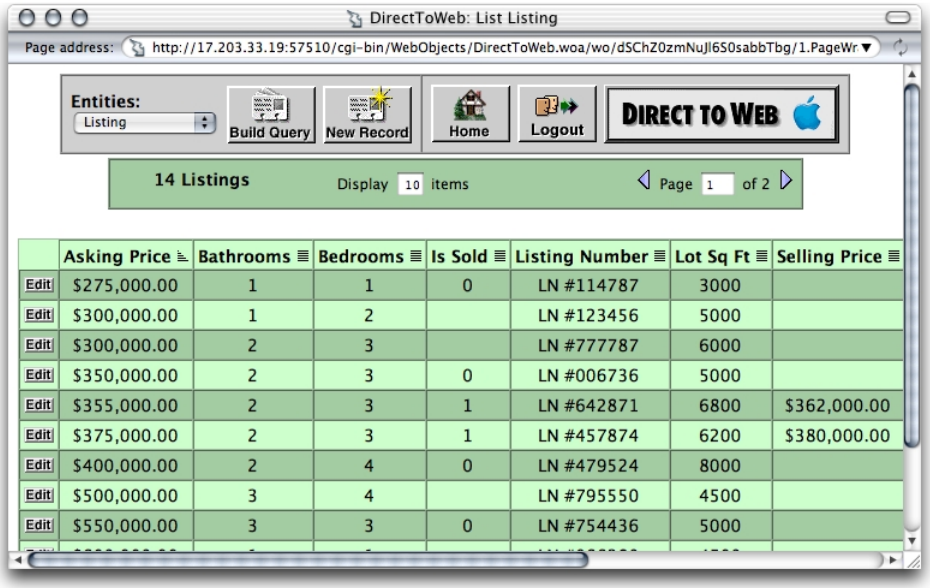
If the query-all page is not specific enough, the user can click one of the “more” links, which brings up a query page specific to the corresponding type of record. Figure 4-8 shows the Web page that appears when the user chooses to view detailed query criteria for Listing records. In this page, the 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 4-8 A query page



When the user clicks Query DB on the query page or the magnifying glass button on the query-all page (Figure 4-7), Direct to Web displays the records that match the query on a list page (Figure 4-9). This page presents the records in batches; the user can change the batch size and navigate from batch to batch.

Figure 4-9 A list page



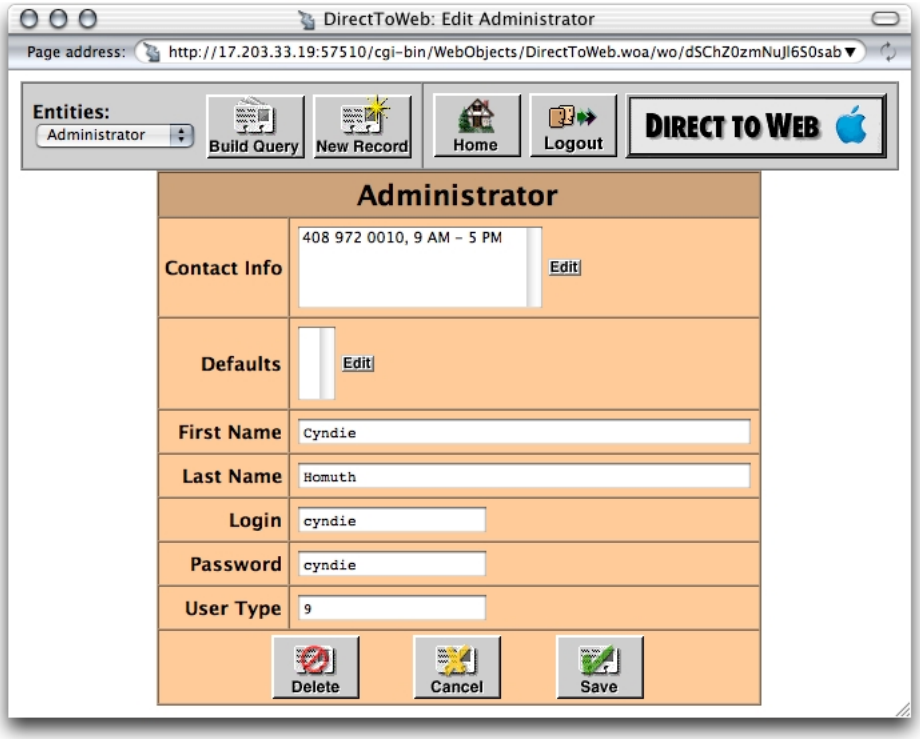
	Asking Price	Bathrooms	Bedrooms	Is Sold	Listing Number	Lot Sq Ft	Selling Price
Edit	$275,000.00	1	1	0	LN #114787	3000	
Edit	$300,000.00	1	2		LN #123456	5000	
Edit	$300,000.00	2	3		LN #777787	6000	
Edit	$350,000.00	2	3	0	LN #006736	5000	
Edit	$355,000.00	2	3	1	LN #642871	6800	$362,000.00
Edit	$375,000.00	2	3	1	LN #457874	6200	$380,000.00
Edit	$400,000.00	2	4	0	LN #479524	8000	
Edit	$500,000.00	3	4		LN #795550	4500	
Edit	$550,000.00	3	3	0	LN #754436	5000	

Note that each Listing record on the list page in Figure 4-9 has an Edit button, which indicates that the user can modify its contents. You can determine whether the attributes of an entity can be modified by the application's user.

If the records the user views are nonmodifiable, an Inspect button appears on each row instead of an Edit button. If the user clicks Inspect next to one of the records, Direct to Web displays an inspect page for the record that reveals additional information about it.

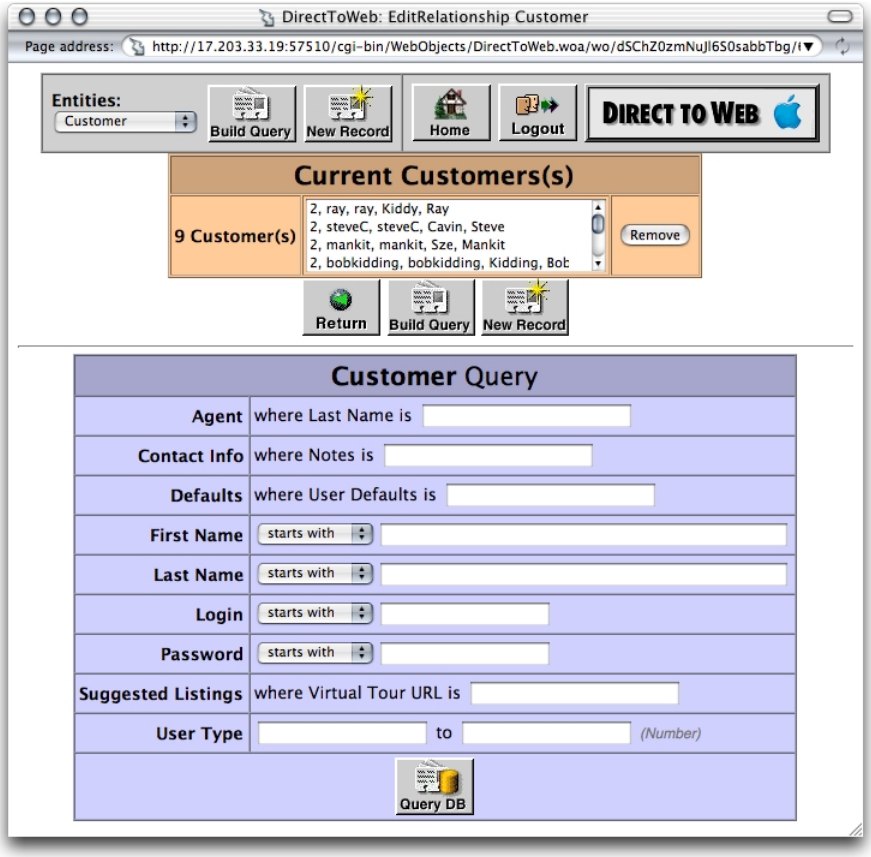
If the records displayed on the list page are modifiable and the user clicks Edit next to one of them, Direct to Web displays an edit page for the record (Figure 4-10). On the edit page, the user can edit the attributes for the object or click Edit next to one of the relationships to edit the relationship.

Figure 4-10 An edit page



The user edits a relationship using an edit-relationship page, shown in Figure 4-11, which allows her to edit to-many and to-one relationships.

Figure 4-11 An edit-relationship page



With the exception of the login page, every Direct to Web page has a toolbar containing a menu and buttons that assist in navigating around the application; it's shown in Figure 4-12.

Figure 4-12 The toolbar

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, shown in Figure 4-13, and the WebObjects look, shown in Figure 4-14.

Figure 4-13 A Neutral-look page

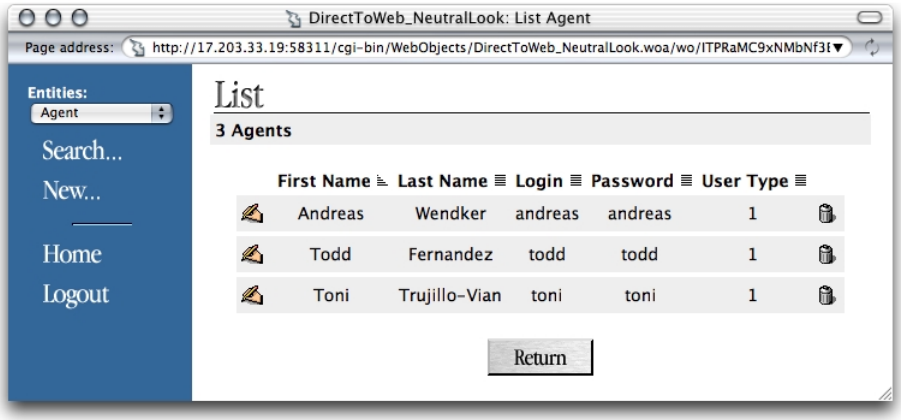
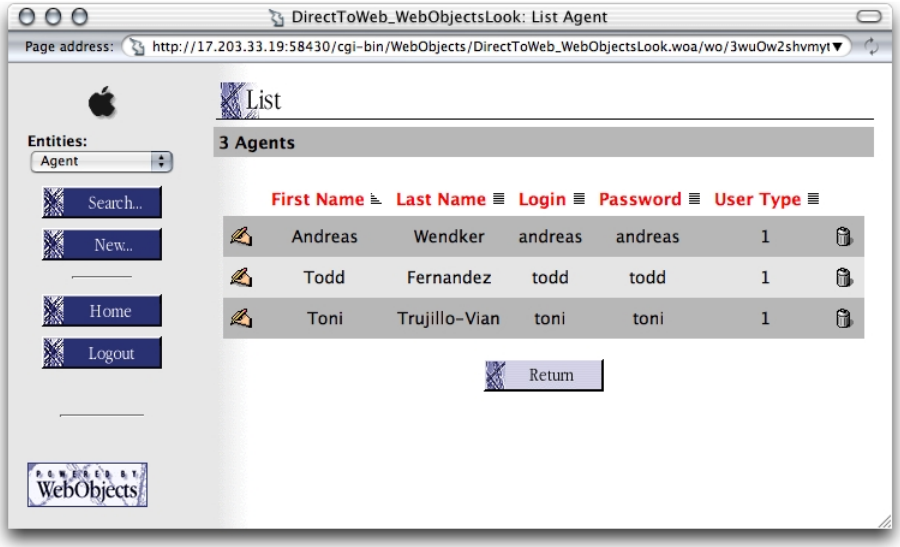


Figure 4-14 A WebObjects-look page

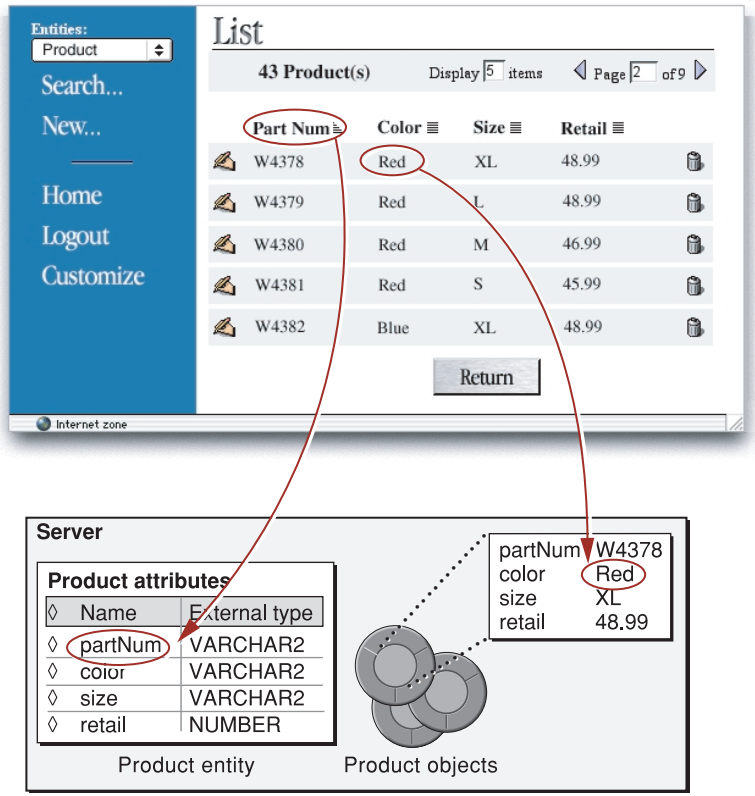


Direct to Web Architecture

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 record. These task pages are created using special components called Direct to Web templates.

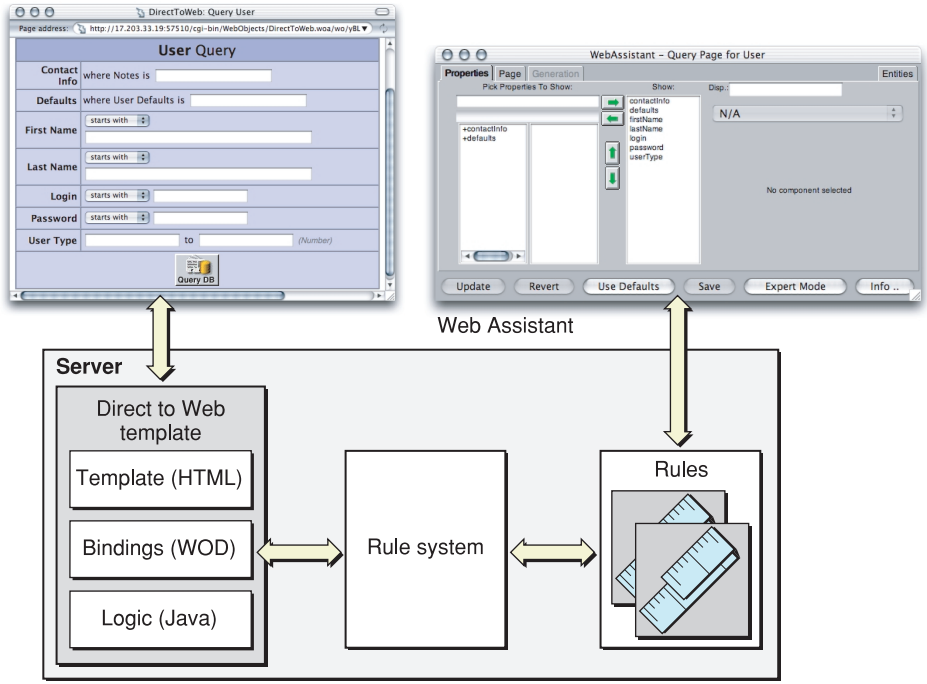
A Direct to Web template uses information from the entities it displays. A data entity, defined in the data model, specifies how a table's columns map to its attributes. The Direct to Web template takes advantage of the entity's property information (that is, information about the entity's attributes and relationships) and determines which properties to display. For example, a Direct to Web template displaying a list page for Listing objects can determine that it needs to display the address, asking price, features, and other attributes for each listing on the page. Figure 4-15 shows a similar scenario using an entity named Product.

Figure 4-15 Determining attributes from the entity



Direct to Web applications can be configured using a Java applet called 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 User entity, do not display the banner." Each rule has a priority; rules with a high priority override rules with lower priority. Direct to Web defines a set of essential rules that define the basic application behavior. You can define higher priority rules that override the default rules. This is exactly what the Web Assistant does. Figure 4-16 shows the relationship between the Direct to Web template, the rule system, the rule database, and the Web Assistant. The **rule system** analyzes data models and sets of rules to dynamically generate an application's user interface.

Figure 4-16 The Direct to Web rule system



Note that when you configure your application with the Web Assistant, you don't need to recompile your code to test the user interface. Direct to Web does not generate code. It generates Web pages at runtime based on Direct to Web templates and rules.

Developing a Direct to Web Application

You perform the following steps to create a Direct to Web application:

1. Create a model using EOModeler.
2. Create a Direct to Web application project using Project Builder.

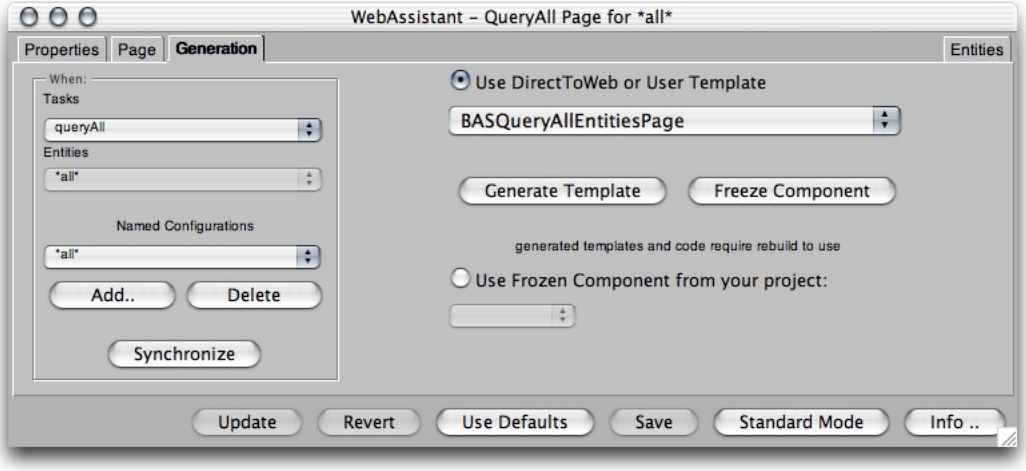
3. Customize your Direct to Web application using the Web Assistant (optional).
4. Further customize your Direct to Web application (optional).

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

The Web Assistant

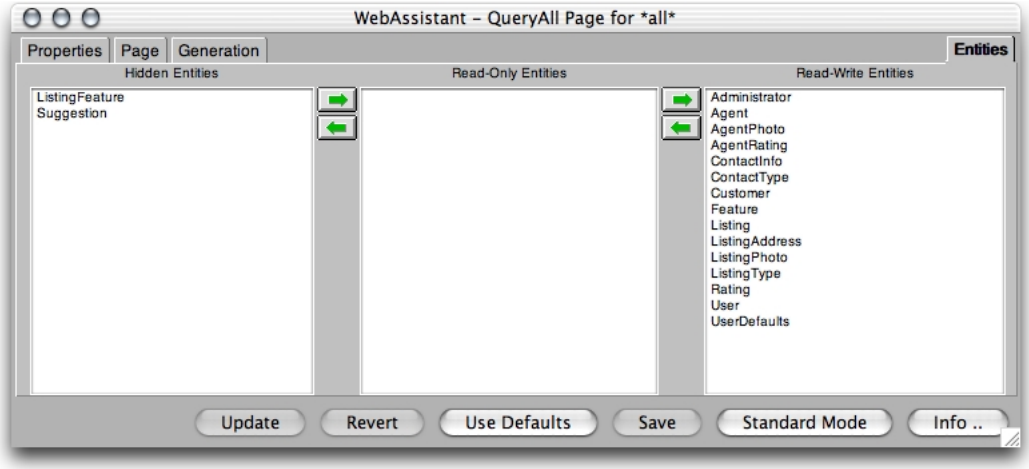
The Web Assistant is a Java applet that runs at the same time as your application. It communicates directly with Direct to Web and allows you to configure your application in many ways. Figure 4-17 shows the Web Assistant in use.

Figure 4-17 The Web Assistant



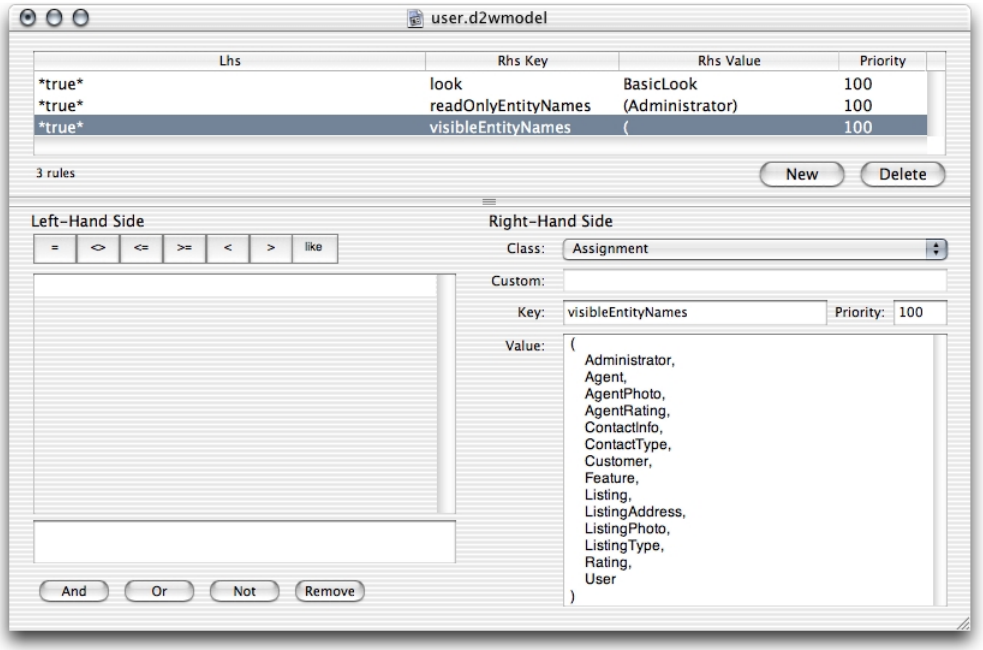
With the Web Assistant, you can designate which entities are modifiable, nonmodifiable, or hidden, as shown in Figure 4-18. You can also set appearance parameters for most of the pages that Direct to Web generates. For example, you can control whether the page displays with a banner. You can also change the background color of the table the page displays, if applicable. The Web Assistant also lets you to configure the way properties (attributes and relationships) appear on list, edit, and inspect pages.

Figure 4-18 The Entities pane of the Web Assistant



As mentioned earlier, as you configure your application with the Web Assistant, it defines rules that override the default Direct to Web rules. Thus, the Web Assistant is the preferred way to modify rules. Nevertheless, sometimes you need to change the default rules or override the default rules in ways the Web Assistant can't. In such cases, you use Rule Editor to edit the rules directly. Figure 4-19 shows Rule Editor in action.

Figure 4-19 Rule Editor



Advanced Customization of Direct to Web Applications

If you need to customize your application beyond what you can do with the Web Assistant and Rule Editor, 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 Web Assistant. When you do that, the page becomes a Web component in your project with a Web page template, a Java source file, and a bindings file. You can edit it with WebObjects Builder just as you would any other Web component. The downside is that you can't customize frozen pages with the 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 link at the bottom of every list page. To do so, you instruct the Web Assistant to generate a Direct to Web template, modify the

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

- **Modify the page wrapper and toolbar.** The page wrapper Web 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 toolbar appropriate for the look. Figure 4-12 (page 58) shows the toolbar for the Basic look. The toolbar is another Web component in your project.
- **Mix Web and Direct to Web pages**. You can navigate to a Direct to Web page from a Web page and vice versa. You can also embed certain Direct to Web functions within a Web 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 Web application with 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 Web applications; thus they have the same advantages that Web applications possess: portability and reduced system administration. What Direct to Web adds to the Web application approach is the ability to dynamically generate all the Web pages, relieving you of designing and coding them yourself. As a consequence, Direct to Web has the following advantages over the Web application 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 proven WebObjects technology, which increases the stability of applications and reduces the time required to test applications before deploying them.

Limitations of Direct to Web

The user interface generated by Direct to Web is based on HTML 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 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 the Direct to Web engine. You don't have to know what actions the engine is performing. This makes it easy for you to develop simple applications. But for certain customizations, the learning curve can get steep.

The dynamic generation of Web pages adds a layer of abstraction on top of the Web application development approach. This layer adds complexity that regular Web applications lack, 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 Web component because Direct to Web templates are more complex than most Web components.

Choosing a Web Application Development Approach

Direct to Web is particularly suited for data-driven 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. Direct to Web allows you to customize an application by adding rules.

For applications that are not data-driven or when you want to have complete control of the layout of the Web pages, the Web application approach can be the most appropriate. Furthermore, mastering the Web application approach can be beneficial when you need to heavily customize a Direct to Web application.

Once you are familiar with the operation of Direct to Web, you can explore using Direct to Web reusable components in an Web application. Direct to Web reusable components allow you to nest a Direct to Web templates. This technique can dramatically reduce the development time of certain types of pages like forms and list pages. For more information on Direct to Web reusable components, see *Inside WebObjects: Developing Applications With Direct to Web*.

CHAPTER 4

Web Applications

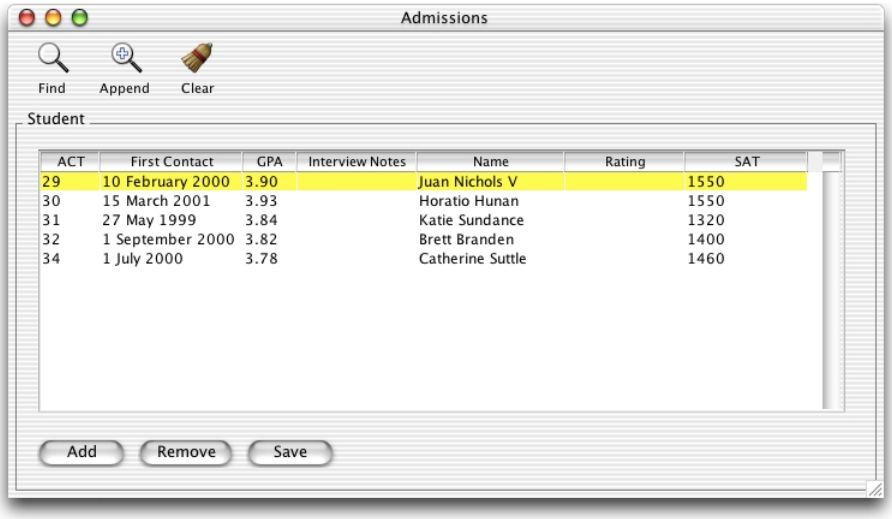
CHAPTER 5

Desktop Applications

If you need to develop distributed applications with more complex and responsive user interfaces than Web applications allow, Java Client might fill that requirement. Java Client applications use Swing to generate the user interface. These applications can be deployed as applets running in a Web browser or as desktop applications running in the client computer's Java virtual machine. The latter is highly recommended. However, since the Java Client architecture isolates the application from data-access mechanisms that are database-engine specific, you are not tied to any particular deployment scheme. For detailed information on Java Client applications, see *Inside WebObjects: Java Client Desktop Applications*. This chapter introduces the Java Client and Direct to Java Client development approaches.

In the example Java Client application shown in Figure 5-1, the user interface is like the interfaces you see in desktop applications.

Figure 5-1 A Java Client application



Java Client is a three-tier network application solution that allows you to develop platform-agnostic desktop applications with database access and full-featured user interfaces. Java Client applications are WebObjects applications: They share much of their API with traditional Web applications, such as Enterprise Objects for database access and the Foundation framework, which supplies data structures and other core functionality to applications.

Java Client Features

If you're looking for a Java application platform with robust data access, rapid development tools, and powerful, innovative customization capabilities, Java Client is the perfect solution. This section summarizes its features.

Better User Experience

Java Client applications differ from Web applications in that the user interface is built on Sun's Swing components, rather than on HTML code. This allows applications to take advantage of the rich user interface elements Swing offers. This is perhaps the primary reason you would choose to develop an application using Java Client: the need for a rich, more interactive user interface.

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

Object Distribution

Java Client is built on the paradigm of object distribution. It distributes enterprise-object instances between an application server and one or more clients—Java applications or applets. How this distribution occurs is up to the application's developer.

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

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

The Best of WebObjects

As with any type of WebObjects application, Java Client gives you a lot for free. Its tight integration with Enterprise Object technology, allow you to take full advantage of the rich data-access and persistence mechanisms the technology offers. Without writing a single line of code, Java Client allows you to connect user

interface widgets to database actions such as saving, retrieving, reverting, undoing, adding records, editing records, and more. Furthermore, Java Client's integration with Enterprise Objects abstracts development above the need to write SQL code. And the development tools you use to build Java Client applications let you build complex Swing-based user interfaces without writing any Java code.

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

Rapid Application Development

In addition to the powerful data-modeling, project-development, and interface-building tools, Java Client includes a sophisticated rapid development environment based on the WebObjects **rule system**, which analyzes data models and sets of rules in order to dynamically generate user interfaces. This application-development approach is called Direct to Java Client.

Direct to Java Client lets you focus on writing custom business logic and provides customization techniques that allow you to build sophisticated user interfaces without writing any code. In addition, with Direct to Java Client you can immediately see how changes in a data model are reflected in an application's user interface.

When to Use Java Client

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

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

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

However, you can also deploy Java Client applications as applets that run in Web browsers. Deploying as applets alleviates many of the issues encountered when running Java Client applications as desktop applications since the user doesn't need to download or install the client application. However, applets introduce other usability and deployment issues. For example, the Web browser must support embedded applets in Web pages and the user must configure the browser to allow applet execution.

WebObjects provides two effective solutions to the application-deployment problems mentioned earlier: the Java Client Class Loader and Java Web Start.

The Java Client Class Loader eases installation of client-side applications by requiring only the installation of a basic Java Client system. When the user launches the application, the classes specific to the application are downloaded. This way, the user always runs the latest version of the application without worrying about downloading any files. The user would have to install Java Client files only if you upgrade your WebObjects software. The drawback of this approach is that application launch is slower because downloading is required to have a working application.

Java Web Start, a more advanced technology, allows users to launch Java Client applications with a single click. The technology also provides streamlined application deployment by providing caching and other mechanisms to ease client-side class management. For more information on Java Web Start, see *Inside WebObjects: Java Client Desktop Applications*.

In deciding whether to use Java Client, you should evaluate the technology with the following criteria in mind:

- **Portability:** Java Client applications are 100% Pure Java applications. Java Client applications running in Mac OS X take advantage of platform-specific interface features such as the global menu and the dirty-window marker.
- **Performance:** After the initial download of Java classes to the client computer, Java Client applications don't exchange large chunks of data between the client side and the server side. Rather, compact business objects are exchanged over the network. Also, the Java Client architecture separates the user interface layer from the data-exchange layer; that way, enterprise-object data flows across the network, independent of user interface data. This feature enables Java Client applications to scale well.
- **Network environment:** Java Client applications can be deployed across the Internet; they are not inherently constrained to intranet environments. However, they are not appropriate for high-volume, high-visibility applications because of the long initial download and other system administration requirements.
- **System administration:** The presence of Java Runtime Environment (JRE) 1.3 or later is not ubiquitous among desktop operating systems. Although Mac OS X includes JRE 1.3 out of the box, JRE 1.3 is not available for Mac OS 9 or earlier. Sun provides the JRE for all Windows platforms, but it does not ship in the box. JRE 1.3 is available on many UNIX-based platforms. So, although the JRE is widely available, it must often be downloaded and installed by the end user. You should evaluate your target market, keeping in mind that some customers will not readily accept by the proposition of installing the JRE.
- **Security:** If you take careful steps to partition your business logic, Java Client applications offer security equal to that of Web applications. By default, Java Client uses HTTP as the transport protocol between the client side and the server side, but it can be replaced with another, more secure protocol such as Secure Sockets Layer (SSL). You can find more information on SSL at http://developer.netscape.com/docs.
- **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 snappier than Web applications.
- **User experience:** If your application demands a rich user interface, the manipulation of complex data, and long sessions, Java Client is an excellent choice.

Two Approaches to Java Client

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

Direct to Java Client technology and Java Client technology are very similar in that they use Enterprise Object technology to access data stores. They differ only at the user interface level.

Think of the relationship this way: A Java Client application is a completely customized Direct to Java Client application. While the user interface in Direct to Java Client applications is generated dynamically at runtime, the user interface in Java Client applications is fixed and built manually.

If you need the precise user interface customization that the nondirect approach allows, it's much easier to integrate a custom interface file into a Direct to Java Client application than to develop a completely custom Java Client application, although the latter is possible and supported. By customizing a Direct to Java Client application, you get the best of both worlds: the advantages of Direct to Java Client and the advantages of a customized user interface.

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

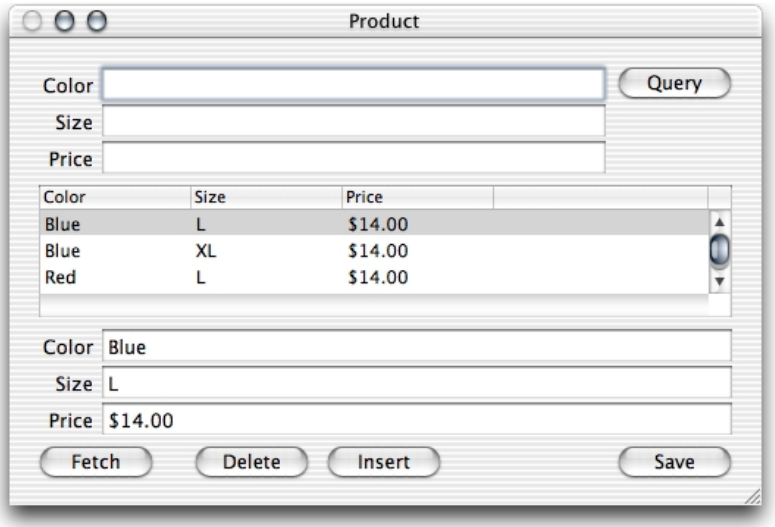
- Java Client Assistant
- Rule Editor
- freezing XML
- freezing nib files
- using custom controller classes
- using factory delegates

Nib files define the layout of the user interface elements of a desktop application. They are created using Interface Builder.

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

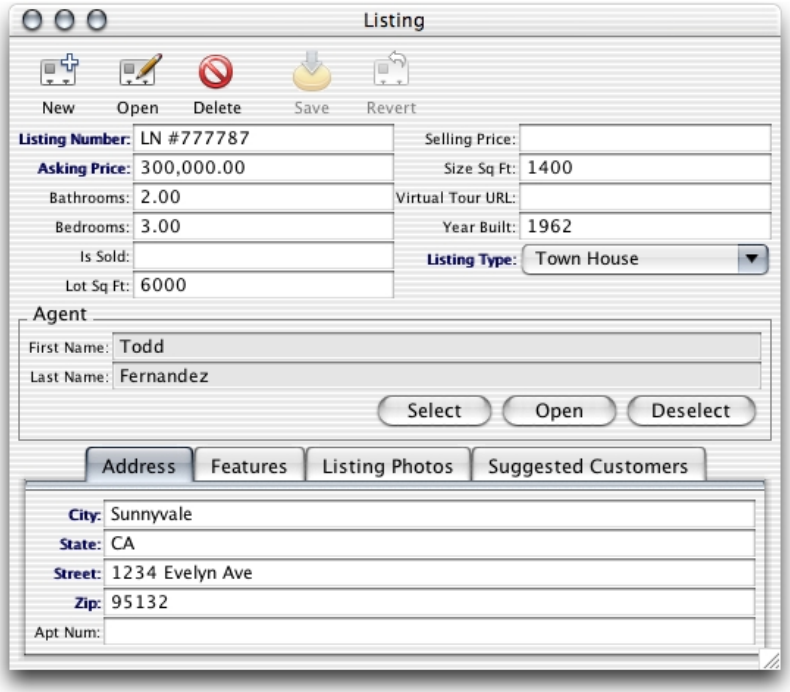
Typically, user interfaces built in Interface Builder for Java Client applications or for use as frozen interface files in Direct to Java Client applications resemble the user interface of a typical desktop application, such as the one shown in Figure 5-2.

Figure 5-2 A typical Java Client application



The dynamic user interface generation provided in Direct to Java Client applications yields interfaces that resemble . However, advanced Direct to Java Client applications are likely to include other, nondynamically generated user interfaces such as custom controller classes or frozen interface files (either built in Interface Builder or using Swing directly).

Figure 5-3 A typical Direct to Java Client application



Java Client Architecture

The Java Client architecture differs from the Web application architecture in that it's distributed across client and server systems as shown in Figure 5-4. The server side interacts with a database server as in Web applications; the client side, in addition to providing the application's user interface, can also contain nonsensitive business logic.

Figure 5-4 Java Client's distributed, multitier architecture

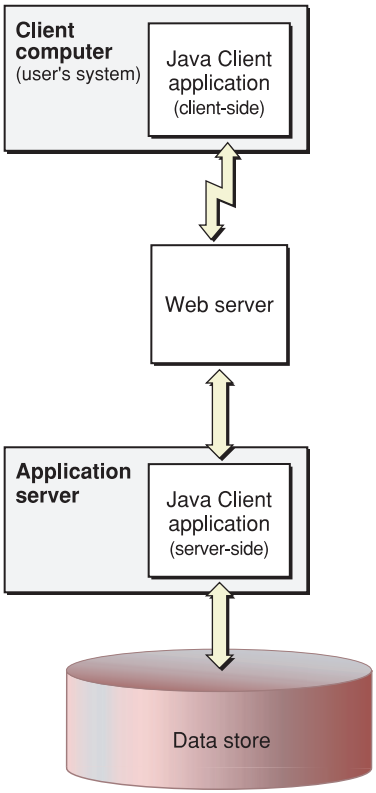
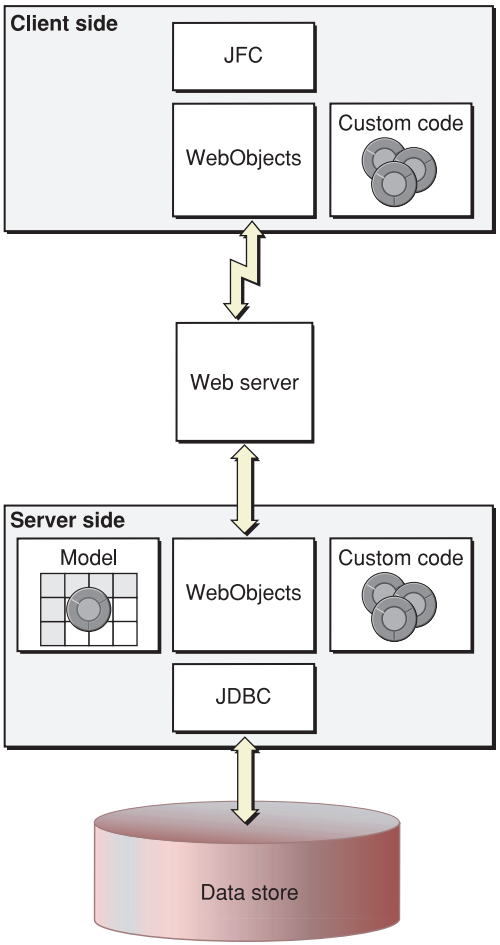


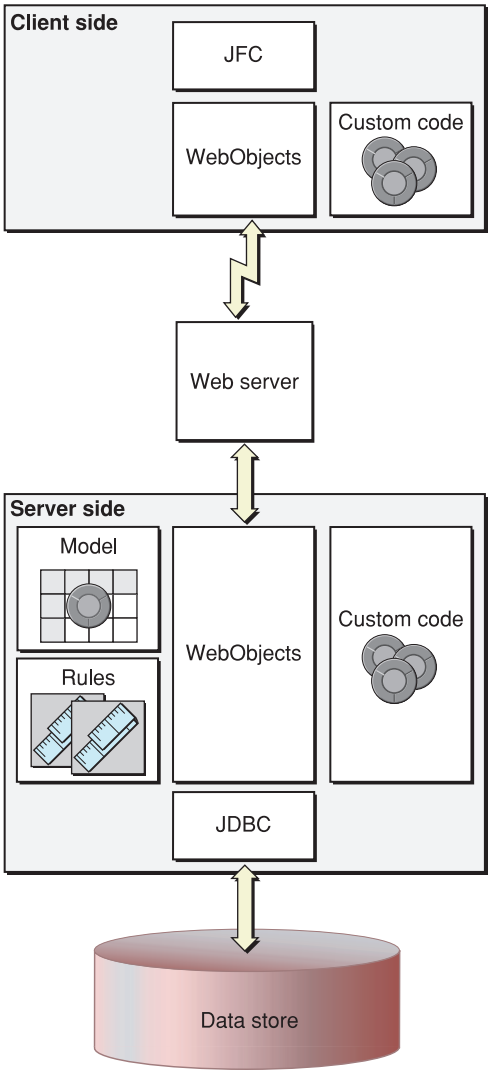
Figure 5-5 elaborates the architecture of Java Client applications.

Figure 5-5 Architecture of a Java Client application



The architecture of Direct to Java Client applications is slightly more complex than that of nondirect Java Client applications, as illustrated in Figure 5-6. It includes the rule system—the part of the server-side application responsible for dynamically generating the user interface and defining its behavior.

Figure 5-6 Architecture of a Direct to Java Client application



The client-side application and the server-side application have duties other than merely providing the user interface and data-store access—for example, each can contain business logic and each communicates with the other through a Web server.

However, the separation of data-store access and the user interface is significant because it provides a rich user interface without compromising security or performance.

Sensitive business logic and database connection logic is provided only by the server-side application. Because compiled Java on the client side can be decompiled, the client-side 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 (data validation, for example), round trips to the server are limited.

The Java Client architecture duplicates the graph of enterprise-object instances on the client-side application so the object graph and its management occur on both the client and the server. WebObjects handles communication between client and server and synchronizes the enterprise-object graphs of the two tiers.

Desktop User Interface

The user interface itself is implemented using Swing. 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.

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

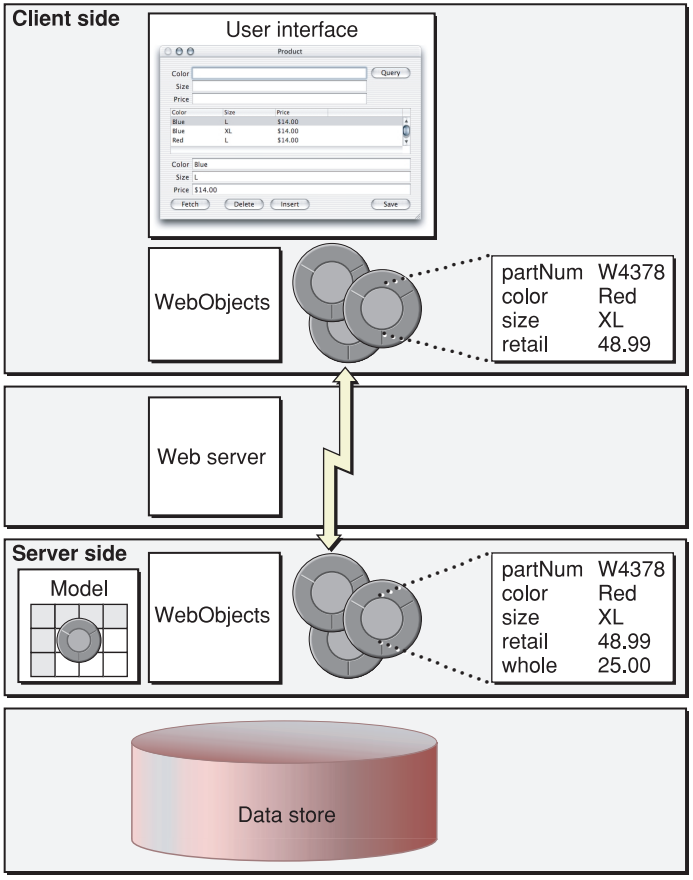
Data Synchronization Between Client and Server

Figure 5-7 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 data-store server.

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

Figure 5-7 Data flow in a Java Client application



Although requested enterprise-object instances are copied from the server to the client, and these objects exist in parallel object graphs on both tiers, the enterprise-object instances on the client do not exactly mirror the enterprise-object instances 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 5-7, the client-side enterprise objects don't have the `whole` property, the price the seller paid to the manufacturer.

Once the client has fetched data, this data is cached and represented internally by the client's object graph. As users modify the data (or add or delete records), the client application updates the client's object graph to reflect the new state. When the client application initiates a save operation, WebObjects synchronizes the server-side objects with the values of the client-side objects. If the business logic on the server validates the changes made, they 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 objects on the client before client-side objects invoke methods on server-side objects.

Java Client and Other Multitier Systems

There are many distributed multitier Java-based architectures on the market today. So how do they compare with Java Client?

- **Client JDBC applications** use a fat-client architecture. Custom code invokes JDBC operations 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 architecture are typical of all fat-client architectures. Security is a problem because the bytecodes on the client are easily decompiled, leaving both sensitive data and business rules at risk. In addition, this

architecture doesn't scale; it is expensive to move data over the channel to the client. Also, client-JDBC applications access the data store directly—there is no server layer to validate data or control access to the data store.

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

However,

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

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

Developing a Java Client Application

The essential tasks you need to perform when creating a Java Client application are listed below:

1. Create a model using EOModeler.
2. Create a project using Project Builder.
3. Write source code for enterprise-object classes (if your enterprise objects require custom business logic).
4. Create your application's user interface with Interface Builder (Java Client approach).
5. Customize your application's user interface (Direct to Java Client approach).
6. Write source code for any application-level logic.

The tasks have much in common with those for developing Web 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 side and one for the client side. The client and server classes can have different sets of properties and business logic. This means that programming a Java Client application requires a specific design technique that isn't necessary in Web application 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 properties and business logic to put in each class.

Usually, client-side enterprise-object classes have the more restricted set of properties 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—Java Client Approach

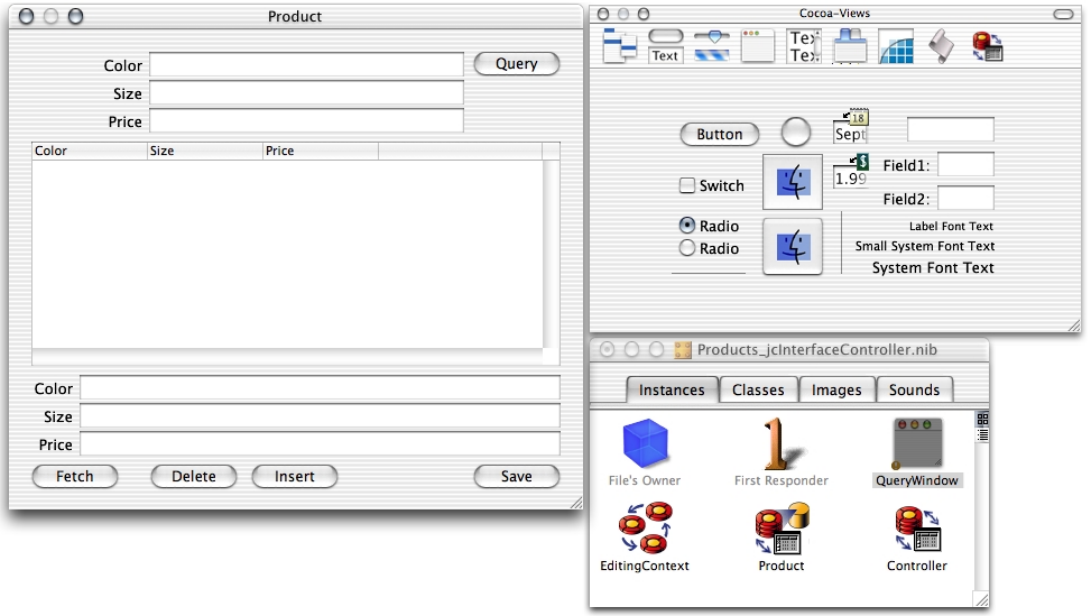
A Java Client application gives you considerable flexibility in how you compose its user interface. Ideally, you provide an application's entire user interface in a single application that runs on the client. But you can also combine Java Client applets, static Web pages, and Web components in various 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 windows, based on account information. In addition, Java Client applets are not limited to the downloaded Swing components; as with any applet, they can create dialogs and secondary windows quickly.

If your application's user interface uses static and dynamically generated Web pages, you create those components of the user interface using WebObjects Builder, as described in "Web Applications" (page 41). The process is different for creating a Java Client application or applet. Instead of using WebObjects Builder to create the user interface, you use Interface Builder.

Note: The process for creating a Java Client user interface is very similar to 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 in a window, as shown in Figure 5-8. It does more, however, than simple user interface layout. Interface Builder also lets you create, edit, and connect objects so they can communicate with one another at runtime.

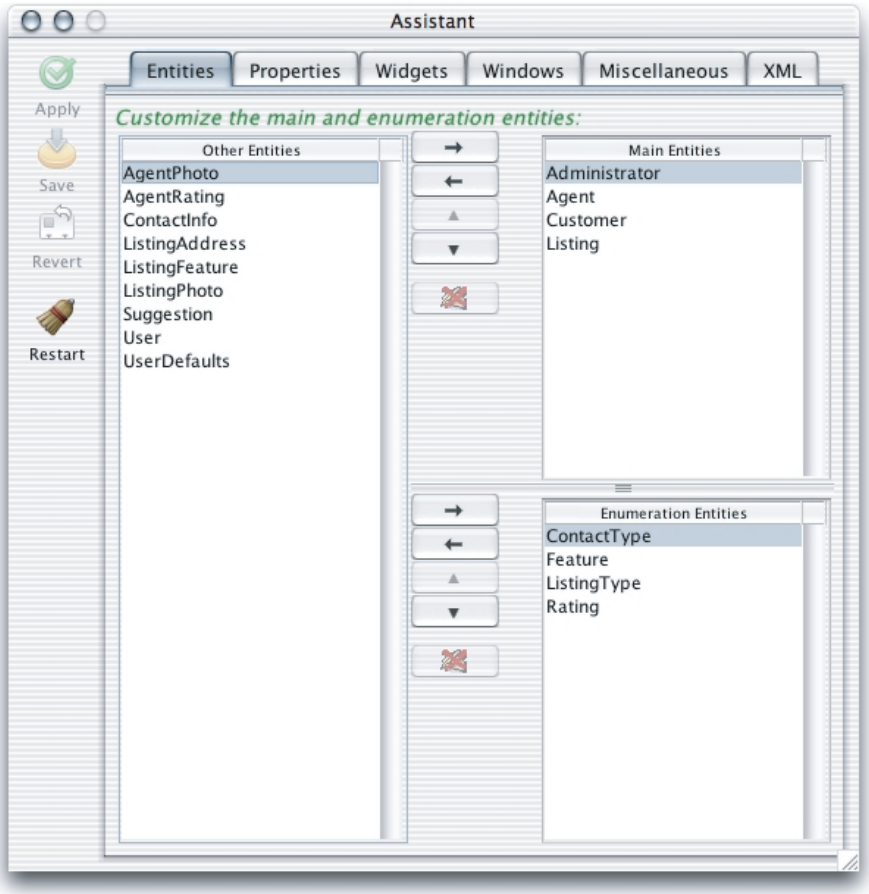
Figure 5-8 Composing a user interface with Interface Builder



Customizing the User Interface—Direct to Java Client Approach

Writing custom rules is another way to customize your Direct to Java Client application. It's similar to writing custom rules for Direct to Web applications. The information about how to generate the user interface of a Direct to Java Client application is stored in model files and rule sets. The default rules generate the default Direct to Java Client application user interface. To customize the user interface, you use the Direct to Java Client Assistant, shown in Figure 5-9, and write rules with Rule Editor. For more information, see "Developing a Direct to Web Application" (page 61).

Figure 5-9 Direct to Java Client Assistant tool



There are also 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 the menus of an application. Additionally, you can extend 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 technology is flexible and extensible; there are numerous customization approaches. There are simple approaches that are code-free and maintainable (using the Java Client Assistant and writing custom rules); there are also more specialized but complex approaches that require a lot of work, and there's everything in between. It's generally a question of what tradeoffs you're willing to make.

Choosing a Desktop Application Development Approach

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

Eventually, you may need to customize a Direct to Java Client application, which requires learning some of the customization methods listed earlier in this chapter. Although this most likely requires you to learn about the rule system, the controller hierarchy, and user interface generation in XML, the customization techniques work at a higher level than Swing. You can also save time by learning the Direct to Java Client customization techniques rather than creating your own Java Client interfaces in Interface Builder or using Swing.

Table 5-1 compares using the two Java Client approaches in various development tasks.

Table 5-1 Comparison of Java Client and Direct to Java Client

Task	Java Client	Direct to Java Client
Customization	Interface Builder and Swing.	Moderate. Available tools: Direct to Java Client Assistant, XML freezing, nib files, custom controller classes, controller factory delegates.
Development	Moderate to difficult, depending on user interface design.	Rapid. User interfaces are automatically generated and easily customizable.
User interface synchronization with data model	Difficult. User interface not synchronized with data model once user interface building begins.	Easy. Synchronization happens throughout much of the customization process.
Localization	Moderate. Must use separate nib files.	Easy. Automated using localizable string tables.

It is possible to manually develop effective Java Client applications without using Direct to Java Client; however, this approach begets little benefit. Using custom nib files in Direct to Java Client applications is the most sensible way to integrate custom user interfaces into Java Client applications.

CHAPTER 6

Web Services Applications

WebObjects allows you to provide and consume Web services, which simplify the development of distributed applications. Web services provide an efficient way for applications to communicate with each other. Based on Simple Object Access Protocol (SOAP), Web services can be used across multiple platforms, including Microsoft's .NET environment. A Web service is composed of operations, which are similar to the methods of a Java class (in WebObjects, that's exactly what they are). For example, a company could develop a Web service that provides the current stock price for a specific stock symbol.

This chapter introduces Web service development in WebObjects. For an in-depth discussion, see *Inside WebObjects: Web Services*.

Providing Web Services

With WebObjects you can easily take advantage of Web services to streamline your organization's internal information-processing systems or to allow your business partners to access and modify company information in an efficient manner.

WebObjects goes a long way towards streamlining Web service development and consumption. The sections that follow give you an idea of how straightforward Web service development is with WebObjects.

A Sophisticated Calculator

Web services allow the seamless integration of existing business processes across a company's departments or between organizations. WebObjects allows you to take logic hitherto available through slow, error-prone user interfaces and publish it as a Web service.

A calculator is something you might want to make available to other applications. Imagine for a moment that you have developed a great Calculator class with methods to add, subtract, multiply, and divide numbers. Listing 6-1 shows this innovative class.

Listing 6-1 `Calculator.java` class

```
public class Calculator extends Object {

    public static double add(double addend1, double addend2) {
        double sum = addend1 + addend2;
        return sum;
    }

    public static double subtract(double minuend, double subtrahend) {
        double difference = minuend - subtrahend;
        return difference;
    }

    public static double multiply(double multiplicand1, double multiplicand2) {
        double product = multiplicand1 * multiplicand2;
        return product;
    }

    public static double divide(double dividend, double divisor) {
        double quotient = dividend / divisor;
        return quotient;
    }
}
```

The class contains no Web service–related method invocations. It doesn't even use objects; all the method parameters and return values are primitive types. The class is the main part of a WebObjects application project called Calculator.

Publishing the Calculator Class as a Web Service

After developing multiple user interfaces to your class's methods to satisfy the demands of your customers, you decide to make it available as a Web service. This way you can concentrate on making the class's logic more efficient instead of on the design of Web pages or the layout of user interface controls in desktop applications.

So, what procedure do you need to follow to be able to deploy your application as a Web service?

First, you need to add the Web service–support framework to your project.

Second, you need to add the following method invocation to your `Application.java` file.

```
WOWebServiceRegistrar.registerWebServiceForClass(Calculator.class, true);
```

Third, build and run the application. You are now a Web service provider.

Web Services Description Language

Web Services Description Language (WSDL) is the lingua franca of Web service-enabled applications. When you provide a Web service, for the most part, you provide a WSDL document that describes it. Your Web service consumers use this document to find out what operations the service supports, the number and data types of the arguments each operation requires, and the number and data types of the values the operations return.

You don't have to worry about generating the WSDL document for your Web service because WebObjects does that for you. Your customers obtain it by accessing a URL similar to this one:

```
http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator?wsdl
```

Listing 6-2 shows the WSDL document generated by the Calculator application.

Listing 6-2 WSDL document for the Calculator Web service

```
<wsdl:definitions targetNamespace="http://noss.apple.com/cgi-bin/WebObjects/
Calculator.woa/ws/Calculator" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://noss.apple.com/
cgi-bin/WebObjects/Calculator.woa/ws/Calculator-impl" xmlns:intf="http://
noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator" xmlns:soapenc="http://
schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/
2001/XMLSchema">
  <wsdl:message name="subtractRequest">
    <wsdl:part name="minuend" type="xsd:double"/>
    <wsdl:part name="subtrahend" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="addResponse">
    <wsdl:part name="return" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="addRequest">
    <wsdl:part name="addend1" type="xsd:double"/>
    <wsdl:part name="addend2" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="divideResponse">
    <wsdl:part name="return" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="subtractResponse">
    <wsdl:part name="return" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="divideRequest">
    <wsdl:part name="dividend" type="xsd:double"/>
    <wsdl:part name="divisor" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="multiplyRequest">
    <wsdl:part name="multiplicand1" type="xsd:double"/>
    <wsdl:part name="multiplicand2" type="xsd:double"/>
  </wsdl:message>
  <wsdl:message name="multiplyResponse">
    <wsdl:part name="return" type="xsd:double"/>
  </wsdl:message>
  <wsdl:portType name="Calculator">
    <wsdl:operation name="multiply" parameterOrder="multiplicand1 multiplicand2">
      <wsdl:input message="intf:multiplyRequest" name="multiplyRequest"/>
```

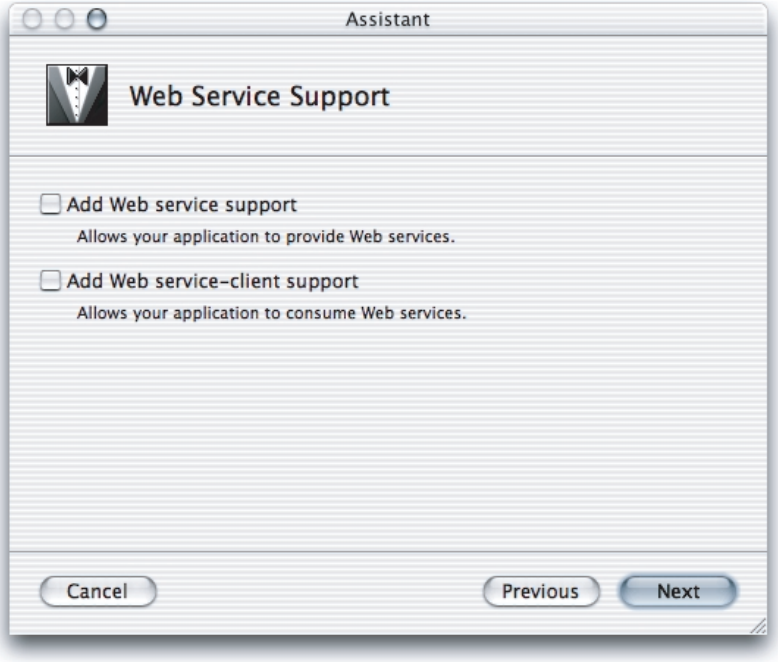
```
      <wsdl:output message="intf:multiplyResponse" name="multiplyResponse"/>
    </wsdl:operation>
    <wsdl:operation name="divide" parameterOrder="dividend divisor">
      <wsdl:input message="intf:divideRequest" name="divideRequest"/>
      <wsdl:output message="intf:divideResponse" name="divideResponse"/>
    </wsdl:operation>
    <wsdl:operation name="subtract" parameterOrder="minuend subtrahend">
      <wsdl:input message="intf:subtractRequest" name="subtractRequest"/>
      <wsdl:output message="intf:subtractResponse" name="subtractResponse"/>
    </wsdl:operation>
    <wsdl:operation name="add" parameterOrder="addend1 addend2">
      <wsdl:input message="intf:addRequest" name="addRequest"/>
      <wsdl:output message="intf:addResponse" name="addResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CalculatorSoapBinding" type="intf:Calculator">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="multiply">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="multiplyRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
      </wsdl:input>
      <wsdl:output name="multiplyResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="divide">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="divideRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
      </wsdl:input>
      <wsdl:output name="divideResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
```

```
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="subtract">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="subtractRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
      </wsdl:input>
      <wsdl:output name="subtractResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="add">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="addRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
      </wsdl:input>
      <wsdl:output name="addResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://noss.apple.com/cgi-bin/WebObjects/Calculator.woa/ws/Calculator"
use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CalculatorService">
    <wsdl:port binding="intf:CalculatorSoapBinding" name="Calculator">
      <wsdlsoap:address location="http://noss.apple.com/cgi-bin/WebObjects/
Calculator.woa/ws/Calculator"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

While somewhat verbose, WSDL documents provide Web service consumers a detailed description of the service. You don't have to worry about interpreting or modifying WSDL documents, however. WebObjects provides programming interfaces to most of the properties an application needs to invoke Web service operations.

Consuming Web Services

To create a Web service client project using Project Builder, you create a WebObjects application project and tell the Assistant to add the Web services frameworks.



Listing 6-3 shows a helper class used to consume Calculator operations.

Listing 6-3 CalculatorClient.java class

```
import java.net.*;
import java.util.Enumeration;

import com.webobjects.foundation.*;
import com.webobjects.webservices.client.*;

public class CalculatorClient extends Object {

    /**
     * Object through which the Web service's operations are invoked.
     */
    private WOWebServiceClient _service_client = null;

    /**
     * Address for the Web service's WSDL document.
     */
    private String _service_address = "http://noss.apple.com/cgi-bin/WebObjects/
Calculator.woa/ws/Calculator?wsdl";

    /**
     */
    public CalculatorClient() {
        super();
    }

    /**
     * Obtains the Web service's operation names.
     * @return the Web service's operation names.
     */
    public NSArray operations() {                                                    //1
        NSArray operations = serviceClient().operationsForService(serviceName());
        NSMutableArray operation_names = new NSMutableArray();
        Enumeration operations_enumerator = operations.objectEnumerator();
        while (operations_enumerator.hasMoreElements()) {
            WOWebServiceOperation operation =
(WOWebServiceOperation)operations_enumerator.nextElement();
            operation_names.addObject((String)operation.name());
        }
        return operation_names;
```

```
}

/**
 * Invokes the Web service's operations.
 * @param operation      operation to invoke;
 * @param arguments      argument list;
 * @return               value returned by the operation.
 */
public Double invoke(String operation, Object[] arguments) {                    //2
    Object result = serviceClient().invoke(serviceName(), operation, arguments);
    return (Double)result;
}

/**
 * Obtains the Web service name.
 * Normally one WSDL file describes one Web service,
 * but it could describe one or more services.
 * @return Web service name.
 */
public String serviceName() {
    return (String)serviceClient().serviceNames().objectAtIndex(0);
}

/**
 * Obtains an agent through which service operations are invoked.
 * @return service agent.
 */
private WOWebServiceClient serviceClient() {                                    //3
    if (_service_client == null) {
        _service_client = clientFromAddress(_service_address);
    }
    return _service_client;
}

/**
 * Obtains a Web service client object through which
 * service operations can be invoked.
 * @return Web service client object.
 */
private static WOWebServiceClient clientFromAddress(String address) {
    WOWebServiceClient service_client = null;
```

```
        // Create the Web service's URL.
        URL url;
        try {
            url = new URL(address);
        }

        catch (MalformedURLException e) {
            url = null;
        }

        // Get a service-client object.
        service_client = new WOWebServiceClient(url);

        return service_client;
    }

}
```

Some of the methods in Listing 6-3 have numbers on the far right of the line of code. The following list explains these methods.

1. The `operations` method uses WebObjects API that extracts information contained in the WSDL document provided by the Web service. In this case it obtains the names of the operations the Web service provides and returns them as an NSArray.
2. The `invoke` method takes an operation name and an array of arguments, invokes the operation, and returns the result.
3. The `serviceClient` method obtains and returns a WOWebServiceClient object; this is the object through which Web service operations are invoked. The object can be used to make many operation invocations. Notice that only one instance of WOWebServiceClient is used throughout the life of a CalculatorClient object.

Figure 6-1 presents a possible user interface for the Calculator Web service.

Figure 6-1 Web page that uses the Calculator Web service

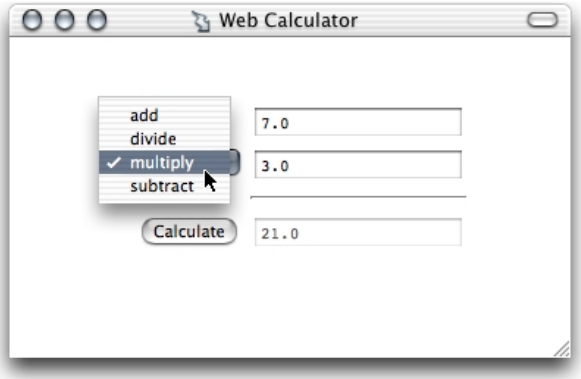
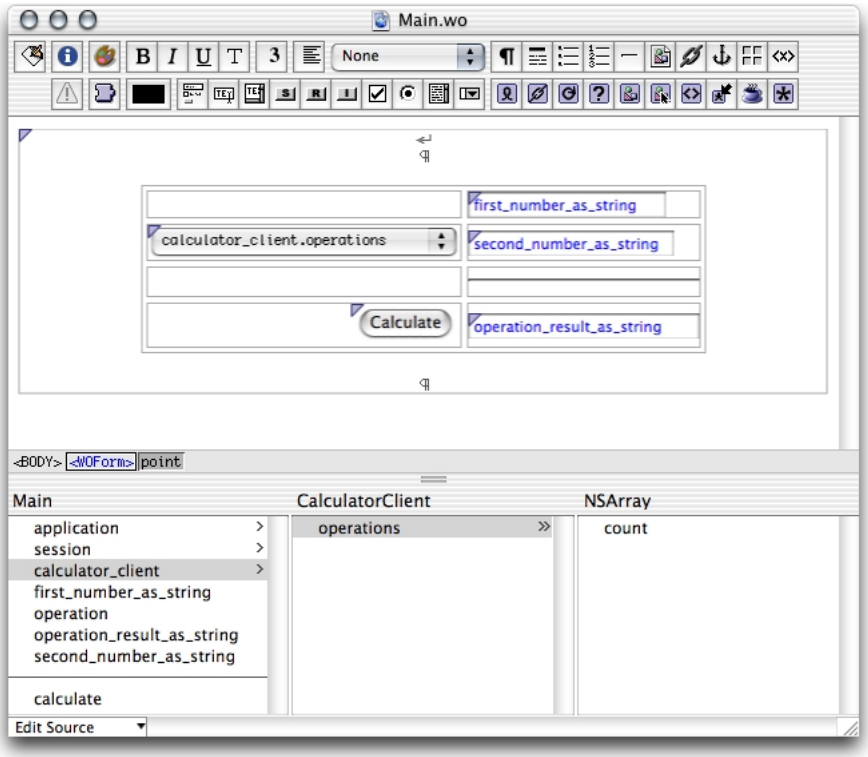


Figure 6-2 shows the Web component that determines the layout of the Web Calculator user interface elements. Listing 6-4 displays the business logic behind the component. As you can see, WebObjects provides facilities to easily consume Web services.

Figure 6-2 Web component that lays out user interface elements for the Web Calculator application



Listing 6-4 Business logic behind Web Calculator's user interface

```
import java.lang.*;

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

public class Main extends WOComponent {

    // User input: operation to invoke. Linked to pop-up menu.
```

```
protected String operation;

// User input: operands. Linked to text fields.
protected String first_number_as_string;
protected String second_number_as_string;

// Operation output: result of operation. Linked to disabled text field.
protected String operation_result_as_string;

// Object used to invoke the Calculator Web service's operations.
protected CalculatorClient calculator_client;

public Main(WOContext context) {
    super(context);

    // Create the Calculator service client.
    calculator_client = new CalculatorClient();
}

// Gets values entered in text fields, invokes the operation,
// and puts the result in the output text field.
public WOComponent calculate() {
    try {
        // Create argument values.
        double argument1 = Double.parseDouble(first_number_as_string);
        double argument2 = Double.parseDouble(second_number_as_string);

        // Create argument list.
        Object[] arguments = { new Double(argument1), new Double(argument2) };

        // Invoke the operation.
        Double result = calculator_client.invoke(operation, arguments);
        double result_value = result.doubleValue();

        first_number_as_string = Double.toString(argument1);
        second_number_as_string = Double.toString(argument2);
        operation_result_as_string = Double.toString(result_value);
    }

    catch (NumberFormatException nfe) {
        nfe.printStackTrace();
```

```
        }

        return null;
    }
}
```

Direct to Web Services

Similar to Direct to Java Client and Direct to Web, the Direct to Web Services approach to Web service development uses a data model and rule sets to generate Web services. However, when you first run a Direct to Web Services application, there's only a default Web service configured. This service is not enabled and contains no operations.

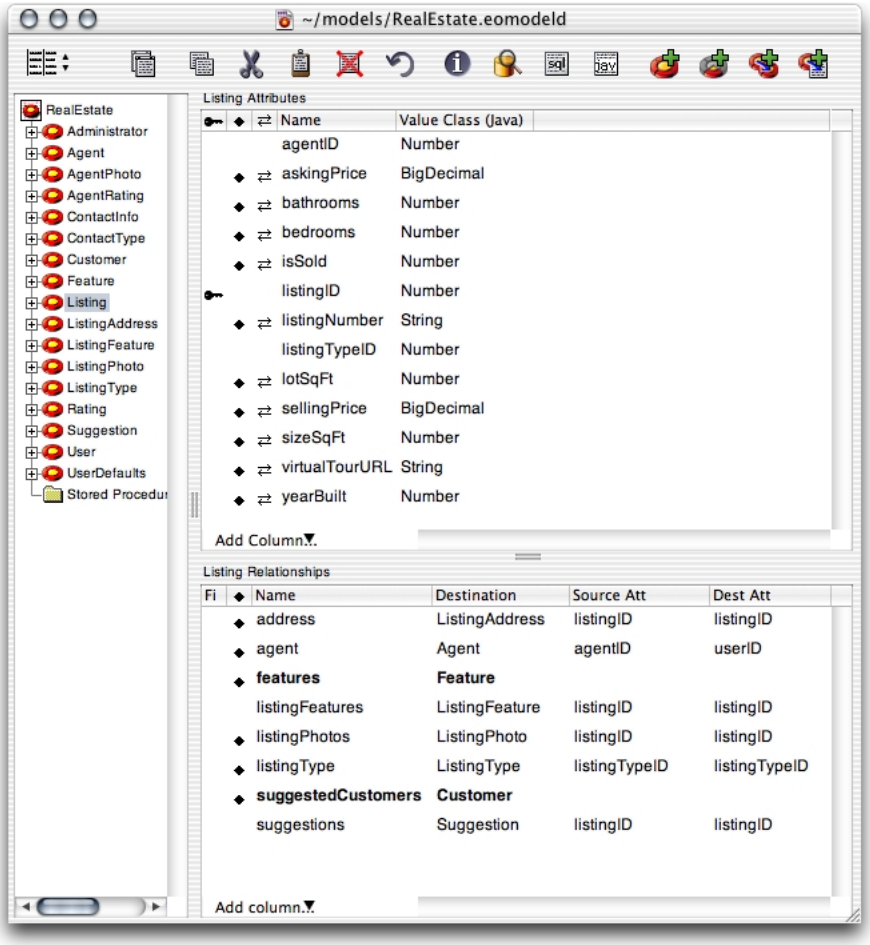
To add Web services and Web service operations to the application, use the Web Services Assistant. With it you define an operation's parameters and return values. In addition, you determine whether the operation's result is returned as an array of enterprise-object instances or as a SOAP document, which can be traversed using the NSDictionary class.

The following sections provide an example of how Direct to Web Services can be used to develop a simple Web service with a single operation.

Developing a Direct to Web Services Application

Just like Direct to Web and Direct to Java Client, a good data model is essential when developing a Direct to Web Services application. Figure 6-3 shows a data model with an entity called Listing, which contains attributes that can store information that a real estate agent collects about properties she wants to sell. Potential buyers can also use it to find their ideal house.

Figure 6-3 Data model with the Listing entity



You configure Direct to Web Services applications using the Web Services Assistant. The Assistant allows you to effortlessly define services and service operations. Consumers can add, update, and delete records using the Web service operations you define. You can also define operations that consumers can use to search your data store.

The first thing you do is create a Direct to Web Services application with Project Builder. You tell the Project Builder Assistant which data models you want to use and it creates an application ready for you to configure with the Web Services Assistant. Figure 6-4 shows how a Web service application that provides a Web service called RealEstate can be configured with the Web Services Assistant.

Figure 6-4 Defining an operation's parameters with the Web Services Assistant

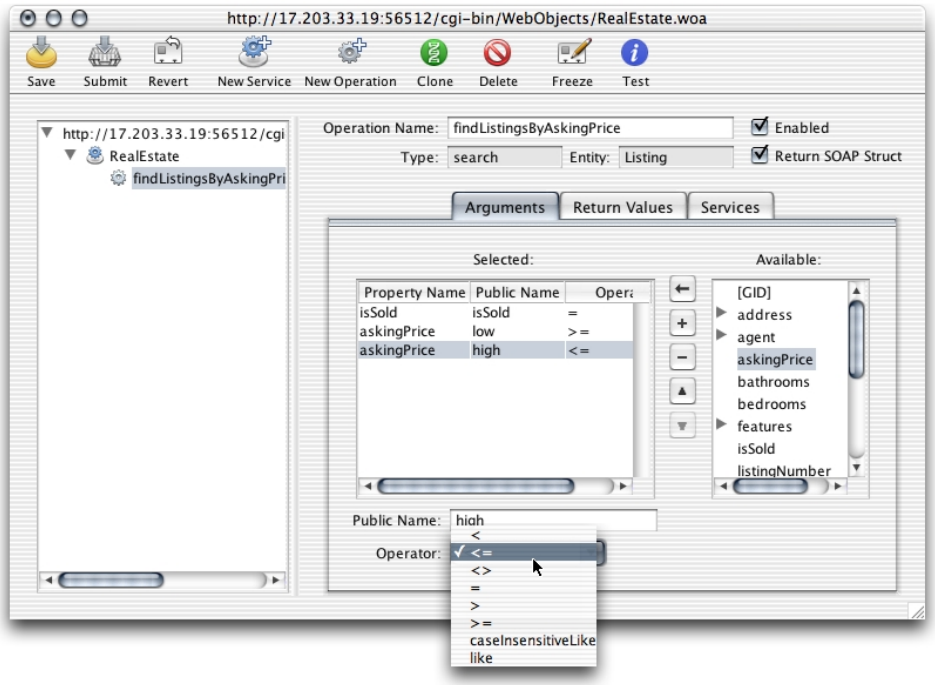
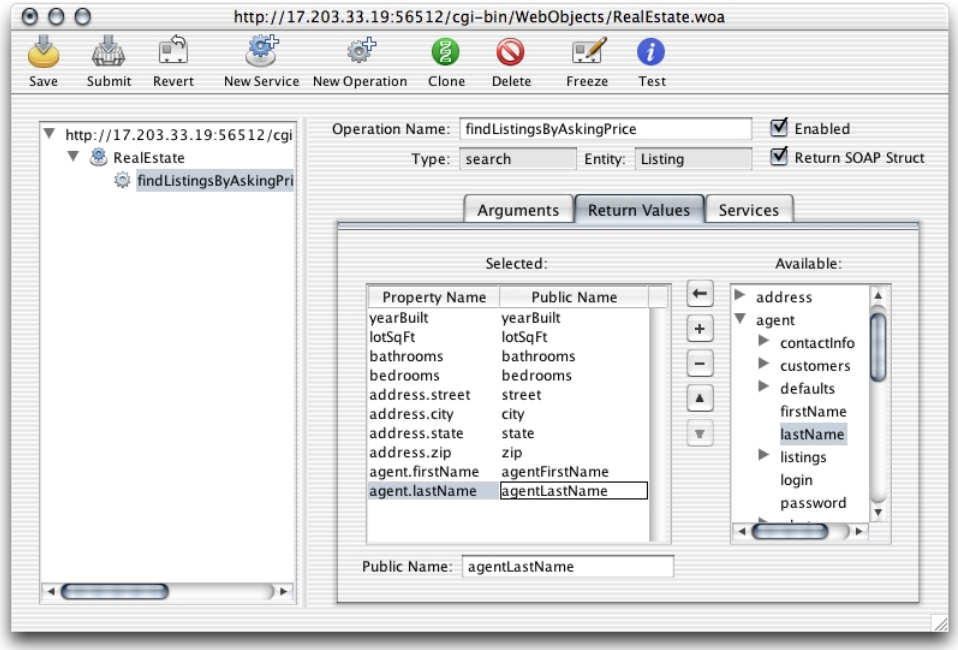


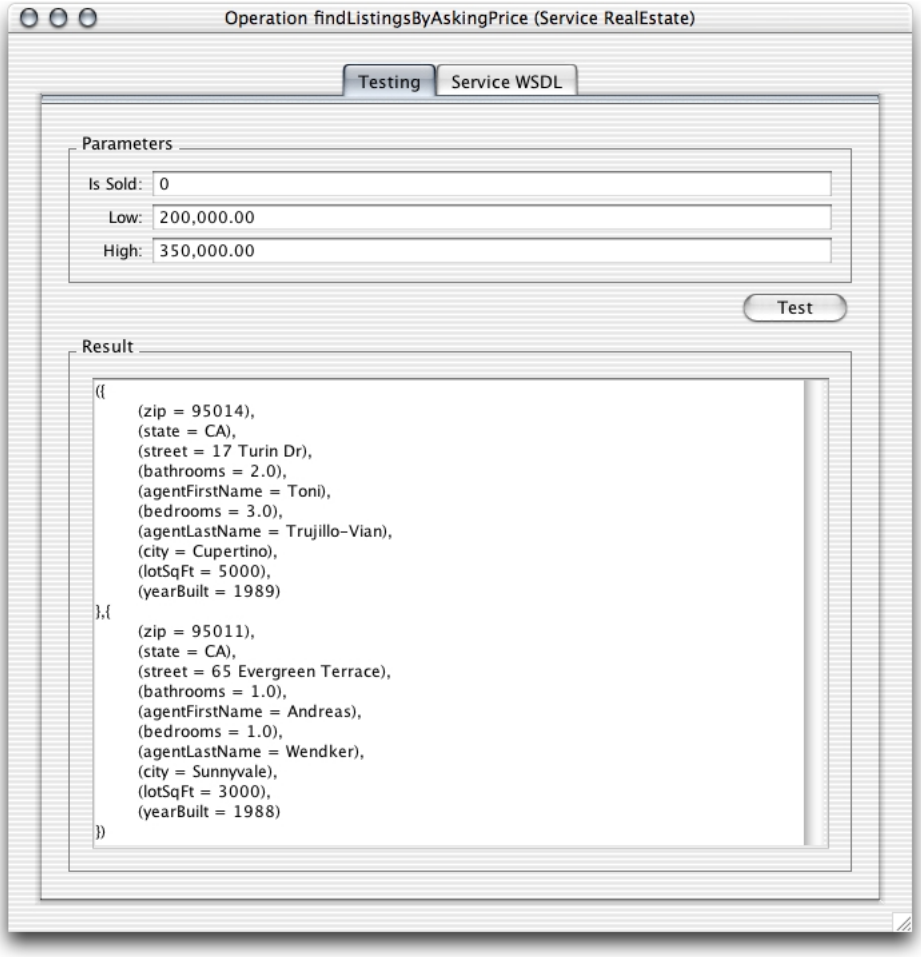
Figure 6-5 shows how return values are added to the operation definition.

Figure 6-5 Adding return values to an operation in the Web Services Assistant



After saving this configuration the `findListingsByAskingPrice` operation of the RealEstate service is ready to be consumed by an application. To facilitate operation debugging, the Web Services Assistant can generate a test client, which you can use to enter arguments and see the corresponding result, as Figure 6-6 demonstrates.

Figure 6-6 Testing the `findListingsByAskingPrice` operation with the Web Services Assistant test client



This section showed you how Direct to Web Services makes it simple to develop Web service operations that access a data store. You can also freeze an operation, in which case Direct to Web Services generates a Web component. After an operation is frozen, you cannot configure it with the Web Services Assistant. You can also use Rule Editor to perform complex customizations.

Consuming Services Provided by a Direct to Web Services Application

This section shows you how Web services developed using Direct to Web Services can be consumed. At first you might want to make sure that you get the correct results using a simple application. Listing 6-5 and Listing 6-6 are part of such an application, while Figure 6-7 (page 113) shows its output.

Listing 6-5 `Application.java` class in simple Web service consumer project

```
import com.webobjects.appserver.*;
import com.webobjects.webservices.support.xml.WOStringKeyMap;

public class Application extends WOApplication {

    public static void main(String argv[]) {
        WOApplication.main(argv, Application.class);
    }

    public Application() {
        super();
        System.out.println("Welcome to " + this.name() + "!");

        testService();
    }

    public testService() {
        ServiceClient serviceClient = new ServiceClient();

        Object[] arguments = { new Integer(0), new Double(200000), new Double(350000) };

        Object[] listings = (Object[])serviceClient.invoke("findListingsByAskingPrice",
arguments);

        System.out.println();
        System.out.println();
        System.out.println("Listings found:");
        System.out.println();
```

```
        for (int i = 0; i < listings.length; i++) {
            WOStringKeyMap key_map = (WOStringKeyMap)listings[i];
            String address = key_map.valueForKey("street") + ", " +
key_map.valueForKey("city") + ", " + key_map.valueForKey("state") + " " +
key_map.valueForKey("zip");
            String agent = key_map.valueForKey("agentFirstName") + " " +
key_map.valueForKey("agentLastName");

            System.out.println("Lot size (square feet): " +
key_map.valueForKey("lotSqFt"));
           System.out.println("Number of bedrooms: " + key_map.valueForKey("bedrooms"));
          System.out.println("Number of bathrooms: " + key_map.valueForKey("bathrooms"));
            System.out.println("Year built: " + key_map.valueForKey("yearBuilt"));
            System.out.println("Address: " + address);
            System.out.println("Agent: " + agent);
            System.out.println();
        }

        System.out.println();
    }
}
```

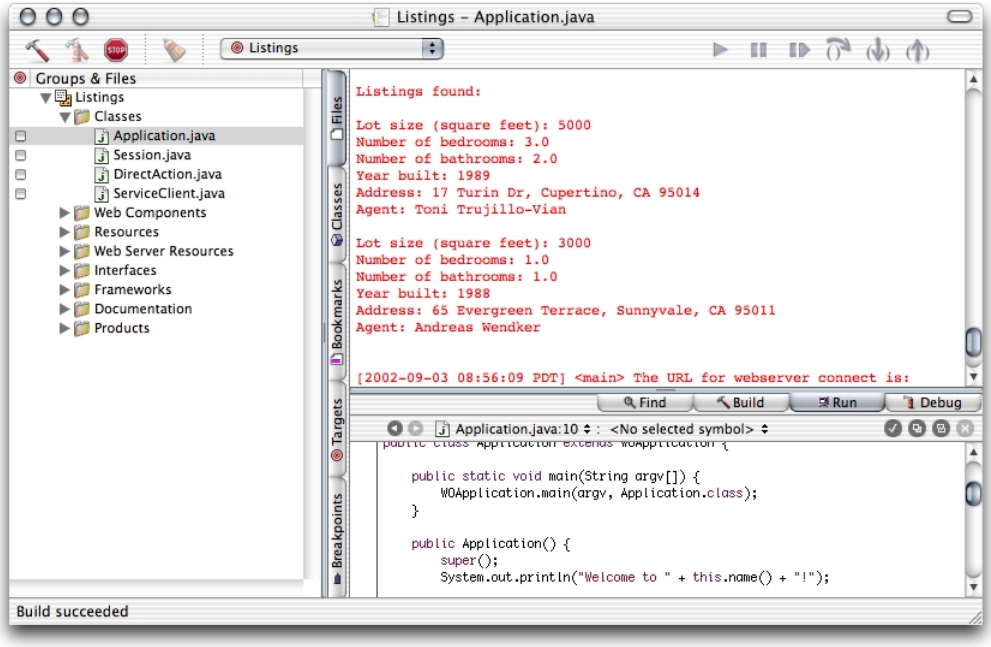
Listing 6-6 `ServiceClient.java` class in simple Web service client project

```
import com.webobjects.foundation.*;
import com.webobjects.appserver.*;
import com.webobjects.webservices.client.*;

import java.net.*;
import java.util.Enumeration;

public class ServiceClient extends Object {
    private WOWebServiceClient _service_client = null;

    public ServiceClient() {
        super();
    }

    /**
     * Invokes a Web service operation.
```

```
     * @return operation result.
     */
    public Object invoke(String operation, Object[] arguments) {
        // Invoke operation.
        Object result = serviceClient().invoke(serviceName(), operation, arguments);

        return result;
    }

    /**
     * Obtains a Web service client object through which service operations
     * can be invoked.
     * @return service client.
     */
    private WOWebServiceClient serviceClient() {
        if (_service_client == null) {
            // Address for the Web service's WSDL document.
            String service_address = "http://noss.apple.com/cgi-bin/WebObjects/
RealEstate.woa/ws/RealEstate?wsdl";

            _service_client = clientFromAddress(service_address);
        }

        return _service_client;
    }

    /**
     * Obtains the Web service name.
     * Normally one WSDL file describes one Web service,
     * but it could describe one or more services.
     * @return Web service name.
     */
    public String serviceName() {
        return (String)serviceClient().serviceNames().objectAtIndex(0);
    }

    /**
     * Obtains a Web service client object through which service operations
     * can be invoked.
     * @return Web service client object.
     */
```

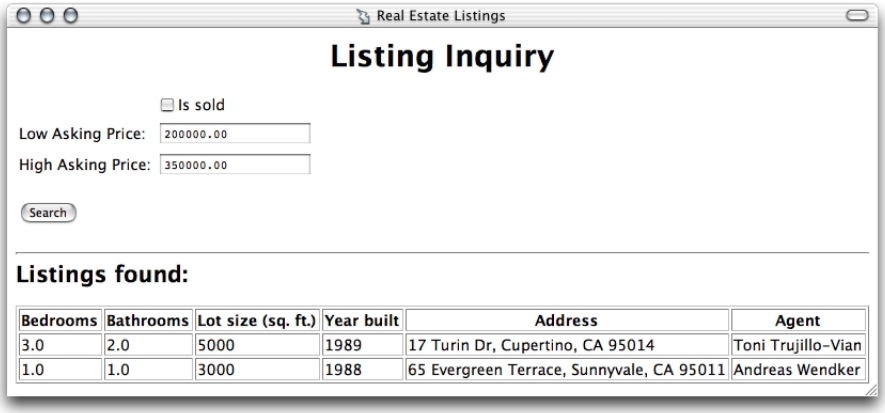
```
    private static WOWebServiceClient clientFromAddress(String address) {
        WOWebServiceClient service_client = null;

        // Create the Web service's URL.
        URL url;
        try {
            url = new URL(address);
        }

        catch (MalformedURLException e) {
            url = null;
        }

        // Get a service client object.
        service_client = new WOWebServiceClient(url);

        return service_client;
    }
}
```

Figure 6-7 Console output of simple Web service client project



Finally, a slightly refined user interface might look like the Web page in Figure 6-8.

Figure 6-8 User interface of Web service client application



Choosing a Web Service Development Approach

Like Direct to Web and Direct to Java Client, you should favor the Direct to Web Services approach because it provides you with a working application with little effort on your part. This approach is especially useful when the main purpose of the application is to provide Web services that perform data-store operations for the service consumers.

When you have tested business logic for which you want to provide a Web service front end, you should use the Web services approach. Of course, you can combine the two approaches to build flexible and robust Web service applications.

J2EE Support

Sun's J2EE (Java 2 Platform, Enterprise Edition) aims to lay out an infrastructure in which solutions (components or applications) from different vendors can work together and share resources. In addition, it provides a framework within which multitier applications can be seamlessly deployed.

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

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

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

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

Enterprise JavaBeans

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

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

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

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

For more information on Enterprise JavaBeans in WebObjects, see *Inside WebObjects: Enterprise JavaBeans*.

JavaServer Pages and Servlets

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

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

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

For more information on JSP and servlets, see *Inside WebObjects: JavaServer Pages and Servlets*.

Java Naming and Directory Interface

WebObjects contains an implementation of Java Naming and Directory Interface (JNDI). Through it WebObjects applications can access multiple naming and directory services, including Lightweight Directory Access Protocol (LDAP), NetWare Directory Services (NDS), Domain Name System (DNS), and Network Information Service (NIS).

JNDI is used in EJB applications to locate beans, data stores, and other resources. For more information on JNDI support in WebObjects, see *Inside WebObjects: Using EOModeler*.

CHAPTER 7

J2EE Support

CHAPTER 8

Choosing Your Approach

Choosing between the four application development approaches, Java Client, Direct to Java Client, Web, and Direct to Web, is the first task you face as a WebObjects developer. 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 119), "User Interface Requirements" (page 120), and "Rapid Development Considerations" (page 121), 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 122).

Internet and Intranet Deployment

The Web application approach is the best approach for deploying applications that users access on the Internet on an intranet. A user on any Internet-enabled computer with a Web browser can access a Web application.

You can also use Direct to Web to create a Web application; however, the user interface generated is generally not flexible enough for public websites. But you can use Direct to Web reusable components as the basis of elaborate Web applications. See "Combining Approaches" (page 122) for more information.

You can also provide and consume Web services on the Internet or an intranet. However, due to the emerging nature of Web service technology, you should take into account security issues before making Web services available on the Internet.

Java Client and Direct to Java Client applications work great on an intranet but are generally unsuitable for Internet deployment because they contain client code that runs on the user's computer. The user must wait for this code to download and the quality of the user's Java virtual machine determines whether the application runs correctly and efficiently. Java Web Start can help in making Internet deployment of Java Client applications more user friendly.

User Interface Requirements

The WebObjects application development approaches differ in the richness and response times of their 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 Web-based user interface employed in Web and Direct to Web applications 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 that 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. However, Java Web Start goes a great way towards eliminating this problem.

Specific Layout and Flow Requirements

If you plan to create a Web application with specific page design and flow requirements, you should use the Web application 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 find the Direct to Java Client user interface well-suited for the task.

Rapid Development Considerations

Using the rapid development technologies that WebObjects provides, you can build an application faster and with less effort than the Web, Java Client, and Web service approaches require. You need to provide only the database–to–enterprise objects mapping (the model) and WebObjects creates your application from it. However, the rapid development approaches produce a very basic interface for your application. You must be experienced with WebObjects development (especially with Rule Editor and Interface Builder) to override it.

These types of applications are good candidates for the rapid development approaches 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.
- **Data-driven applications.** Direct to Web and Direct to Java Client can be used to develop in-house applications for such tasks as bug and feature tracking, customer account management, and writing online help. For internal-use applications, the user interface refinement is not as important as their development time, making these applications ideal candidates for the direct 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.

The Web 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 Web application and link its Web components to 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 application development approaches are summarized in the following paragraphs.

The primary advantage of the Web application approach is its portability. Any user with access to the Internet and a Web browser can use the application. Its disadvantages are the limitations of the Web-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 Web application approach. However, it also allows you to develop data-driven applications extremely quickly. The downside of Direct to Web is that it generates a rudimentary 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. Also, like Direct to Web, Direct to Java Client imposes a particular user interface that may not be suitable for your application.

Table 8-1 shows which of the four approaches is appropriate when considering the scope, type of user interface, and development effort of your project.

Table 8-1 Development approaches for WebObjects applications

	Direct to Java Client	Direct to Web	Java Client application	Web application
Internet		x		x
Intranet	x	x	x	x
Desktop	x		x	

Table 8-1 Development approaches for WebObjects applications (continued)

	Direct to Java Client	Direct to Web	Java Client application	Web application
Web browser		x		x
Simple interface (dynamic)	x	x		
Custom interface (fixed)	x	x	x	x
Web services	x	x	x	x

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:

- *Inside WebObjects: Java Client Desktop Applications*
- *Inside WebObjects: Web Applications*
- *Inside WebObjects: Developing Applications With Direct to Web*
- *Inside WebObjects: Web Services*

Document Revision History

Table A-1 describes the revisions to *Inside WebObjects: WebObjects Overview*.

Table A-1 Document revision history

Date	Notes
September 2002	Revised to reflect changes made in WebObjects 5.2.
	Added chapter about Web services support, including Direct to Web Services.
	Changed references to *HTML-based application* to *Web application*.
	Combined contents of "HTML-Based Applications" and "Direct to Web" chapters in one chapter, "Web Applications" (page 41).
January 2002	Revised to reflect changes made in WebObjects 5.1.
	Combined contents of "Java Client Applications" and "Direct to Java Client Applications" chapters in one chapter, "Desktop Applications" (page 69).
	Added chapter on J2EE support.
December 2000	First version of *Inside WebObjects: WebObjects Overview*.

APPENDIX A

Document Revision History

Glossary

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.

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

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.

Cocoa Object-oriented application-development environment tailored for the production of Mac OS X applications.

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 LAST_NAME column that contains the values for each employee's last name.

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 Web application from a model.

Direct to Web Services A WebObjects development technology that can generate a Web service application from a model.

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 Web components.

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

enterprise object An 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.

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.

Entity-Relationship modeling A discipline for examining and representing the components and interrelationships in a database system. Also known as ER 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 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.

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

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.

JFC (Java Foundation Classes) A set of classes that implement graphical user interface components, also called Swing components.

JDBC An interface between Java platforms and databases.

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

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

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

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

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

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

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

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

nib file File that contains the description of a desktop-based user interface. The file is created using Interface Builder.

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 Application used to manage the development of a WebObjects application or framework.

property In Entity-Relationship modeling, an attribute or 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.

reusable component A component that can be nested within other components and acts like a dynamic element.

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.

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.

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

rule A specification used to customize the interfaces of applications developed using the rapid development technologies of WebObjects.

Rule Editor A tool used to edit the rules in applications developed using the rapid development technologies of WebObjects.

rule system WebObjects feature used in direct technologies, such as Direct to Java Client, in order to dynamically generate an application's interface. The rule system performs this process using data-model information as well as rule sets.

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.

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.

Web Assistant Tool used to customize a Direct to Web application.

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

Web page template HTML file that specifies the overall appearance of a Web page generated from a Web component.

Web Services Assistant Application used to customize a Direct to Web Services applications.

WebObjects Builder An application used to edit Web components.

Index

E

F, G

H

I

J

K

L

M

N, O

P

Q

R

S

T

U

V

W

X, Y, Z