Pro Oracle Collaboration Suite 10*g*

John Watson

Apress®

Pro Oracle Collaboration Suite 10*g*

Copyright © 2006 by John Watson

ISBN-13: 978-1-59059-679-1

ISBN-10: 1-59059-679-X

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jonathan Hassell
Technical Reviewer: Jonathan Ives
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Julie M. Smith
Copy Edit Manager: Nicole LeClerc
Copy Editor: Jennifer Whipple
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Liz Welch
Indexer: Brenda Miller
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

CHAPTER 2

Oracle Application Server Architecture and Components

At first sight, Oracle Application Server 10*g* is a huge and terrifying environment within which to work. As you become familiar with it, it will remain huge but no longer terrifying. A problem that beginners to Application Server often suffer from is being unable to see the wood for the trees. There are so many interacting components that it may seem impossible to grasp the overall picture of what is going on. It has to be said that the documentation may not help with this. This chapter will describe the entirety of Oracle Application Server and drill down to the individual components.

Oracle Collaboration Suite is a set of services that run in the Oracle Application Server environment. This introduction to Oracle Application Server will be sufficient for understanding its function as a part of the technology stack that enables the Oracle Collaboration Suite. The initial configuration of Oracle Application Server after an Oracle Collaboration Suite install will work, but to tune a large and complex Oracle Application Server site for optimal performance requires extensive knowledge. At large sites, there will be an Oracle Application Server administrator in addition to the Oracle Collaboration Suite administrator (not to mention the database administrator, the system administrator, the network administrator, the directory administrator, and all the other specialist roles one finds in a large IT installation).

Oracle Application Server Instances, Farms, and Clusters

In Oracle Application Server terms, an *instance* is a set of processes and memory structures created by running the Oracle Application Server executable code installed into an Oracle home. An Oracle *home* is the directory structure created when an Oracle product is installed. One Oracle Application Server Oracle home can only support one Oracle Application Server instance; by contrast, an Oracle home with the database software installed within it can support multiple running database instances (always provided that they have different names).

Although one Oracle home can only support one Oracle Application Server instance, it is possible for one web site to consist of multiple instances, each installed into its own Oracle home, and the homes can optionally be installed on different machines. This gives Oracle Application Server its scalability and fault tolerance. What appears to be one web site with a single point of entry for the end user may in fact consist of multiple Oracle Application Server instances on multiple machines. By enabling load balancing across these instances, and session failover from one instance to another, an Oracle Application Server web site can be configured to support an unlimited number of concurrent users with phenomenal reliability. The possibilities for security are also comprehensive, as different parts of the web site can be located in various locations in a firewall-protected network environment.

Oracle Application Server instances come in two general forms: *infrastructure* instances, and *middle tier* instances. These two overall types of instance have their own variations. It is the middle tier instances that provide services to end users. Middle tier instances run application software, such as the various applications that make up Oracle Collaboration Suite. An infrastructure instance is never used directly by end users; it provides support services to middle tier instances. These support services include user authentication and authorization, configuration management, and directory services.

An Oracle Application Server *farm* consists of one or more middle tier instances connected to a single infrastructure instance. In a large-scale implementation, there may be several middle tier instances, each on its own machine, all controlled by one infrastructure instance. The infrastructure instance may itself have components running from different Oracle homes on different machines for scalability reasons, but a farm contains only one infrastructure. The infrastructure is therefore a single point of failure for an Oracle Application Server environment; but it is possible to configure failover even for the infrastructure, and there are various options for fault tolerance.

It is possible to create cluster*s* within a farm. A *cluster* is a group of one or more middle tier instances configured to run an identical set of Java applications. Note that clustering is not possible for any type of Oracle Application Server service other than Java applications; it is not, for example, possible to cluster a Portal application. Within a cluster, Oracle will guarantee that all instances are the same, or rather, that they are the same as far as end users are concerned. Any configuration change made to one instance will be propagated automatically to the other instances in the cluster. It also becomes possible to distribute end-user connections across the clustered instances, using heuristics that should guarantee an even workload on each instance, and to have end-user sessions fail over from one instance to another should an instance become unavailable. Clusters greatly enhance the manageability and fault tolerance of the web site. The farm/instance/cluster structure can be represented as an entity-relationship diagram, as in Figure 2-1.

The front end to an Oracle Application Server, the process to which users connect, is a web listener, the Oracle HTTP Server (OHS). This is in fact a distribution of Apache. All Oracle Application Server instances, whether infrastructure or middle tier, come with an Apache web listener. This web listener can optionally be front ended by a Web Cache. As a general rule, the Web Cache is used for middle tier instances but not for infrastructure instances. The Web Cache is an acceleration service that will improve response times to end users and may also assist with overcoming some limitations of Apache and security issues. Since Apache is the entry point to an Oracle Application Server web site, that is where this discussion will start.

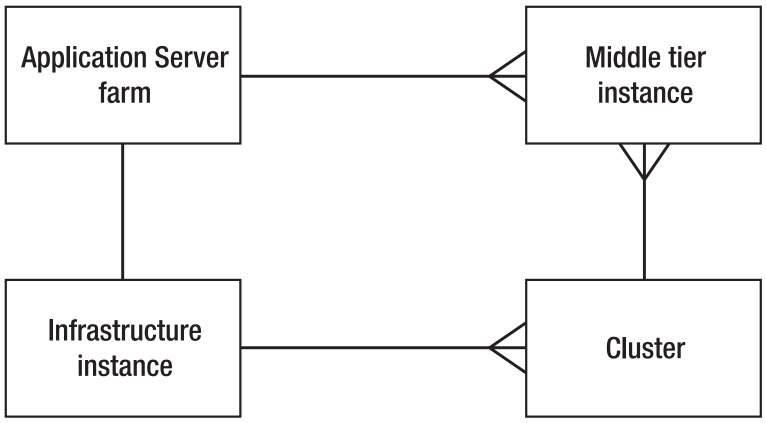

Figure 2-1. *The farm-instance-cluster relationships*

The Apache Web Listener

A web listener is a process that monitors one or more ports on one or more addresses for incoming URLs from browsers. It parses these URLs into filenames, locates the files on disk, and copies them back to the browser. This is a very simple process; you can write your own web listener in a few dozen lines of Java. This is why there are numerous web listener programs that can be downloaded free of charge from any number of sites. It is also why Oracle did not write its own web listener. Instead, it chose to distribute the Apache web listener.

Note that the term *web server* is seriously confusing, as it combines the terms *web listener* and *application server*. For this reason, try to avoid using the term *Apache web server.* The term *HTTP server* is more acceptable (and is frequently used on the Apache Software Foundation web site as a description of the Apache web listener) but in general it is best to use the terms *web listener* to refer to the front-end processes that manage users' web sessions, and *application server* to refer to the back-end processes that run application software.

Apache is a web listener that is available under terms similar to the standard GNU General Public License from the Apache Software Foundation. It has been developed over many years by public-spirited people (and also people who want to show off their skills) who have donated the fruits of their labor to mankind in general. The result is a very powerful product, but also a product that shows many of the problems of software that has been developed by a large and disparate community without much control. You may find that there are odd inconsistencies in the way things are done in the Apache environment and also several ways of achieving the same result. There are also parts of Apache functionality that have fallen into disuse but are still part of the product, and parts that must have been developed for some particular user that no one else will ever use. But in spite of these failings, Apache has become the de facto standard for web listeners. Why pay for a web listener when the one used by 80% (figure for 2005 from the Apache Software Foundation web site) of the world's web sites is available for free? There is also a vast amount of information available on Apache. A decent book shop could have whole bookcases devoted to it.

The Apache license places restrictions on how it can be "sold" and distributed. Oracle has negotiated an arrangement whereby it distributes Apache as compiled executable code. You must use the Oracle distribution with Oracle Application Server. If you download the Apache source code and compile it yourself, it may well work as a front end to Oracle Application Server, but Oracle Corporation will not support you.

A Web Listener or an Application Server?

A basic web listener is just a file server. It receives URLs from browsers and passes back files. This functionality is only adequate for the most basic of web sites: applications that consist of linked HTML documents. To convert a web listener into a web server, it must be possible to run code on the server machine to generate web pages dynamically.

The first technique developed for dynamic page generation was CGI (Common Gateway Interface). CGI is a means for launching executable code on the web listener machine in response to user requests. The URL specifies a filename, but rather than finding the file and copying it back to the browser, the web listener will launch an operating system shell, load the file into the shell, and run it. Clearly, the filename must be an executable program; if the program generates any output, the output, not the file, is sent back to the browser.

CGI works. A CGI program can do anything. But it is not a very efficient processing model. For every CGI invocation, the web listener must launch an operating system shell. This means that a busy web site could be running hundreds or thousands of shells concurrently, launching and terminating them with a frequency that will bring the operating system to its knees. It is also hard to control CGI execution. In principle, the web listener just launches the program and waits. If the program hangs or gets into a loop that consumes 100% of CPU resources, the web listener knows nothing about this. Modern web listeners (and Apache is certainly one of them) do have more efficient ways of running CGI programs: they can monitor the processes and maintain persistent shells. But the inherent inefficiencies of CGI remain. There may also be security issues with CGI; the mechanism was not developed with security in mind. This is why Oracle chose to develop its own facilities for running software for end users—software written in Java and PL/SQL and run in a fashion that will scale to Internet proportions and can provide absolute security.

The Apache license does not permit distributors to modify Apache itself. But Apache does come with a mechanism whereby its behavior can be modified legally by creating Apache modules. An Apache module is a dynamically linked program that is invoked when appropriate URLs are received by the web listener. The web listener parses the URL and, rather than mapping it onto a file for download to the browser, will dynamically load and link the appropriate module and pass the request through to that module for further processing. The mechanism for mapping a URL to a module is based on the path element of the URL. For example, some paths could instruct Apache to launch the module that enables CGI, in which case the file and extension part of the URL will specify the file to be executed through the CGI interface. Another path could be mapped onto the Oracle PL/SQL gateway (in which case the filename and extension will refer to a package and a procedure) or to the OC4J (Oracle Containers for J2EE) module (in which case the file and extension will refer to a Java class). Anyone can write an Apache module to do anything at all. The only difference between the Oracle distribution of Apache and other public domain distributions is in the modules that Oracle has written.

The ability to generate web pages on demand, probably based on database content and constructed in response to user input, is beyond the power of a web listener. For this you need an application server. The combination of a web listener communicating with users over HTTP and an application server behind it that can run software for the end users makes up a web server.

Apache Modules

When Apache is enhanced with modules such as those supplied by Oracle, it becomes a routing process that receives URLs and dispatches them to the appropriate service, which is implemented by the module. If the service generates any output, this is returned via Apache to the browser that invoked it. In this way, the Apache web listener acts as the front end to the Oracle Application Server.

There are a number of Apache modules developed by Oracle Corporation that are shipped with Oracle Application Server. Principal among these are the following:

modplsql: This module accepts URLs that request the running of PL/SQL procedures within an Oracle database. Depending on the path element of the URL, it will use Oracle Net, Oracle's proprietary client/server networking protocol, to connect to a particular database, either with a hard-coded username and password or after generating a login prompt. The filename and extension elements of the URL nominate a PL/SQL package and procedure to be run within the database.

modoc4j: This module accepts URLs that request the running of Java servlets. A *servlet* is a Java program that can be invoked with HTTP and can return a page of HTML. The path element of the URL determines which OC4J instance to pass the request to. The filename and extension nominate the Java class file to run. The Java is not run by modoc4j but by the OC4J instance, which is a process external to Apache.

modosso: This module enables the connection to the Single Sign-On (SSO) service. Single Sign-On–aware applications use modosso to locate the Single Sign-On server and read and write the Single Sign-On cookie.

modossl: Apache can be configured to use secure sockets; there is a public domain module for this. Oracle's implementation of secure sockets, which is integrated with the Oracle Internet Directory (OID), is enabled with this module that replaces the standard Apache secure sockets module.

Every installation of Oracle Application Server comes with an Apache web listener. In the case of an infrastructure instance, Apache is the point of contact for middle tier instances when they need infrastructure services, such as Single Sign-On. For middle tier instances, Apache is the service to which end users' browsers send URLs. Apache dispatches these URLs to appropriate middle tier Oracle Application Server services and then returns whatever output they generate to the browser.

Apache modules are dynamically linked libraries that enhance Apache's capabilities. They run as threads within Apache child processes; some modules carry out work themselves, others are control structures for external processes. modplsql is an example of the first type of module—it logs sessions onto databases and invokes PL/SQL procedures within them. modcgi and modoc4j are examples of the latter type of module. modcgi does not run code; it launches and controls operating system shells that run code. Similarly, modoc4j does not run Java applications; it routes requests through to OC4J instances, external to Apache, that run them. So the Apache web listener is merely a dispatcher process that routes requests to application services provided by modules, which may themselves route requests on to processes that are external to the Apache environment. See Figure 2-2 for this routing model.

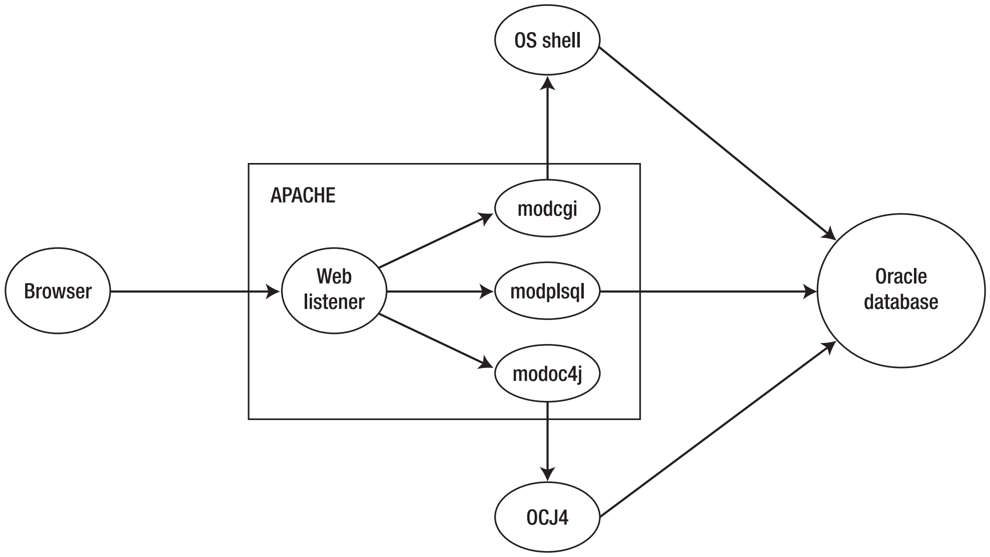

Figure 2-2. *Apache as a router to application services*

The Apache Processing Architecture

The Apache processing architecture is based on a parent process that runs all the time, and child processes that are launched on demand. On UNIX, the child processes are real processes; you can see them at the operating-system level by running the `ps` command. On Windows, the child processes are separate threads in a single multithreaded process. In either the UNIX or the Windows processing model, the dynamically linked modules run as threads within each child process. The processing model is a child process launched by the parent process whenever a URL is received. It is the parent process that monitors the ports and addresses on which the Apache listener is listening.

In principle, the child process is active only for the duration of the request-response cycle; it is launched on demand and terminated after servicing the request. In practice, Apache will usually be configured to keep the child processes alive for a period during which they can service several requests. Furthermore, Apache will usually prespawn a number of child processes. This can improve performance dramatically, as it reduces the overhead of continually spawning and destroying processes according to user demands. HTTP version 1.1, which is supported by all modern browsers and the current release of Apache, also allows for the possibility of keeping the connection between browser and web listener open for a series of request-response cycles, rather than making and breaking the connection for each request-response cycle, as should be required by the stateless nature of HTTP.

Configuring Apache

Apache is configured with the `httpd.conf` file. The layout of this file and its location are defined by the Apache Software Foundation, and Oracle Corporation has very little freedom to modify this. The file is read when Apache starts, and not subsequently. This means that any changes will not be effected until the listener is restarted.

In all Oracle Application Server Oracle homes, there will be an Apache directory structure that includes (among others) the directories in Table 2-1.

Table 2-1. *Some Apache Directories*

Directory	Description
`ORACLE_HOME/Apache`	Apache root
`ORACLE_HOME/Apache/Apache`	Core Apache files
`ORACLE_HOME/Apache/Apache/bin`	Executable Apache code
`ORACLE_HOME/Apache/Apache/conf`	Apache configuration files

The Oracle distribution includes (among other things) the directories in Table 2-2.

Table 2-2. *Some Oracle-Specific Apache Directories*

Directory	Description
`ORACLE_HOME/Apache/modplsql`	PL/SQL gateway configuration files
`ORACLE_HOME/Apache/modoc4j`	modoc4j configuration files

The `httpd.conf` file, located in the directory `ORACLE_HOME/Apache/Apache/conf`, is a series of directives, all in the form of *token value* pairs. For example, if you want your Apache listener to monitor port number 8000, you would add `Listen 8000` to the `httpd.conf` file, where *Listen* is the token and *8000* is the value. Or to allow Apache to map URLs that include the path `/pub` onto the physical directory `d:\Apache\public`, you would add this directive with the Alias token:

```
Alias /pub "d:\Apache\public\"
```

Note that for this directive the *value* has two attributes: first the virtual path that will be used in URLs, then the physical path that the virtual path will be mapped on to. Following these additions, if a user issues the URL `http://host.domain:8000/pub/index.html` to your `host.domain`, Apache will retrieve the file `index.html` from the physical directory `d:\Apache\public` and send it back to the browser. If the file does not in fact exist in that directory, depending on other directives, Apache may return an "Error 404 – file not found" message, or perhaps some default page, or perhaps a listing of the files that are available in the directory.

A vital directive is `include`, as in

```
include "c:\oracle\ias9i\Apache\Apache\conf\oracle_apache.conf"
```

This instructs Apache to append the file `oracle.conf` to the `httpd.conf` file, and to action all the directives it finds therein. These directives (which are for the most part other `include` directives) are the pointers to the dynamic link libraries that make up the Oracle modules and the files that configure them:

```
# Advanced Queuing - AQ XML
include "c:\oracle\ias9i\rdbms\demo\aqxml.conf"
#OiD DAS module
include "c:\oracle\ias9i\ldap\das\oiddas.conf"
include "c:\oracle\ias9i\Apache\jsp\conf\ojsp.conf"
#Directives needed for OraDAV module
include "c:\oracle\ias9i\Apache\oradav\conf\moddav.conf"
#
include "c:\oracle\ias9i\xdk\admin\xml.conf"
#
include "c:\oracle\ias9i\Apache\modplsql\conf\plsql.conf"
include "c:\oracle\ias9i\Apache\oradav\conf\oradav.conf"
```

Because Oracle Application Server stores configuration data in text files such as `httpd.conf` and in a centrally maintained repository, Oracle does not support manual editing of the configuration files. You should only adjust the Apache configuration by using the management tools provided with Oracle Application Server.

Some directives may well require adjustment from their default settings. The infrastructure instance may also need adjustments. These examples of directives that will usually require changes from default are from the Apache `httpd.conf` file on an Oracle Collaboration Suite middle tier instance running on Linux:

- `StartServers 5`: Specifies that Apache will launch only five child processes on startup. When there are more than five concurrent requests, Apache will have to spawn more processes dynamically.
- `MaxClients 150`: States that Apache will permit only 150 concurrent connections. If more than this is attempted, users will receive Error 500 messages.
- `MaxrequestsPerChild 0`: Limits the number of requests a process will handle before terminating; *zero* means an infinite number. If any libraries have memory leaks, and some do on some platforms, the default of zero (which means an infinite number) will cause problems. Setting it to, for example, 10,000 can do no harm and may help.
- `ExtendedStatus on`: Controls the amount of information returned by the URL `http://host.domain:port/server-status`. *Off* may be less of a security risk than *on*.
- `ServerAdmin you@your.address`: Determines the e-mail address that will be displayed on some pages for users to mail the administrator. This default is not very helpful.

The Infrastructure Instance

An infrastructure instance is optional if the web site is only going to do fairly simple things. But for a complex application environment such as Oracle Collaboration Suite, it is required. An infrastructure instance offers these services to middle tier instances:

- Configuration management
- Oracle Internet Directory
- Single Sign-On
- Certificate Authority

The last three are known collectively as *Identity Management*. It is possible for these services to be distributed over multiple Oracle homes on multiple machines for performance and fault tolerance. The infrastructure includes an Oracle database that is used to store configuration information for all the middle tier instances in the farm, the data maintained by the Oracle Internet Directory, and the repositories required by some middle tier components. The Oracle Internet Directory is an LDAP-compliant directory service used by middle tier applications for maintaining user accounts and privileges, and also for name resolution. The Single Sign-On facility allows end users to connect to many middle tier applications on many middle tier instances after responding to just one logon prompt. The Oracle Certificate Authority can issue digital certificates to enable use of Secure Sockets Layer (SSL) communications and authentication.

The infrastructure instance is front-ended by an Apache web listener, just like any other instance that gives access to these services either by end users or (more frequently) by applications running on middle tier instances.

Configuration Management and the Metadata Repository

A farm has a central repository of configuration data for all its instances. This is known as the *metadata repository* and is stored in an Oracle database. The metadata repository stores information about the overall configuration of the farm and the infrastructure, the various server components configured in each middle tier instance, and the individual applications deployed to these components. A farm has only one metadata repository, but middle tier instance configuration information can in fact be stored in three different locations.

First, the operation of the various components of a middle tier instance is actually controlled by text files in that middle tier's Oracle home. The format and location of these text files is component-specific. For instance, Apache (described earlier in the "Apache Web Listener" section) is controlled by a file `httpd.conf`, whose layout is specified by the Apache Software Foundation. Oracle Corporation has very little control over the format of this file. Alternatively, the Web Cache (described in the "Web Cache" section) is controlled by two files, `webcache.xml` and `internal.xml`, which are formatted with XML (Extensible Markup Language) tags according to Oracle's specifications. Java components are configured with a plethora of XML files; some of these are under Oracle's control, but others must conform to the standards developed by the Java community. It is these text files that are read by the components when they start, but the information within them is echoed out to a central repository.

The second possible location for configuration data is a file-based repository. This is a centrally stored set of XML files containing data for all middle tier instances. The file-based repository does allow central control of multiple middle tier instances, and if an organization does not wish to go to the trouble (and the expense) of creating an infrastructure instance, this may be acceptable if the web site is small and simple. A repository of some kind is a prerequisite for creating Oracle Application Server clusters. The file-based repository is not adequate for managing an Oracle Collaboration Suite web site. For that you need an infrastructure.

The third possible location for configuration data is the database created as part of an Oracle Application Server infrastructure installation. This is a preseeded database containing tables for storing the metadata repository. It also has the schemas needed for storing the Oracle Internet Directory tables, the Single Sign-On code (Single Sign-On does not store any data; it uses Oracle Internet Directory as its data store), and repositories for various middle tier services, such as Portal.

Within a farm, middle tier configuration information is always in two places: the component configuration text files, and the repository. If these two get out of sync with each other, there may be problems. This is a situation that should never occur. But if for example an administrator edits a configuration file by hand, rather than with the management tools that are provided, the two sources of information will differ. This can also happen if an instance is not available when management commands are issued. There are tools provided for propagating changes from the text files to the repository (known as a *configuration update*) and from the repository to the text files (known as an *instance resynchronization*) if the two sources of information have diverged.

The metadata repository database can also be used for storing Oracle Internet Directory data and for the component repositories; the necessary preseeded schemas are always created at installation time. Using the one database for all these functions may give acceptable performance but, if necessary (depending on workload), a separate infrastructure installation can be used to store the directory data, and a third can store the component repositories. Thus, the one infrastructure can be distributed across multiple Oracle homes on multiple machines. If performance and fault tolerance are critical (and they may well be—the metadata repository is vital to the function of an Oracle Application Server web site), the database can be configured as a Real Application Cluster (RAC) database, and also as a Data Guard primary database. Given these two options, it should be possible to configure a zero-downtime and zero-data-loss environment.

Note that the metadata repository database should *never* be used for storing user data, and indeed the license does not permit this legally. Apart from database administration tasks, such as performance tuning and backup, it should not be necessary to ever log on to the metadata repository. But apart from its specialized use, a metadata repository database is a database like any other. As installed out of the box, it is not optimized in any way and will require the usual tuning and other maintenance activities. The exact release of the database shipped with the infrastructure will depend on the release of Oracle Collaboration Suite; there are compatibility issues between Oracle Application Server, the database, and Oracle Collaboration Suite that must be considered before attempting any upgrades.

Oracle Internet Directory

LDAP is the standard for Internet directories. Like HTTP, LDAP is a layered protocol. It runs over TCP and is in fact a TCP implementation of the X500 directory standard. Several global software vendors sell Internet directories. Some major ones are Novell's eDirectory; Sun ONE Directory Server from Sun Microsystems; Microsoft Active Directory; and the Oracle Internet Directory. These have varying degrees of compliance to the LDAP standard, and therefore varying degrees of interoperability.

The information stored in an Internet directory will typically be used for user authentication and authorization and for name resolution. The nature of this type of work is such that there will be many more lookups of data than updates. For this reason, LDAP directories store information in a *tree*. Tree structures are optimal for data retrieval (unlike the structures used in a relational application). A directory tree consists of entries. An *entry* is defined as an instance of an object class. The *object class* defines the structure of the entry in terms of its data attributes. By analogy with a normalized relational database, an entry could be thought of as a row, and the object class as a table. An Oracle Internet Directory stores its data in an Oracle database; the necessary schemas are in the preseeded database created as part of an Oracle Application Server infrastructure installation.

An Oracle Internet Directory instance is implemented as a single listening process that monitors a port for requests from LDAP clients, and a number of server processes that action the requests. These processes must all reside on the same machine, as the communication between them is via whatever IPC (interprocess communication) protocol is provided by the operating system. The LDAP clients can be on any remote machine, since LDAP runs over TCP (though there may be firewall and security issues), and the database storing the information can also be on a remote machine, since the server processes connect to the database using Oracle Net (aka SQL*Net, for those DBAs who still insist on using Oracle 8 terminology). This is illustrated in Figure 2-3.

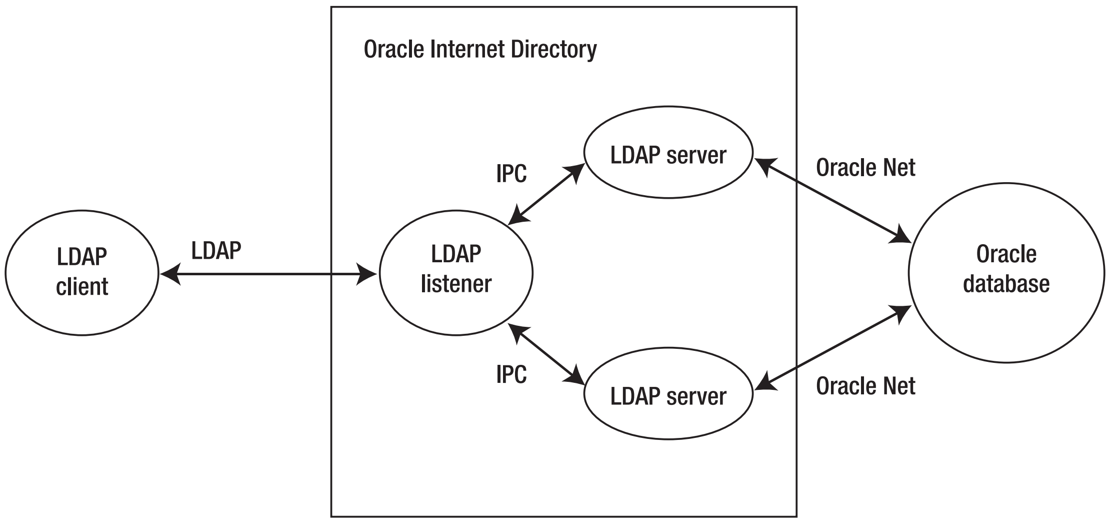

Figure 2-3. *The flow of an LDAP request*

The flow of information for an LDAP request over various protocols is therefore

1. A client process passes a search request to the listener process with LDAP.
2. The listener process passes the request to a server process with IPC.
3. The server process passes the request to the database with Oracle Net.
4. The database executes the query.
5. The database returns the result to the server with Oracle Net.

6. The server returns the result to the listener with IPC.

7. The listener returns the result to the client with LDAP.

A single Oracle Internet Directory instance will be inadequate if it cannot service in a timely fashion the volume of requests being received. Additional server processes can be launched, but the instance only has one listener process that must manage all the concurrent requests. If this is a bottleneck, another instance can be launched on the same machine from the same Oracle home that will monitor a different port, or instances can be launched on separate machines that have an Oracle Application Server infrastructure Oracle home installed. All these Oracle Internet Directory instances can, and in most cases should, connect to the same database; they make up one logical directory server, so that the same set of information can be disbursed by many LDAP instances. In this way, access to a single Oracle Internet Directory can be distributed across a number of physical locations, addresses, and ports, giving both scalability and fault tolerance. Another reason for launching additional Oracle Internet Directory instances is to make use of SSL. If it is necessary to use the SSL implementation of LDAP, a separate Oracle Internet Directory instance (or indeed a separate group of instances) is needed to listen for this variation of the LDAP protocol.

Oracle Internet Directory instances aren't controlled by the administrator directly. It is done through a fly-by-wire approach implemented by a monitor process. The various control utilities (graphical and command-line) do not actually control the instances—they insert rows into a table in the database storing the directory that contains instructions to launch, terminate, or modify instances. The monitor process then reads these commands and actions them. The monitor process is the `oidmon` process. The `oidmon` must be running before the `oidldapd` processes—the directory server processes—can be launched. On a Windows installation there is only one multithreaded `oidldapd` process; on a UNIX or Linux installation there are multiple single-threaded processes.

In some cases, one Internet directory is not sufficient. There may be a number of reasons for this, but the most obvious are scalability, network overhead, and single-point-of-failure. Creating a number of directories that contain the same information means that the workload can be distributed; end users can connect to a local directory rather than having to use wide area links; and if one directory becomes unavailable, requests can be diverted to another directory that contains the same directory tree. But using multiple directories does imply an additional management overhead, because in a multidirectory environment it is vital that the directories should all return the same result for a given query, meaning the databases must contain the same data.

The LDAP standard ensures that multiple directories contain identical copies of the directory tree through the concept of directory replication. Each directory runs an LDAP replication server, which propagates changes made to the entries in its local directory to the replication servers of the remote directories where they are applied. Clearly, mechanisms must exist to handle the possibility of conflicts. Conflicts will arise if the same entry is updated simultaneously in different ways in two different locations. Oracle Internet Directory handles this situation by using the Oracle database and Oracle Net, rather than LDAP. Directory replication is in fact implemented with database multimaster advanced replication for the transport of changes, a standard Oracle database facility that includes automatic conflict resolution capabilities.

Similar in concept to directory replication is directory synchronization. The term *replication* means maintaining identical directory trees in different directories. Replication can only function between identical Internet directories—directories purchased from the same software vendor. *Synchronization* handles the situation where it is necessary to exchange information between incompatible directories purchased from different vendors, or perhaps directories that have different object class definitions. LDAP is supposed to be a vendor-independent standard, so any Internet directory should be able to replicate to any other; but in practice, it doesn't work like that. The various Internet directory vendors have created their own routines for replication that exploit their own platform capabilities and usually do not work with each other. Also, it may well be that different vendors have implemented object classes in different ways. For example, all directories will have an object class that defines a user, but the class could be called *user* in one vendor's directory, and *party* or *person* in another. Furthermore, the different directories will have used different attributes and different data types in the object classes that define their users.

Directory synchronization defines a mechanism whereby changes can be propagated between different directories. The objective is not to keep the directories identical, but merely to ensure that enough information is passed between them to eliminate the need for duplicating the data entry process. For example, if a user changes his password in an application that uses Microsoft Active Directory to store authentication details, the password change should also be reflected in the Oracle Internet Directory, which is used to authenticate the user to Oracle Collaboration Suite applications, so that the password change is automatically reflected there as well. The Oracle Internet Directory implements this capability through the synchronization service. The synchronization routines take certain attributes from entries in one directory and map them on to an LDAP message for transmission to another directory. The actual transfer of information is done automatically over LDAP, but the administrator must manually define the mappings first. Synchronization does mean that basic operations (such as creation or deletion of user accounts) can be propagated between different vendors' directories.

In an ideal world, it would not matter whose directory you used, but it doesn't work like that. For example, there is no way that a Windows file server will accept authentication from an Oracle Internet Directory. Most organizations will end up running more than one directory, but the replication and synchronization services will ease the pain of having to do this. An environment with directories from several vendors, each supporting a different set of applications, can become an administration nightmare—not so much for technical reasons (though these may be an issue) as for political reasons. With user definitions existing in several places with different maintenance procedures, which is to be considered the "source of truth" when they diverge? How long is divergence permitted? Is it legal to make certain updates at all sites? Issues such as these are the subject of business process analysis and must be thoroughly researched and documented by your business analysts.

The Single Sign-On Service

A vital problem of a modern IT environment is that of maintaining user accounts in many applications. This is a problem for administrators and for end users. All end users in a typical organization today will have accounts on different corporate systems for, at least, e-mail, file server, human resources self-service, internal requisitions, and diary management. Then each end user will have more accounts on systems specific to their role in the organization. The end result is that users have a dozen or more usernames and passwords. Not only is managing this situation very time-consuming for the uses, it is also fraught with risk for the organization itself. History shows that a large number of users in this situation resort to practices that are highly dangerous for security, such as

- Using easily remembered (and therefore guessable) passwords
- Keeping hard copies of login details in obvious places
- Using the same password for all accounts

The last possibility is particularly worrying if users reuse the password they select for internal systems when they register with external public services where it may be available to anyone. But even if users are disciplined in maintaining security in this environment, there is a large price to be paid in terms of business efficiency.

Single Sign-On centralizes management of passwords. To the end user, it appears as though he only logs on once, typically to a corporate portal, and then has access to all the application services he is authorized to use without any further login prompts. In reality it is a bit more complicated.

To enable Single Sign-On, first the users must be registered in an Oracle Internet Directory. This is the repository of the users' identities, including their passwords. Then all the applications that are going to use Single Sign-On must be written with the appropriate toolkits provided by Oracle to make them Single Sign-On aware. The applications themselves do not do any user authentication; they delegate authentication to the Single Sign-On service. Authorization is still controlled within the application.

When a user contacts a Single Sign-On application, the application checks the user's browser for a cookie that identifies the user: the Single Sign-On cookie. If this cookie is there, the user is logged into the application immediately, with whatever privileges his username entitles him to. If this cookie does not exist (or if it has expired) the application will redirect the user to the Single Sign-On server. The Single Sign-On server generates a login screen, authenticates the user with a username/password prompt, and if successful sets the Single Sign-On cookie in his browser. The user gets redirected back to the application he originally asked for, the cookie is read, and he is logged on without further prompts. Then for the duration of the cookie's validity he can access any Single Sign-On–aware application in the same manner without any prompts. So although he is in fact being logged on to each application individually, the process is transparent to the user. Since the password is maintained centrally in the Oracle Internet Directory, when it is changed it is changed for all the applications that are using this mechanism. This avoids having separate passwords maintained by each application.

Single Sign-On is written in PL/SQL. It is a set of procedures in a schema created in the metadata repository database. These generate login screens to prompt for a username and password, and then use LDAP to request an Oracle Internet Directory to authenticate the username-password pair. The cookie that is set in the browser is an encrypted, nonpersistent cookie. It should therefore be impossible to hijack a user's session by copying the cookie to another machine. It is also possible (and recommended) to use the HTTPS protocol (which is HTTP with secure sockets) for all communications between browsers and the Single Sign-On service in order to encrypt the password during its transmission from browser to Single Sign-On server.

Figure 2-4 illustrates the Single Sign-On connection cycle, which consists of a number of steps:

1. The browser issues a URL contacting an SSO-enabled service. This request does not include the SSO cookie authentication.
2. The service returns a redirection URL, instructing the browser to contact the SSO server with a logon request.
3. The browser contacts the SSO server with a logon request.
4. The SSO server sends a logon prompt window to the browser.
5. The browser returns a username and password to the SSO server.
6. The SSO server passes the username and password to the OID for validation, using the LDAP protocol. Note that steps 1 through 5 have used HTTP, as will 8 and 9.
7. The OID returns a positive authentication (for the purposes of this example) back to the SSO server, using LDAP.
8. The SSO server sends an SSO cookie to the browser with a redirection URL instructing the browser to contact the service it originally requested.
9. The browser contacts the SSO-enabled service, this time with the SSO cookie, and will be logged in to it without further authentication required.
10. Any URLs contacting other SSO-enabled applications will result in an immediate logon until the lifetime of the SSO cookie expires.

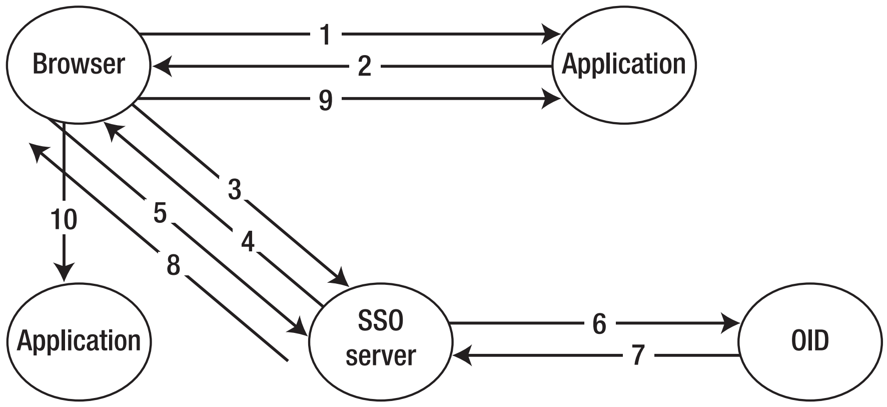

Figure 2-4. *Single Sign-On request flow*

Identity Management and Digital Certificates

The capabilities of the Oracle Internet Directory and the Single Sign-On service can be leveraged by enabling the full Identity Management capability. This uses account provisioning to create, modify, and delete accounts automatically in a number of applications and the certificate issuing authority to generate digital certificates.

Account provisioning centralizes the creation and maintenance of users' accounts in a number of applications. In a typical organization, every employee will have accounts for several organization-wide systems as well as those specific to his job, and possibly for various means of external access. The workload of maintaining these individually is significant and also raises the likelihood of security problems. Enabling account provisioning means that once an employee is registered in, for example, the HR system, he will automatically have accounts created for him in a number of other corporate systems to which all employees should have access. Then when the employee leaves the organization, all his accounts on all systems will be terminated; there will be no danger of any delay in, for example, canceling his dial-in access.

If security is to be based on SSL communication, all users must be given a digital certificate. This is the document identifying who they are, and in an Oracle environment is stored in the Oracle Internet Directory. Managing digital certificates can be automated by using the Oracle Certificate Authority. This is an Oracle Application Server infrastructure service that will generate digital certificates from a parent certificate installed in the infrastructure.

The parent certificate must be purchased from a third-party supplier. This can be from any recognized certificate-issuing authority, though there may be procedural (or even political) reasons for favoring one supplier over another. There will also be cost implications. A certificate for an installation that will have many thousands of users will be far more expensive than a certificate valid for a few hundred users, and the cost does vary from one supplier to another. It is possible to use an Oracle-supplied certificate shipped with the product, but while this will be adequate for internal use (because your own users will trust authentication based upon it) this will not usually be adequate if users will interact with people from external organizations. External users will want to see certificates that can be verified with a recognized certificate-issuing authority.

Oracle Collaboration Suite makes use of the provisioning service to create accounts for users in all Oracle Collaboration Suite Applications and issues and revokes digital certificates as necessary.

Distributing Infrastructure Components

An Oracle Application Server farm consists of a group of Oracle Application Server instances controlled by one metadata repository. It is not possible to have multiple metadata repositories in one farm; in effect, the repository defines the farm. But there is no necessity to run the Single Sign-On service and the Oracle Internet Directory off the same Oracle home as the metadata repository. They can even run off an infrastructure that is part of a different farm.

It is possible to run a complete infrastructure from one Oracle home. During the installation of Oracle Application Server, if one takes the option to install an infrastructure instance, by default all infrastructure components are selected. But if desired, one can deselect Internet Directory and Single Sign-On and instead point the new installation toward an already available Internet Directory and Single Sign-On server.

By distributing the components of the infrastructure, it is possible to provide a highly fault-tolerant and scalable environment. This is discussed in Chapter 17.

Connecting to an Infrastructure Instance

The front end of an infrastructure instance, as for any Oracle Application Server instance, is its Apache web listener. It is not customary to run a Web Cache in front of this listener, as it is very unlikely that that there would be any benefits. Nearly all requests to an infrastructure instance are of such a nature that the Web Cache would have to pass them straight through to Apache without caching the results, so there would be no point.

End users rarely contact an infrastructure instance directly. On contacting a Single Sign-On–enabled application, users will be redirected to the infrastructure for authentication, but that is all. When they are logged in to an application, the application may be continually contacting the Oracle Internet Directory and making directory lookups to check authorizations and retrieve other information, but the user knows nothing of that.

The exception to this rule is the Delegated Administration Service (DAS). This is a Java application deployed to the infrastructure instance that lets users manage their registrations within the Oracle Internet Directory.

The URLs to connect directly to the Identity Management services are

- `http://host.domain:7777/pls/orasso`: This is the URL for the Single Sign-On service running on the infrastructure installed on `host.domain` with an Apache web listener running on its default port of 7777. It will generate a screen with a list of services that have delegated authentication to the Single Sign-On server and a link for a login prompt. While you can issue this URL directly, users will usually be redirected to it by Single Sign-On–enabled applications.

- `http://host.domain:7777/oiddas`: This is the URL for the DAS. This is a self-service application that lets users change their passwords (stored in the Oracle Internet Directory and validated by Single Sign-On) and change various other items of personal information. Users with sufficient authorization can use this tool to manage other users—create or delete them and change their privileges.

Middle tier Oracle Application Server instances connect to their infrastructure whenever necessary. They log on to it using an encrypted password that is derived from (among other things) the hostname of the node they are running on. For this reason there are issues with changing hostnames in an Oracle Application Server environment, but it can be done. If a middle tier instance cannot contact its infrastructure, the effect will depend on the application being used; there is not necessarily an immediate failure. Some application server services may only need an infrastructure service for Single Sign-On. In that case, if the infrastructure is unavailable, users who are already logged on and authenticated can continue working unaffected; only new login requests will fail. Other applications may be making continuous use of the infrastructure, perhaps to construct menus as users navigate through the application; these will fail immediately.

As an administrator you will connect to and control the infrastructure instance through the tools provided as discussed in Chapter 5.

The Middle Tier Instance

A middle tier Oracle Application Server instance runs services that are used by end users whose entry point to the instance is the Apache web listener. Every Oracle Application Server instance has one (and only one) web listener. If the web listener can service the request from a browser by itself (typically because the request is for a static file) it will do so, but if the URL maps onto an Application Server service, identified by the path element of the URL, the web listener will pass the request to the appropriate module.

Middle tier instances come in three forms; the choice is made at installation time. These options (from simplest to most complex) are known as J2EE (Java 2 Enterprise Edition) and Web Cache; Wireless and Portal; and Business Intelligence and Forms. The more complex installations include all elements of the less complex installations; thus, a Business Intelligence and Forms instance will include J2EE, Web Cache, Wireless, and Portal.

The Middle Tier Instance Types

The simplest middle tier instance is the J2EE and Web Cache instance. This is an Apache web listener augmented with the Oracle-supplied Apache modules that enable the PL/SQL gateway and OC4J. It also comes with a preconfigured Web Cache, which can be reconfigured or even disabled if desired. The PL/SQL gateway is a mechanism whereby end users can from their web browsers invoke PL/SQL code that will run within an Oracle database. Java code is compiled to run within a virtual machine—a computer that doesn't actually exist. The Java runtime environment is a simulated environment that supports the Java Virtual Machine (JVM) standard. OC4J is Oracle's Java runtime environment. In addition to the standard Java requirements, it also provides excellent facilities for control, monitoring, and fault tolerance. One middle tier Application Server instance may run many OC4J instances, but it can only have one Apache web listener.

The second possible middle tier installation is Portal and Wireless. This instance type requires an infrastructure to be already available. If during the installation dialog you cannot give the location of an infrastructure instance, the installation will fail. A Portal and Wireless instance includes all the J2EE and Web Cache facilities, plus the Portal and Wireless components. Portal is a tool for developing web sites that can deliver customized content. Portal applications make use of toolkits that enable use of the Oracle Internet Directory and Single Sign-On. Wireless can be used to develop applications that use the standard wireless protocols (such as WML or cHTML) to develop interfaces that make applications usable on any device. For instance, your software will be accessible to users in their cell phones as well as on their PCs.

The third middle tier instance type is Business Intelligence and Forms. In addition to the facilities provided by the J2EE and Web Cache and Portal and Wireless instance types, a Business Intelligence and Forms instance includes the products that were marketed in earlier days under the names Developer 2000 Forms and Reports and Discoverer. Forms is a tool for developing and delivering applications that have a much richer user interface than those possible with web applications based on HTML. The standard web application environment is based on HTML documents, but HTML has limitations. For example, an HTML application can have radio buttons, but it cannot have rolling combo boxes. The HTML standard simply doesn't have tags for creating rolling combo boxes, but Forms does. The term *Business Intelligence* covers Reports and Discoverer. Reports is a facility for scheduling jobs—or running them in real time—that will extract information from an Oracle database and present it in a formatted fashion. Discoverer is a facility for allowing end users to query the database while shielding them from the complexities of the application's relational design.

A J2EE instance can be installed without an infrastructure instance, though during the installation you will be asked if you want to associate the instance with an infrastructure. If you choose not to associate the instance with an infrastructure the instance will not be part of a farm, and it can only operate in a self-contained fashion; there will be no possibility of load balancing or fault tolerance with other instances, and the applications deployed to it will not be able to make use of an Oracle Internet Directory or the Single Sign-On service. The other middle tier types cannot exist without an infrastructure, which must have been installed first.

The PL/SQL Gateway

The PL/SQL gateway is installed with all Oracle Application Server instance types. It is an Apache module that can handle URLs that invoke PL/SQL procedures stored within a database. The procedures can perform any program logic that may be required, but whatever else they do, they will usually generate a page of HTML to be returned to the browser. This page will be created by using the PL/SQL Web Toolkit, a set of packages installed within the database that can be used to produce web pages. These pages can be windows that prompt users to enter data, or windows that display information retrieved from a database, or any combination. The PL/SQL Web Toolkit is a very efficient mechanism for generating HTML applications that use data stored in an Oracle database.

The PL/SQL gateway is enabled by the file `plsql.conf` which is included in the `oracle_apache.conf` which is itself included in `httpd.conf`. A simple file could be as small as just two lines:

```
LoadModule plsql_module c:\oracle\ias9i\bin\modplsql.dll
include "c:\oracle\ias9i\Apache\modplsql\conf\dads.conf"
```

This file nominates the dynamic link library that enables the PL/SQL gateway and has an `include` directive for `dads.conf`. This is the file that creates DADs (Database Access Descriptors). A DAD instructs Apache to direct certain URLs to modplsql by mapping a path in the URL to a database connect string. The DAD can include database login credentials; if it does not include these, a login prompt (in the form of a "403 – Authentication Required" window) is sent back to the browser. The following is a simple `dads.conf` file containing two DADs:

```
<Location /pls/orasso>
    SetHandler pls_handler
    Order deny,allow
    Allow from all
    AllowOverride None
    PlsqlDatabaseUsername orasso
    PlsqlDatabasePassword orasso
    PlsqlDatabaseConnectString infra_as.bplc.co.za:1521:inf
    PlsqlDefaultPage orasso.home
    PlsqlDocumentTablename orasso.wwdoc_document
    PlsqlDocumentPath docs
    PlsqlDocumentProcedure orasso.wwdoc_process.process_download
    PlsqlAuthenticationMode SingleSignOn
    PlsqlPathAlias url
    PlsqlPathAliasProcedure orasso.wwpth_api_alias.process_download
    PlsqlSessionCookieName orasso
</Location>
<Location /hr>
    SetHandler pls_handler
    PlsqlDatabaseConnectString hrdb
</Location>
```

The `<Location>` tags name the DADs and create virtual paths that Apache will recognize as those to be routed to modplsql. In this example, the first DAD is path `/pls/orasso`, which is the standard DAD for connecting to the Oracle Application Server infrastructure Single Sign-On service. The second DAD, `/hr`, is a simpler one that would have been created by the administrator.

If the DAD includes a logon name and password, as in the first DAD, these credentials will be used to log on to the database nominated by the connect string, which can be either a full hard-coded address (as in the first DAD) or an Oracle Net connection identifier (as in the second DAD). If there isn't a username/password, as in the second DAD, modplsql will generate a 403 "Authentication Required" message that will prompt for a username and password. If the URL includes the `file.extension` elements, these will be used to locate a procedure in a package (which must be available to the schema used to log on) to be executed; otherwise, if there is a nominated default procedure (such as the `orasso.home` procedure in the first DAD), then that procedure will run.

To complete the example, if the Apache web listener with the `dads.conf` file listed previously is running on the machine `mid1_as.bplc.co.za`, and the URL `http://mid1_as.bplc.co.za:7777/hr/demo.hello?name=john` is issued by a browser, and there is a PL/SQL package within the database referred to by the connect identifier hrdb created with this code

```
create package demo as
procedure hello(name varchar2);
end;
/

create package body demo
as
procedure hello(name varchar2)
as
begin
htp.print('hello '||name);
end;
end;
/
```

then, following a login prompt, a page will be sent back to the browser that says "hello john." This is a very elementary use of the PL/SQL Web Toolkit (part of which is the htp package) for generating pages of HTML.

Substantial use is made of the PL/SQL gateway throughout Oracle Collaboration Suite. In particular, Single Sign-On is a PL/SQL application invoked through the DAD or virtual path `/pls/orasso`. Try it; if your installation uses the default ports and `host.domain` is the node where your infrastructure is running, the URL `http://host.domain:7777/pls/orasso` will connect you to Single Sign-On. Portal is also to a large extent a PL/SQL web application, as may be the applications Portal connects users to.

Oracle Containers for J2EE (OC4J)

Java applications come in two general forms: Java servlets and Java beans. A *Java servlet* is a Java application that communicates with HTTP. It can be launched in response to a URL from a browser and will usually generate a page of HTML in reply. Java servlets are used for writing user interface code. *Java beans* do not do user interface work. They are software constructs that manipulate data, and you communicate with them by sending them messages that ask them to do something, using protocols such as RMI (Remote Method Invocation) or JMS (Java Message Service).

Java beans then come in several forms. A *session* bean contains code that implements application logic. An *entity* bean represents data; it will be populated with a row from a database. Typically, you communicate with Java beans by sending them messages using the RMI protocol, though there is the variant known as *message-driven beans* (MDBs) with which you communicate with JMS.

While it is possible for a Java servlet to perform application layer functions and to communicate with a database, a more common application architecture would be to use a Java servlet to manage the user interface layer of the applications, and Java beans to manage the data processing layer. The end users will communicate with a servlet by issuing a URL over HTTP. The Apache web listener receives the URL and passes it to the module modoc4j. modoc4j passes it on to the OC4J instance to which the servlet has been deployed. The servlet will generate and manage whatever user dialogs are necessary and eventually pass a request to a session bean, which will manipulate data represented by entity beans that connect to a database. Breaking up an application into many tiers assists developers and maintenance substantially by providing abstraction between the various layers. It becomes possible, for instance, to redesign the user interface without affecting the data manipulation code, or to change the relational structures of the data storage without affecting anything else.

An OC4J instance is a process running externally to Apache. It can launch and control servlets and Java beans, which will run within a JVM. The actual runtime environment, the JVM, is provided by the host operating system. So if your application server is a Solaris machine, you will be using Sun's JVM; whereas if you are using a Linux server you will be using the Linux JVM. Solaris and Linux implement the JVM very differently, but neither your programmers nor Oracle Application Server need to worry about this, because both environments provide a JVM that conforms to the standard API (application programming interface). OC4J is a control structure for starting JVMs and managing the Java applications running within them.

The user's entry point to an OC4J is, as for all Oracle Application Server components, a URL sent to an Apache web listener. The `httpd.conf` file will include a line such as this

```
include "c:\oracle\ias9i\Apache\Apache\conf\mod_oc4j.conf"
```

which points to the file that configures the module modoc4j. The following is a simple version of this file:

```
LoadModule oc4j_module modules/ApacheModuleOc4j.dll
<IfModule mod_oc4j.c>
    Oc4jMount /j2ee/*
    Oc4jMount /hrj demo
</IfModule>
```

This file first specifies the dynamic link library that is the modoc4j code and then creates two virtual paths with the `Oc4jMount` directive. This is a token that takes two values: the first is a virtual path, the second is the name of an OC4J instance to which to send URLs with that virtual path. So if the Apache instance receives this URL

```
http://mid1_as.bplc.co.za:7777/hrj/helloworld?name=john
```

Apache will pass it through to an OC4J instance called `demo`, which will load and run the Java class `helloworld` using the value `john` for the argument `name`. The first `Oc4jMount` directive only has a single value: the virtual path `/j2ee/*`. All URLs that include this path will be dispatched to the default OC4J instance, which is known as the `home` instance.

An Oracle Application Server instance may have many OC4J instances created within it, each optimized for running a particular set of Java applications. The configurable elements include the number of JVMs that the instance will launch; the amount of memory they are allowed to take for Java heap space; and the databases to which the Java applications may connect. It is possible to deploy a Java class as an individual file, but this is rarely done. Java applications that are deployed to an OC4J instance should be packaged as Java archives: these are ZIP files that may contain many class files of Java servlets and beans, as well as static pages of HTML and image files that together make up an application.

If a number of J2EE and Web Cache middle tier instances are part of a farm, it is possible to group them into clusters. Clustering gives Oracle Application Server its fault tolerant capabilities. The infrastructure guarantees that all clustered instances are absolutely identical in three respects: the way modoc4j is configured; the Java applications are deployed to them; and, to a certain extent, the way their Apache web listeners are configured. This means that end users can connect to any instance in the cluster and run the same software. It thus becomes possible to load-balance users across instances by using Web Cache or a third-party tool. Furthermore, within a cluster, the Apache web listener for one instance can route URLs to any OC4J in any other instance, which allows for a second level of load balancing. But even more significant, it is possible to configure clustered OC4J instances to replicate the state of the running applications between one another. This gives fault tolerance; if one OC4J instance crashes, the end users' sessions can be reestablished against a surviving instance without the user being aware of the problem.

There are various communications protocols used in the OC4J application environment. First, end users' browsers use HTTP to send URLs to the Apache web listener. These requests are routed to modoc4j within the Apache executable, then modoc4j uses AJP1.3 (Apache Jserv Protocol version 1.3) to send the request on to the appropriate OC4J instance, which could be running in a different middle tier instance on a different machine. The OC4J instance will load and run the requested Java class, which is probably a servlet. The servlet may then use RMI to contact a Java bean, which could again be running in a different OC4J instance on a different machine. Then the bean might use Oracle Net to contact a database, and LDAP to contact an OID.

Portal

A portal isn't really an application in its own right; it is the front end to many applications. But Oracle Application Server Portal is much more than just a web page with links to a few applications. It is a development environment for designing portals and provides a secure and manageable environment for connecting to enterprise software services and information resources. A portal page makes data from multiple sources accessible from a single location, with personalization for (and by) each user. Portal applications are fully integrated with Single Sign-On and Oracle Internet Directory.

Portal uses a data repository to record the components that can be used to make the web pages it will present to users. These pages are constructed dynamically for each user according to his identity. The Portal repository is a set of schemas in an Oracle database. These schemas are found in the database that is created as part of an Oracle Application Server infrastructure. The workload on the Portal repository database is significant, and for this reason it will often be desirable to have a separate database for this purpose, rather than to colocate the portal repository with the Oracle Application Server metadata repository and the Oracle Internet Directory data.

The applications accessible through a Portal page are reached through *portlets*. A portlet appears to a user as a link. The portlet is generated by Portal according to specifications provided by the application developer. The application must be written with the Portal development kits in order to make it Portal-aware. It will also be integrated with Single Sign-On and the Internet Directory.

Portal is implemented by the Parallel Page Engine (PPE). The PPE is the OC4J application that constructs Portal pages on demand for end users. It uses modplsql to retrieve information from the Portal repository. The PPE is capable of generating pages that can be displayed with standard HTML in a browser, or on a wireless device using whatever markup language is appropriate for the device. It is Portal that makes Oracle Collaboration Suite applications available on wireless devices such as mobile telephones and PDAs.

Forms, Reports, and Discoverer

Forms, Reports, and Discoverer are now shipped as part of Oracle Application Server. Oracle Collaboration Suite does not use them, so this discussion is merely a brief mention in order to complete the description of the Oracle Application Server.

Forms and Reports were distributed in earlier releases as Developer 2000. Forms is an application development environment for building user interfaces. These can be much more sophisticated than those possible in the HTML environment, no matter what tools (Java or PL/SQL, for example) are used to generate the HTML. Forms do not use HTML for displaying information; they use a Java applet downloaded to the user's terminal. This means that the applications are much easier to use, but it also means that the user needs a relatively powerful machine to display them and a fast network because of the additional communication needs of the applet. Forms applications run superbly across an intranet, but the demands they place on the network and the display devices may not always make them suitable for an Internet deployment.

The Forms viewing applet is responsible for local window management on the user's terminal. The Forms logic is run by a Forms runtime engine. This is a process running on the Oracle Application Server machine. Thus Forms applications are truly three-tier: The database tier handles the data access, implemented with SQL; the application server middle tier runs the application, using a Forms Server to launch runtime engines for each concurrent user; the user interface tier in a Java applet on the user's terminal manages interaction with the user. A significant processing capability is needed on all three tiers.

Reports is a tool for retrieving information from a database and formatting it for display. A report is designed by a programmer but run by an end user. The report can be designed to accept numerous parameters, allowing the user a degree of flexibility in information retrieval. The output can be either HTML or PDF, or it can be wrapped up in XML tags. The reports are generated by a Reports server running on the Oracle Application Server middle tier. Unlike Forms runtime engines, which are launched on demand for each user, reports engines are prespawned—a fixed number that waits for requests. If there are more concurrent requests than Reports engines, a queuing mechanism manages the situation.

Discoverer is a tool for querying a database, but unlike Reports the output can be controlled completely by the user. A Reports report can be customized through parameters at runtime, but it must still have been predesigned. A Discoverer report can be designed completely under the control of the user requesting it. The advantage of Discoverer over any other query tool is that it places a layer of abstraction between the database and the user—the Discoverer End User Layer (EUL). The EUL is a set of views that can conceal the complexities of the database design from the user. Rather than having to comprehend a normalized relational structure, the user can design reports based on a simplified view of the database content. It must be remembered, however, that this ease of use comes with a price: the EUL views have to be populated on demand. So while Discoverer makes life easy for end users, it can put extra strain on the database.

Discoverer report definitions, known as *workbooks*, are stored in the database along with the EUL, and the reports themselves are run by Java processes on the middle tier. End users need nothing more than a browser to design, launch, and view ad hoc reports.

The Web Cache

All Oracle Application Server instances are front ended by an Apache web listener. This is all very well for conformance to standards, but there are some limitations in what Apache can do, particularly with regard to performance and scalability. The Web Cache is designed to address these. It can also assist with security issues and resilience against failures. But the most obvious advantage of using a Web Cache is performance: it will accelerate delivery of data to your users.

The ideal situation is that neither end users nor developers will even know that the Web Cache is there. But the users will perceive a performance far greater than if it were not.

Apache's Limitations

An Apache web listener is restricted in terms of the number of concurrent connections that it can service. The standard distribution can only handle 256 connections. It is simple to change this: edit the source code to adjust the limit and recompile. Oracle Corporation has done just that. The Apache distributed with Oracle Application Server can handle 1,024 concurrent connections (the `MaxClients` directive must be changed to enable this). But this does not mean that Apache will be particularly efficient at supporting 1,024 concurrent connections, or that it will handle an overload of requests in an elegant fashion. The Web Cache can address these scalability limitations of Apache in two ways. First, it can handle thousands of concurrent connections. Second, it can distribute these connections across a pool of Apache web listeners. The algorithms used for this distribution can load-balance connections according to the capacities of the machines on which each Apache web listener is running and, if necessary, limit the number of requests sent through to each web listener to avoid overloading.

Apart from issues of scalability, there is also the question of whether it is actually desirable to have Apache handle every request from end users. Access to web sites is typified by many repeated requests for the same or similar information. The most obvious example is the introductory page to a web site, commonly the `index.html` page. Every user who contacts the web site starts with this page, so it must be transmitted unchanged thousands or perhaps millions of times a day. The method by which Apache would do this is not particularly efficient; it may well read the file off disk for every request. The Web Cache can keep copies of popular documents in memory with an indexed access method, and serve them from memory far faster than Apache could serve them from disk. In this way, the Web Cache can improve response times to end users and reduce the workload on the Apache web listener(s).

Web Cache Architecture

The Web Cache is a specialized web listener. It follows the standard web listener processing model of a single process that monitors one or more ports on one or more addresses for incoming URLs, but from there things change. If it can satisfy the URL from its memory cache, it will do so; otherwise, it will pass the request back to a web listener. The web listener will process the URL as it would process one received directly from a browser. It will retrieve the document from disk (or perhaps pass it further back to an Oracle Application Server service) and return it to the source of the URL, in this case, the Web Cache. The Web Cache then returns the document (whether a static file read by Apache off disk, or a file generated dynamically) to the browser. But as it does this, if it is configured with appropriate rules, it will keep a copy of the document in its memory cache. Then the next time the same URL is received, from the same browser or another, the Web Cache can serve it out immediately from memory without having to bother the Apache web listener.

So the Web Cache is the front end to the Apache front end; it routes requests from browsers to the web listener and forwards the responses while keeping copies of them ready for identical requests.

The efficiency of this is limited by the likelihood of repeated requests for the same information. There are also security and data integrity issues. What if a popular document that is in the cache is updated on disk? Will users see the current version from disk, or the old version from cache? What about documents that need to be generated dynamically? There are certainly issues to consider with using a Web Cache, but provided that the rules it uses to determine which documents to cache and for how long are appropriate, it can improve the performance perceived by end users dramatically, reducing the strain on the application server(s) behind it. The default preconfigured rules should be suitable for most sites; they will accelerate delivery of content to users without compromising security or accuracy.

Fault Tolerance and Load Balancing with the Web Cache

A large web site supporting thousands of concurrent connections will consist of several middle tier Oracle Application Server instances, probably part of one farm. All the middle tier instances, including their Apache web listeners, will be configured identically, so that users can connect to any one and access the same applications. They will all be connecting to the same backend database(s) and may even be configured into a cluster.

This is all very well, but a mechanism is required for spreading the incoming URLs across all the web listeners. For this you need a very expensive device: a layer-four switch that can take the incoming connections and redirect them to the various addresses of the web listeners. Web Cache does exactly that, but in a highly intelligent fashion. It can not only do layer-four switching for load balancing but also layer-seven switching. It can direct some URLs (identified with wild cards, using POSIX pattern recognition) to some web listeners (or groups of web listeners), and other URLs to other web listeners, using whatever load-balancing limitations you care to program into it. If a web listener becomes unavailable, the Web Cache will detect this and cease to send it URLs until it is again available.

If your web listeners are overstressed, Web Cache can cover up for this by serving data from its cache, even if the data is out of date, if the rules you configure permit this. This also gives protection against denial-of-service attacks on your site; it is virtually impossible to crash a Web Cache by sending it a flood of requests, whereas conventional web listeners are very vulnerable to this type of overload. The only limitations on the number of concurrent connections a Web Cache can manage are determined by your operating system, and for most platforms this limit is many thousands.

By putting one, or several, Web Caches in front of your web site, you should be able to improve performance and reliability dramatically. For this reason, the Web Cache is enabled by default on all middle tier Oracle Application Server instances. It comes preconfigured with a set of caching rules that will certainly help all web sites; but if you choose to customize these rules to your own environment, the benefits will be even greater.

Oracle Application Server, Web Cache, and Security

All web applications should consider security to be an absolute priority. If e-commerce is to succeed, all users must have complete confidence in the impossibility of hacking the web site. Apart from its capabilities for scaling and accelerating a web site, Web Cache should also be integrated into its security structures.

A vital part of security is the use of firewalls. A *firewall* is, in effect, a router. It is a dual (or more) homed device that connects two (or more) networks. It accepts connection requests from addresses on one network interface and forwards them to addresses on another network interface. But whereas a standard router will forward all requests, a firewall will be configured with rules. These rules can restrict the protocols that will be routed, the addresses from which requests will be accepted, the addresses to which requests will be sent, the ports that can be used, or any combination of these variables. Using several layers of firewall and placing different servers behind the different firewalls can make hacking a web site virtually impossible. Each firewall can be configured with a different set of rules so that traffic that is permitted by one is not permitted by another. In combination, no hacker will be able to get through to server systems storing sensitive data. A general principle is that servers storing data should be in a totally protected zone where they can only be contacted by your own (one hopes secure) systems, but servers offering only processing services can be in a zone of intermediate security where they can be contacted by external clients.

Web Cache can significantly enhance that security of a web site. A possible scenario is illustrated in Figure 2-5.

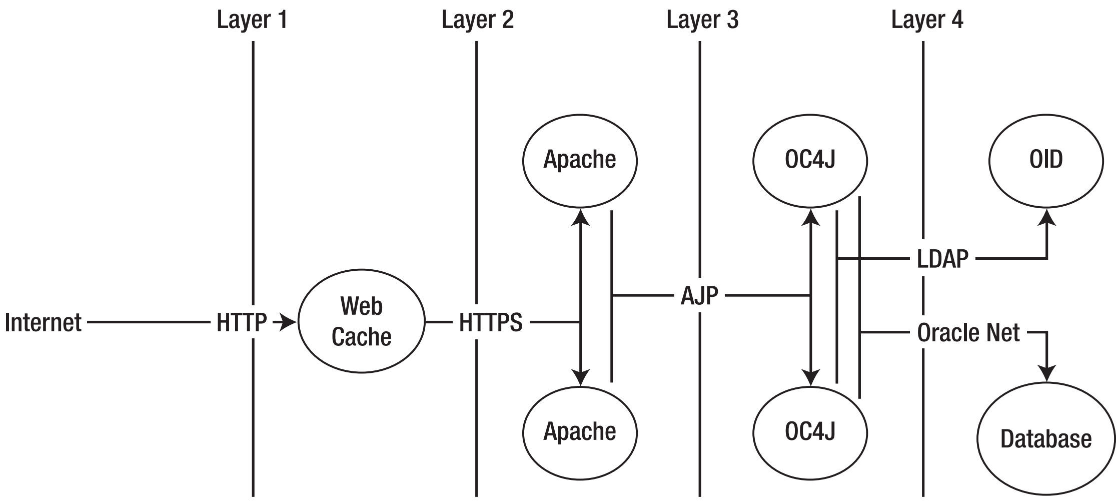

Figure 2-5. *A multilayer firewall topology*

The external firewall is configured to accept only one protocol on one port: HTTP, typically on port 80. It is further configured to route HTTP to only one port on one address, that of the Web Cache. Any other protocol, or any requests for any other addresses, will be rejected. The Web Cache can be configured to forward requests only to the Apache web listeners, behind a second firewall. This firewall will be configured to accept requests from only one address and only one protocol: HTTPS from the Web Cache. It will route these requests only to the Apache web listeners in the next protected firewall zone on the appropriate port. Note that the Apaches can themselves be configured to accept only HTTPS traffic, authenticated with the digital certificate of the Web Cache, from the Web Cache machine. Web Cache will be doing the protocol conversion. The third firewall can be configured to accept only requests from the Apache listeners' addresses; furthermore, these requests would have to be using the AJP protocol, to invoke Java processes in the OC4J instances in the next protected zone, on whatever ports they are monitoring. Yet another firewall could transmit only Oracle Net requests to Oracle database servers, or LDAP requests to OID servers. Both the database and OID servers, which contain sensitive data rather than providing only processing capability, are thus protected by several layers of security.

This four-level firewall is an extreme example. It will be absolutely impossible to bypass all four layers, hacking servers at each stage to fool all the firewall routers into believing that the malicious traffic is legitimate.

The Web Cache is an integral part of the security structure and can indeed replace the first two firewalls. If the Web Cache is installed on a dual-homed device, it can be considered to be a router configured with rules that will allow it to monitor only one or two protocols (HTTP or HTTPS) on one network interface, and to forward them only to a nominated list of addresses (the Apache web listeners) on the other network interface. The Apache web listeners will themselves be configured to reject all traffic that does not come from the Web Cache.

The Oracle Application Server Architecture As a Whole

To conclude the description of Oracle Application Server architecture, consider it as a whole. A web site will typically consist of a farm: one or more middle tier Oracle Application Server instances controlled by a single infrastructure instance.

The infrastructure instance offers support services to the middle tier instances and the applications deployed to them: a central repository of configuration metadata, and the identity management and security services provided by the Oracle Internet Directory, the Single Sign-On service, and the Certificate Authority. It would be unusual for an end user to connect directly to an infrastructure instance.

The middle tier instances run applications for users. These will usually be written in Java and run in OC4J instances. Middle tier instances also provide the PL/SQL gateway and CGI processing, and (depending on the installation type) can also run Portal and Wireless, and Forms, Reports, and Discoverer servers. All middle tier instances have an Apache web listener as their entry point. This routes user requests through to the appropriate application server service by using modules provided by Oracle and dynamically linked to Apache. All the applications deployed to a middle tier instance can make use of Oracle databases.

Web Cache is a transparent front end to one or more middle tier instances. It accelerates delivery of frequently requested content and can also act as a load balancer and be a part of the site's fault tolerance structure.

Figure 2-6 shows the whole picture.

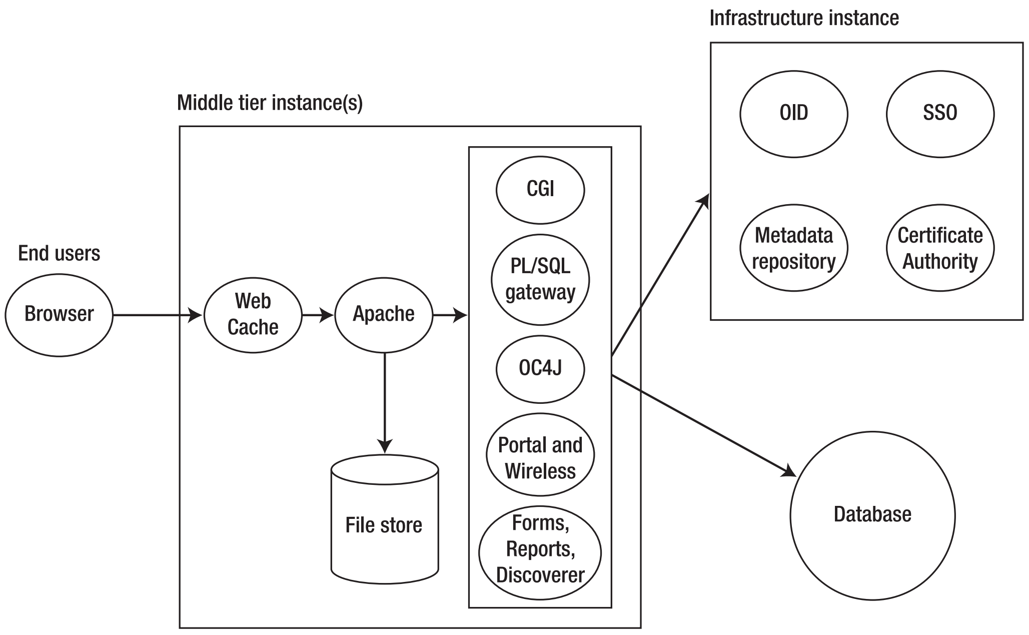

Figure 2-6. *The complete Oracle Application Server topology*

As end users connect to and navigate through applications, they issue URLs or other HTTP requests. These are received by the first layer of the middle tier instance: the Web Cache. The Web Cache returns the document requested if it can, but otherwise will forward the request to the second layer of the middle tier, the Apache web listener. Apache will service the request if it is for a static file, but otherwise it will forward it to the next layer, the appropriate application server service. The service and the applications deployed to it may well make use of the infrastructure instance for security and other purposes and will almost certainly connect to an Oracle database containing the application's data. The document retrieved or generated by the middle tier is returned to the end user's browser via Apache and the Web Cache, completing the HTTP request-response cycle.