ORACLE® White Paper

Developer/2000 and the NCA: An Architectural Overview

July 1997

Developer/2000 and the NCA:
An Architectural Overview

July 1997

Authors: Mark Doran
Rita Morin
Christin Nowakowski
Carl Zetie

Developer/2000 and the NCA: An Architectural Overview

Introduction

The Network Computing Architecture (NCA) provides Developer / 2000 applications with a framework for multi-tier, distributed computing. Through application servers that exploit standards-based Web interfaces and cartridges that plug into the “software bus” provided by Oracle’s Web Application Server, Developer / 2000 applications can be deployed with increased reliability, availability, and scalability (RAS).

This technical white paper explains how Developer / 2000 application servers and cartridges fit into the NCA. The paper explains how current releases of Developer / 2000 exploit the NCA, as well as indicating the direction of future evolution.

The paper addresses each of the Developer / 2000 application servers in turn: Forms, Graphics, and Reports.

Developer/2000 Forms Server

With the introduction of the Forms Server in Release 1.4W, Developer / 2000 introduced support for a distributed three-tier architecture. The Forms Server allows customers to realize the advantages of the Web by moving application logic and processing off of the client and onto a middle layer application server.

The Forms Server consists of two components:

Listener: The Forms Server Listener initiates the Forms runtime session and establishes a connection between the Java client (i.e. Web browser, NC, AppletViewer, etc.) and the Forms Server Runtime Engine.

Runtime Engine: The Forms Server Runtime Engine is a modified version of the Forms Runtime Engine, with user interface functionality redirected to the Java client. It handles all form functionality, except UI interaction, including trigger and commit processing, record management, and general database interaction.

The role of the Forms Server depends on whether or not Developer / 2000 is implemented as an application cartridge. Below, we have described how the Forms Server works in both the cartridge and non-cartridge models. This description applies to both release 4.5 and 5.0 of Forms, as the implementation is the same in both.

Cartridge Model

If you choose to implement Developer/2000 as an application cartridge, you will create a cartridge in the Oracle Web Request Broker and register the cartridge with the Forms Cartridge Handler, a component that interfaces with the Forms Server and the cartridge you have created.

To start and run a Forms application on the Web, end users will use a Java-enabled Web browser to access a URL. The following sequence occurs automatically (Figure 1):

1) The URL corresponds to an application cartridge residing on the application server. The Web Listener accepts this URL request and passes it to the Web Request Broker.

2) The Web Request Broker recognizes the cartridge name in the URL as the name of the Forms cartridge, and initiates the Forms cartridge.

3) The Forms cartridge dynamically constructs an HTML file by merging information from three sources: the cartridge's parameter settings (defined when the cartridge was created), information from the application cartridge HTML file (if any), and parameters and values from the URL. This HTML file includes an applet tag which points to the Forms applet.

4) The newly constructed HTML file is downloaded to the user's browser via the Web Request Broker. The browser reads the applet tag, and requests the associated Java applet from the Web Server.

5) The Java applet sends a request to the Forms Server Listener asking for a particular Forms application (i.e. an FMX).

6) The Listener contacts a Forms Server Runtime Engine. (The Listener maintains a pool of available Runtime Engines to minimize application startup delays). Each active user receives a dedicated Runtime Engine.

7) The Listener establishes a direct socket connection with the Runtime Engine, and sends the socket information to the Java applet. The Java applet then establishes a direct socket connection with the Runtime Engine. The Java applet and the Runtime Engine now communicate directly, freeing the Listener to accept startup requests from other end users. (At this point, neither the Web Server nor the Forms Listener is involved in the communication between the applet and the Runtime Engine.) The Java applet displays the application's user interface in an applet window outside the main window of the end user's Web browser.

8) As in a client-server implementation, the Runtime Engine communicates directly with the database through SQL*Net or ODBC, depending on the datasource.

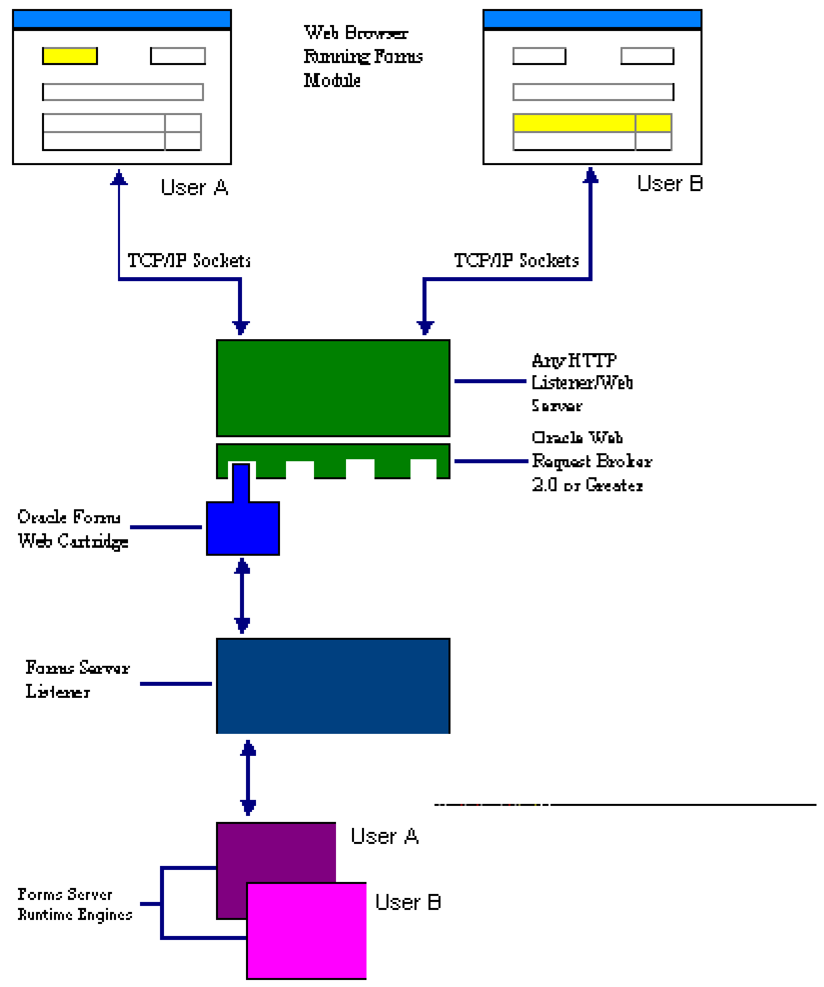

Figure 1: Cartridge Model

In a cartridge implementation, because the initial HTML file is built dynamically by the Forms cartridge, you can create a generic application cartridge containing common parameters, and then reuse it for each of your applications. To use the cartridge with another application, you can simply modify the URL of the application to specify different parameter values, rather than create a new HTML file.

Non-Cartridge Model

If you choose to run your applications on the Web with a non-cartridge implementation, you must have a Web Listener (commonly referred to as the Web Server), but you do not need a Web Request Broker. Therefore, in the non-cartridge model, Oracle's Web Server is not required; any Web Server will suffice.

To start and run a Forms application on the Web, end users will use a Java-enabled Web browser to access a URL as with the cartridge implementation above. However, a slightly different sequence occurs once the URL request is submitted (Figure 2):

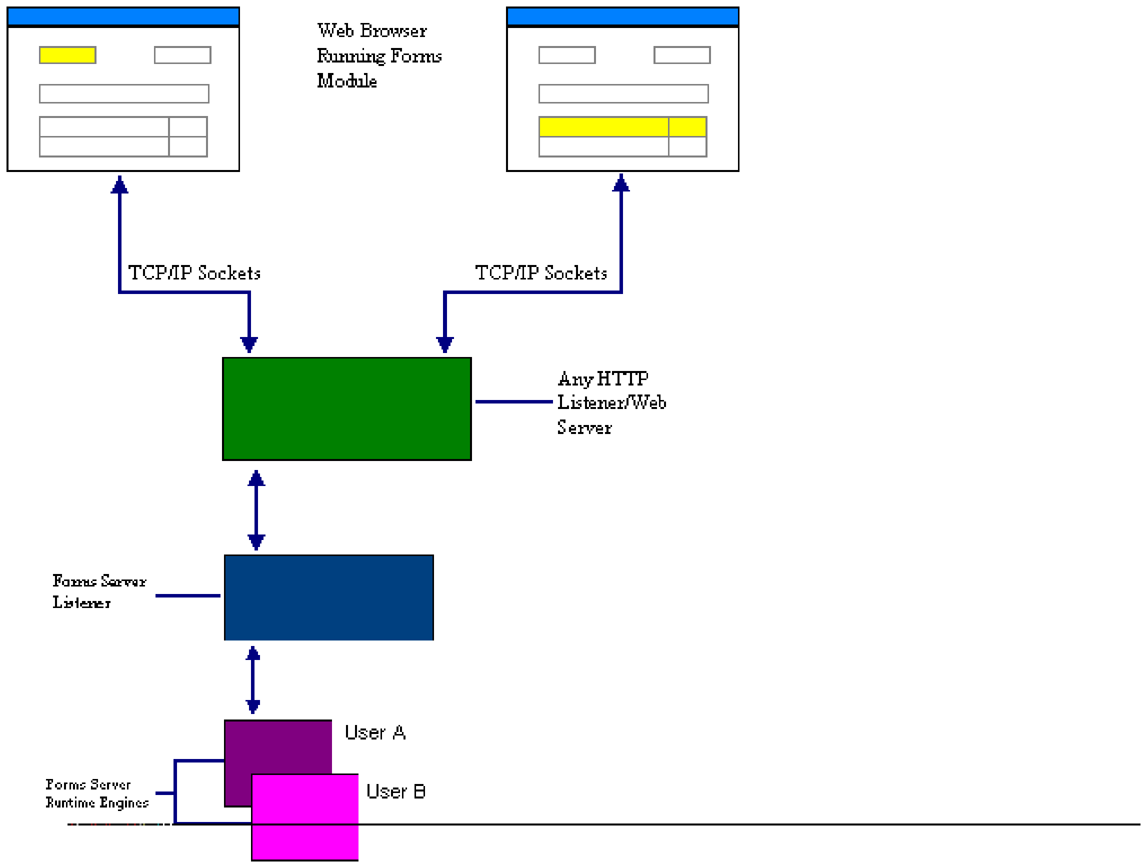

Figure 2: Non-Cartridge Model

1) The URL corresponds to a static HTML file residing on the application server. The Web Listener accepts this URL request.

2) The Web Listener locates the HTML file on the application server and downloads it. The browser reads the applet tag, and requests the associated Java applet from the Web Server.

3) The Java applet sends a request to the Forms Server Listener asking for a particular Forms application (FMX) to be started.

4) The Listener contacts a Forms Server Runtime Engine. As in the cartridge model, each active user has a dedicated Runtime Engine.

5) The Listener establishes a direct socket connection with the Runtime Engine, and sends the socket information to the Java applet. The Java applet then establishes a direct socket connection with the Runtime Engine. The Java applet and the Runtime Engine communicate directly, freeing the Listener to accept startup requests from other end users. The Java applet displays the application's user interface in an applet window outside the main window of the end user's Web browser.

6) As in a client-server implementation, the Runtime Engine communicates directly with the database through SQL*Net (or another driver, for non-Oracle datasources).

In a non-cartridge implementation, you must create a static HTML file for each application. Each static HTML file you create contains hard-coded information specific to the individual application. Even a change in the runtime parameters of an application requires a new HTML file.

Future Releases

In a release after 2.0, the Forms Runtime will be implemented as a cartridge. The Web Application Server will invoke a Forms cartridge directly to run a Forms application, eliminating the need for a separate Listener process and Runtime processes. In this configuration, the Forms cartridge will be able to exploit all of the functionality of the Web Application Server, including Inter-Cartridge Exchange (ICX) and dynamic load balancing.

Initially, each instance of the Cartridge will handle a single connected user, just as the Forms Runtime does today, but in later versions a single cartridge will be able to serve multiple simultaneous users for even greater scalability.

Graphics Cartridge

In this section we will discuss the architecture and details of the Graphics Cartridge and how this cartridge is configured with both the Web Server 2.1 and 3.0. This applies to both Developer/2000 release 1.5 and release 2.0, since the details and implementation will not change with these releases.

Graphics Cartridge and Web Server 2.1

The Oracle Graphics Cartridge enables users to run existing Graphics displays from their Web browser, by connecting to any Web Server that supports the Oracle Web Request Broker (WRB). The WRB provides the "glue" between the Web Server and the cartridge and manages the client connections between them. Below is a diagram of the architecture of the Graphics cartridge (Figure 3).

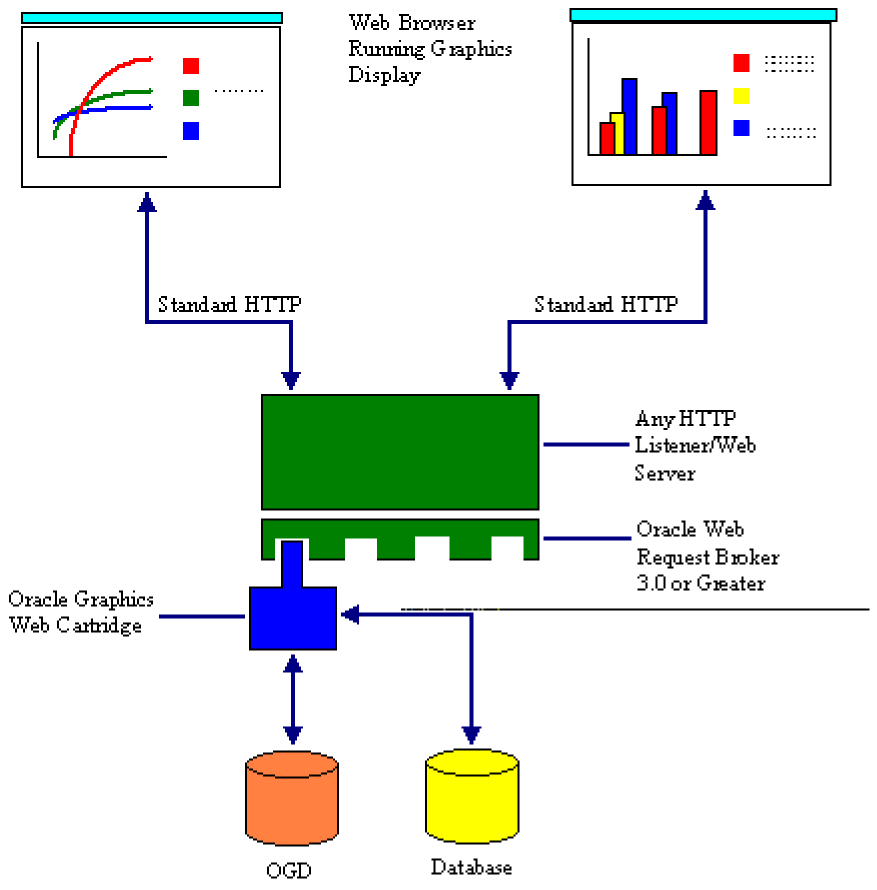
Web Browser Running Graphics Display
Standard HTTP
Standard HTTP
Any HTTP Listener/Web Server
Oracle Web Request Broker 3.0 or Greater
Oracle Graphics Web Cartridge
OGD
Database

Figure 3 : Graphics Server architecture

To begin, the user must enter the URL of the requested HTML page they wish to see. Included in this URL are arguments that specify which Graphics cartridge should be invoked. Also, any other parameters that need to be passed to the cartridge can be included in the URL. The browser sends this URL to the Web Server where it is parsed. The request is handed to the WRB, who communicates with the correct cartridge. Once the Graphics cartridge has received the parameters from the WRB, it then makes its connection to the database and generates the appropriate output. This output is generated as an HTML page that contains a GIF-stream image. The image is returned to the browser, where it is displayed.

With Web Server 2.1, interactive graphics such as drill down are not supported.

Graphics and Web Application Server 3.0

With the introduction of Oracle Web Application Server 3.0 (WAS), a “farm” of Graphics cartridges can be maintained. A number of cartridges can be pre-started so that incoming requests can be handled quickly. Also, with the load balancing features of WAS, new cartridges can be started as needed to handle the request load, up to a predefined limit set by the cartridge administrator. In addition, idle cartridges may be automatically terminated to free resources. This architecture allows for many more connections and satisfies the users' needs of quick display generation.

With WAS 3.0, the HTML page has a special construct in it that allows it to detect when the user performs a mouse click in the display region of the page. Also included in the HTML page is the session ID of the cartridge connection. This ID is used for subsequent requests to the cartridge, such as a mouse click, so that the correct state information is used. If the user does click on the display, a request is sent back to the cartridge via the Web Server with the coordinates of the mouse click event. If an action is required, the Graphics cartridge then generates a new GIF-stream which is then returned to the browser. This type of architecture allows fully interactive Graphics displays on the web.

Developer/2000 Multi-tier Report Server

In this section we will discuss the capabilities of the Multi-tier Report Server and how those capabilities enable you to run reports on a remote application server. Note that the Reports Server differs significantly in this respect from the Forms and Graphics Servers. When used in conjunction with the Reports Web CGI or Web Cartridge, the Reports Server enables you to run dynamic reports from a Web Browser using standard URL syntax. Unlike the Forms and Graphics Servers, however, the Reports Server can also be used in a non-Web environment, for example to deliver highly scalable server-based reporting in a client / server environment.

Developer/2000 Release 1.4

Although not strictly related to the Report Server, it is useful to briefly recap how Reports delivered Web (HTML) output in earlier releases. Release 1.4 introduced the ability to define at runtime the parameters and criteria of a given report through a URL. By the implementation of the reports runtime engine as a CGI (common gateway interface) executable, a URL mimics the standard command line as used in a client / server two-tier environment, for example, :

http://rptserv.us.oracle.com/r25cgi32?REPORT=QTR.RDF+ USERID=scott/tiger+DESFORMAT=HTML+EMPNO=10

When a URL of this nature is received, the web server spawns a runtime engine and the specified report is executed. The output is then returned in the desired format to the calling browser (Figure 4). This simple architecture allowed reports to be run on the server rather than the client, and the output to be delivered in a browser in HTML format.

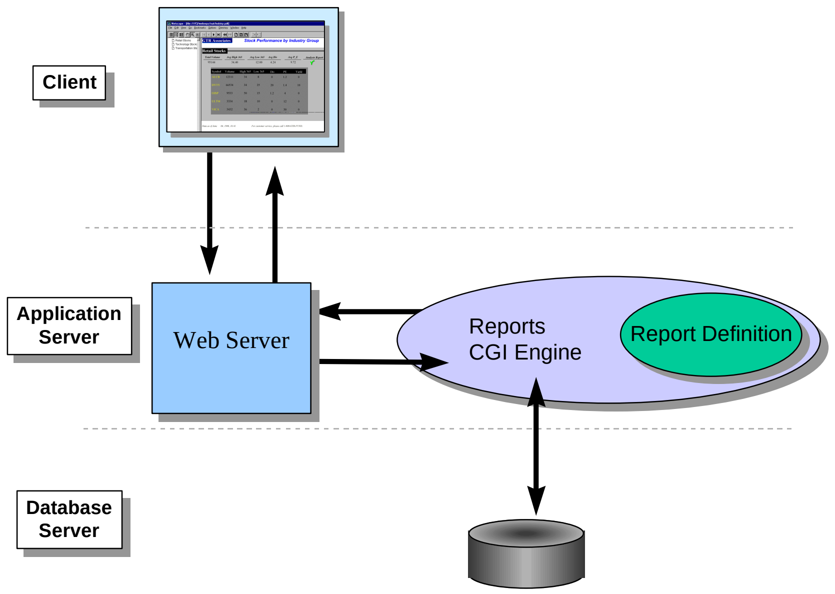

Figure 4: R1.4 Dynamic Web Reporting with Reports CGI Interface

Note that this is not a true Server implementation, and has none of the scalability characteristics of the Server described below. It is simply a way of executing the report on the server machine and delivering the output in HTML.

Developer/2000 R1.5 & R2.0

Release 1.5 introduces a true multi-tier architecture for executing and distributing reports. The centerpiece of this new architecture is the Developer/2000 Multi-tier Report Server. With the Report Server, reports may be executed remotely on much more powerful UNIX or NT server platforms where resources are greater, and at the same time significantly reduce the load on the client machine.

The Multi-tier Report Server is a multithreaded executable which generates report output dynamically based on requests received from “clients”. In this context a client is not necessarily an end user’s PC; requests may be issued from end user PCs, from an interface with any web server on the middle tier, or from third-party applications via the Reports ActiveX Control. The Reports Server handles client requests by entering all report submissions into a job queue. As one of the Server’s Runtime Engines becomes available, the next job in the queue is dispatched to that Engine and executed (Figure 5).

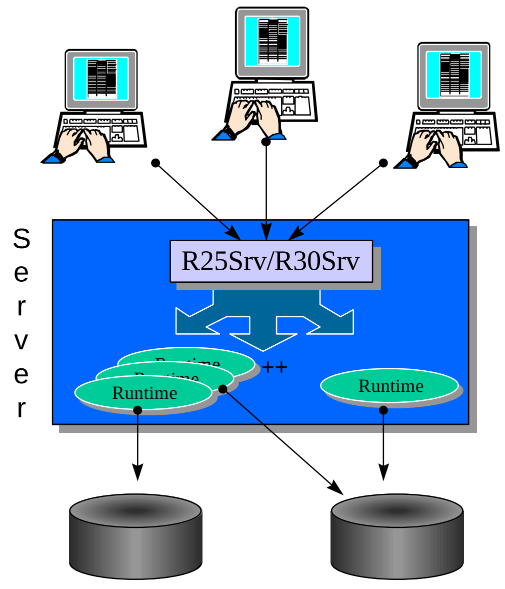

Figure 5 : Report Server Architecture (Release 1.5/2.0)

Submitting Requests to the Server

Client requests for reports will be prioritized on a first-come, first served basis and my be submitted to the Report Server via the following methods:

Lightweight Client [R25CLI/R30CLI] -A lightweight executable, known as the Command Line Interface (CLI) is all that is needed on a client machine to submit a report to the Report Server; for example, in Release 2.0:

r30cli REPORT=rptname.rdf MODE=character DESTYPE=file DESNAME=outputfile.lis SERVER=rptserv

Notice the new command line argument **SERVER=<name>**. The purpose of this is to specify to which Report Server process the report should be submitted. The remote procedure call (RPC) mechanism, which is used to communicate between client and application server, uses SQL*Net 2.3 as it's transport layer and hence is able to take advantage of all the Domain Name Services and Fail Over benefits which are inherent within it. The naming of a Report Server is managed in the same manner as a SQL*NET V2 alias to a database, using the TNSNAME.ORA file.

Web Server CGI Interface [R25CGI/R30CGI] -A Web Server -independent interface to the Report Server that takes a URL as input and communicates to the Report Server, allowing users of any web server to generate and receive dynamic reports from their web browser (Figure 6).

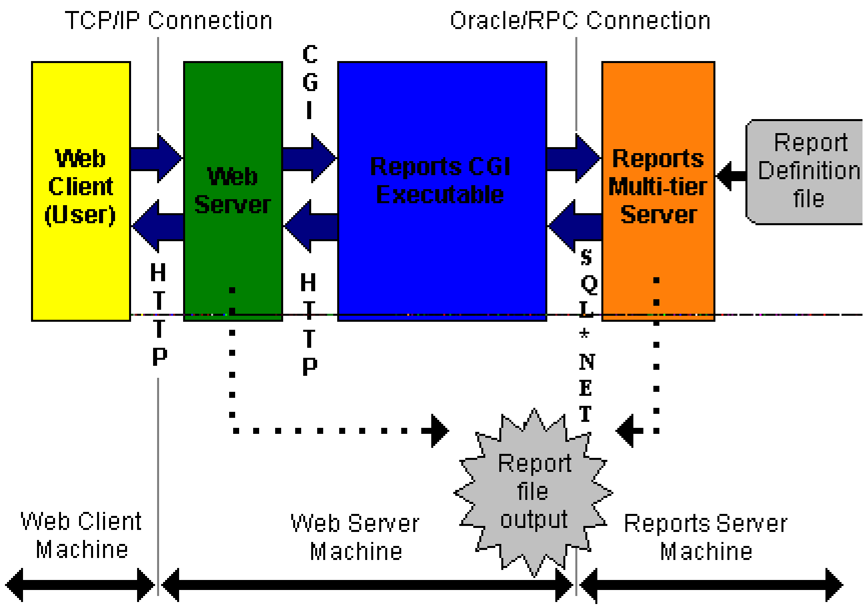

Figure 6 : Reports CGI Architecture (Release 1.5/2.0)

When the client submits a URL, the Web Server runs the Reports Web CGI. The Web CGI then does the following (for example, in Release 1.5):

http://rptserv.us.oracle.com /your_virtual_cgi_dir/r25cgi32?arg1+arg2+argsN

1) Parses the client request.

2) Converts the request to a Reports Multi-tier Server command line.

3) Submits the command line to the Reports Multi-tier Server (synchronously).

4) After the report is finished, retrieves the name of the report output from the server and creates an HTTP redirection ("Location:") to the output file. After that, it is up to the Web Server to manage this redirection (typically by displaying the file back in the Web client browser).

5) Terminates.

Web Application Server Cartridge Interface- Plug-in cartridge for Oracle’s Web Application Server that has a similar syntax to the CGI based interface (Figure 7).

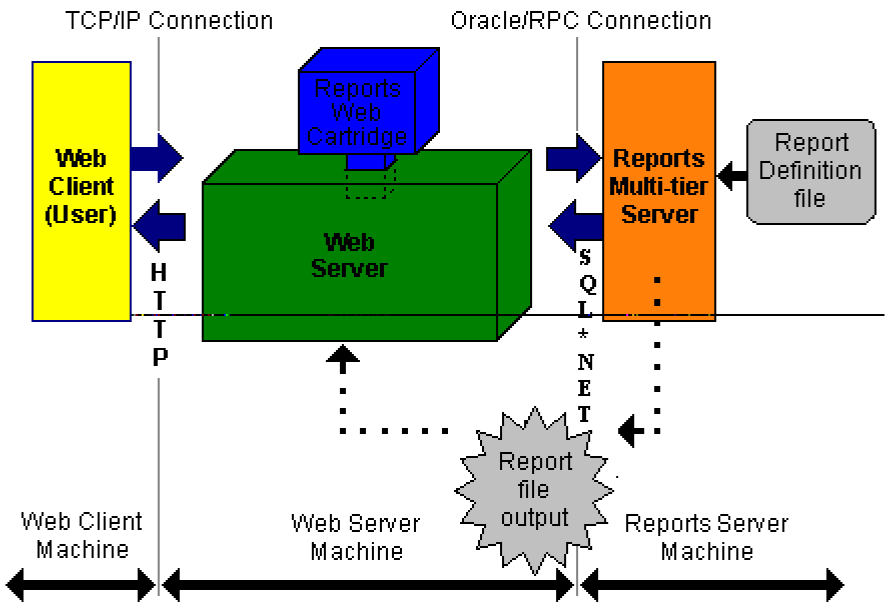

Figure 7 : Reports Web Cartridge Architecture (Release 1.5/2.0)

After URL submission from the client, the Web server invokes the R25OWS Web Cartridge. The Web Cartridge then does the following: (for example, Release 1.5):

http://rptserv.us.oracle.com/report_cartridge?arg1+arg2+argsN

1) Parses the client request.

2) Converts it to a Reports Multi-tier Server command line.

3) Submits the command line to the Reports Multi-tier Server (synchronously).

4) After the report is finished, retrieves the name of the report output from the server and creates HTTP redirection ("Location:") to the output file. After that, it is up to the Web Server to manage this redirection (typically by displaying the file back in the Web client browser).

5) Terminates.

Reports ActiveX Control -Submits and displays reports from any third party application development tool that supports the use of ActiveX (OCX) controls.

Direct Developer/2000 Forms Interface -Submits reports directly from the Developer/2000 Form Builder via RUN_PRODUCT or RUN_REPORT.

Cached Report Output

The Developer/2000 Multi-Tier Report Server enables the use of an output cache for generated reports. If a given report has had multiple submission requests with the same parameters and is within a defined time period, the Report Server will determine that they are duplicates and deliver to the user the output from the cache instead of executing the report multiple times.

Load Balancing

Up to this point, we have only considered the case where a single server supports all connected users. For a truly scalable solution supporting larger numbers of users, the system needs to be able to distribute user requests dynamically across several servers; in other words, to perform load balancing. In this section we will discuss the capabilities of the Web Server (Web Application Server) and how those capabilities can be used by the application servers to provide load balancing.

Web Server 2.0 & 2.1

In Web Server 2.0 & 2.1, the Web Server can be pictured as consisting of a listener, a dispatcher, and a number of cartridges communicating through a standard interface layer called the WRBX. All of these components must run on a single machine (Figure 8). Note that this diagram has been simplified slightly; for a complete description, refer to the Web Server documentation.

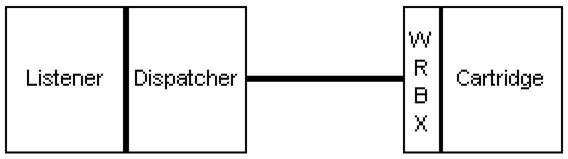

Figure 8 : Web Server 2.x architecture

The *listener*, commonly called the Web Server, is responsible for accepting an incoming URL and returning the resulting HTML. For a simple static HTML page, no other component is involved: the listener finds the HTML page in the virtual file system and returns it to the browser. If the URL does not refer to an HTML page, the listener passes it to the *dispatcher*. This component determines which *cartridge* should handle the request, and passes the request to that cartridge. The *WRBX* is the standard interface that provides communication between the dispatcher and the cartridge. The dispatcher and WRBX together constitute the Web Request Broker (WRB).

In this context, we can think of a Web Server cartridge as a software component that knows how to run a certain kind of application. The Java cartridge runs Java programs, the PL/SQL cartridge runs PL/SQL programs, and the Forms cartridge runs Forms applications.

In this model, each instance of a cartridge processes only one request at a time, and the Web Server assumes that the cartridge is busy until it exits, having completed the request. If the requested cartridge is busy, the Web Server returns an error to the browser. It is also possible to have multiple instances of a particular cartridge; for example, you might have multiple Graphics cartridges. Each instance can handle a single request.

Release 2.0/2.1 of Web Server has no load balancing capabilities. When a user requests a URL from a particular listener, the request is always serviced on that listener; there is no capability for the Web Server to pass the request to a different machine.

Web Application Server 3.0 & 3.1

In the 3.x releases, the picture becomes more sophisticated. The connection between the listener/dispatcher and the cartridge is no longer a "hard link" on a single machine. Instead, the connection is mediated by a set of intermediary services that allow a single URL to refer to many cartridges distributed across many machines. This architecture brings with it the capability to perform load balancing by "intelligently" distributing incoming requests. The listener/dispatcher, services, and cartridges can all be placed on separate machines, or even duplicated across multiple machines (Figure 9).

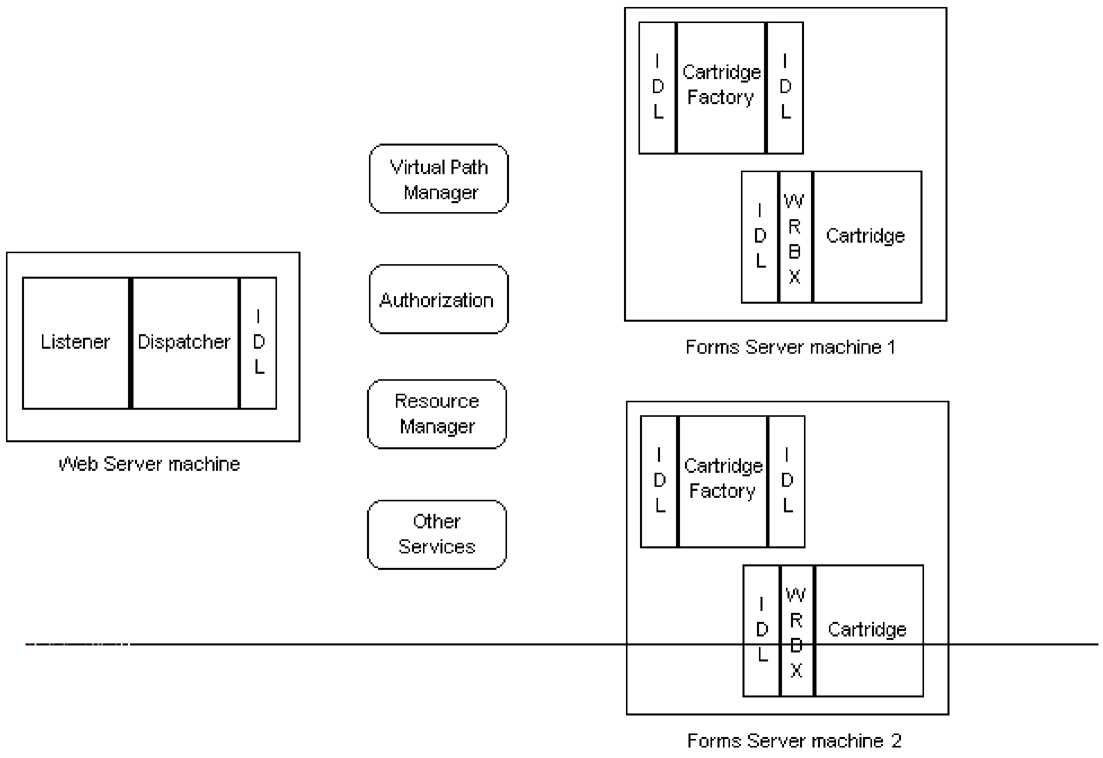

Figure 9 : Web Application Server 3.x architecture

In this version, the various components of the Web Application Server are distributed across multiple machines and communicate through CORBA services (represented here by the Interface Definition Language, or IDL, layers on each component.) The services, such as the Virtual Path Manager and the Resource Manager, can potentially be distributed across multiple machines, but normally they will be on a single machine. Although only Forms Server machines are shown in this diagram, there could potentially be many machines capable of running the Forms, Reports, and Graphics Servers, and each machine could have more than one type of cartridge.

When a URL arrives at the listener from a browser, the listener passes it to the dispatcher just as in the case for the 2.x Web Server. The dispatcher dynamically resolves the URL to a particular server machine through the intermediary services including the Authorization Service, the Resource Manager, and the Virtual Path Manager. If more than one machine is capable of handling the request, the Resource Manager can choose the least loaded one. The URL is handed to the *cartridge factory* on the chosen machine, which starts or activates the cartridge to handle the request. This dynamic allocation of requests across machines allows the Web Application Server to balance the load across all available servers.

In release 3.0 the load balancing that the Resource Manager performs is a “card deal” or “round robin”. For each cartridge type, the Resource Manager is told how many instances of that cartridge can be run on each machine. The Resource Manager hands requests to each machine, in turn, as long the machine has available cartridges to run them. If a machine becomes so heavily loaded that all its cartridges of a given type are busy, the Resource Manager will skip that machine until one or more of the cartridges completes its current request.

Note that this load balancing takes no account of how heavily loaded each machine actually is, nor of how powerful each machine is. It depends solely on how many instances of a given cartridge are running, and makes no distinction as how many instances a server is capable of running. Release 3.1 will add the ability to specify the “Quality of Service” of each machine, which will allow the administrator to specify such factors as a weighting factor per machines, the number of cartridge instances per machine, etc., to more effectively balance the workload.

Since the Reports Server and Forms Server currently perform most of their processing outside of their respective cartridges, they will very rarely be busy enough to be affected by the limit on the number of cartridge instances. However, this dynamic mechanism will allow requests to a single URL to be automatically distributed across a number of machines, which increases the scalability of the architecture. In the future, when the Reports and Forms servers perform the processing in their respective cartridges, they will automatically inherit this load balancing feature.

Forms and Load Balancing

Since the Forms Server currently performs most of its processing outside of the cartridges, a Forms Server machine will very rarely be busy enough to be affected by the limit on the number of cartridge instances. However, this dynamic mechanism will allow requests to a single URL to be automatically distributed across a number of machines, which increases the scalability of the architecture. In the future, when the Forms Servers perform the processing within the cartridges, they will automatically inherit this load balancing feature.

As the above explanation has shown, it will not be possible to achieve full load balancing using the Web Application Server for Forms Servers with the flexibility that the most demanding applications require until Web Application Server 3.1 is available, and until the Forms Server is implemented entirely as a cartridge. In order to provide an intermediate solution, Developer/2000 will be providing a Forms-specific mechanism for load balancing. This solution will be available with release 1.6 of Developer/2000 and will work with Web Server 2.1 as well as Web Application Server 3.0 and 3.1.

This solution consists of two pieces: a Master Metric Server (MMS) and a Slave Metric Server (SMS). The SMS is a very simple process that periodically, every few seconds, sends a network packet to the MMS reporting the number of processes running on its host, as a measure of how loaded that host is. The MMS is a process that keeps track of these messages to determine how busy each host is. The MMS runs on the machine that hosts the Web Request Broker or Web Application Server and the Forms cartridge. A SMS runs on each machine where a Forms Server is available; a Web Server (not necessarily Oracle Web Application Server) must also be running on that machine.

When a URL requesting a Forms application is sent to the Web Server on the MMS host machine, the dispatcher hands it to the Forms cartridge as normal. The Forms cartridge communicates with the MMS to determine the least busy Forms Server host. The cartridge then creates dynamic HTML to launch the Forms applet, in the same way as in R1.4W, with one important exception. In R1.4W, the dynamic HTML always directed the browser to load the applet from the machine originating the HTML and connect back to a Forms Server on that same machine. With R1.6, the dynamic HTML can direct the browser to load the applet from any machine with a Forms Server. In this way, the request can be directed to the least loaded machine, as determined by the SMS and MMS processes.

Note that the master machine must be running Oracle Web Server or Oracle Web Application Server, because that is where the cartridge is installed, but the slave machine can be running any Web Server that can service the dynamically generated HTML request.

Graphics and Load Balancing

Graphics already processes requests within the cartridge, so it will immediately benefit from Web Application Server load balancing.

Reports and Load Balancing

The Reports Server performs sophisticated load balancing independently of the Web Application Server. Requests issued from clients will be distributed to a dynamic and configurable number of Report Server runtime engines, hence multiple reports may be executed concurrently from a given Developer / 2000 Reports Server.

As the number of jobs in the queue increases, the Report Server will dynamically invoke additional engines to execute queued jobs as necessary up to a maximum limit set in a Report Server configuration file. During idle periods in which there are more server engines than job requests in the Report Server queue, idle reports runtime engines can be shutdown automatically to conserve machine resources. The maximum idle period along with several other parameters may be defined in the Report Server configuration file, **servername.ora**.

Summary

Oracle's Networking Computer Architecture provides a robust, scalable framework for enterprise solutions. Developer / 2000 combines this deployment architecture with proven, market-leading development tools to deliver solutions for the largest, most demanding environments.

Developer / 2000 already provides the market's most complete, robust, and scalable environment for multi-tier solutions. Forthcoming releases will extend Developer / 2000's lead even further.