Pure Java Implementation of a Scalable Web Server with a Integrated Servlet Container

Qing Jiang Lee

A Thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

Friday, April 20, 2001

0-612-59332-0

Abstract

Pure Java Implementation of a
Scalable Web Server with a
Integrated Servlet Container
Qing Jiang Lee

As the Internet continues to grow, e-commerce has become a new way of doing business within almost every industry. Today, some popular Web sites are handling thousands - and even tens of thousands - of Web page requests per second. Web server technology is central to the current Internet client/server model, since the performance of the server directly affects the performance of the Web site using it.

In this thesis, I present a survey of contemporary commercial Web servers, describe the general concepts of Web server and Servlet container, then describe my own design and implementation of a purely Java-based scalable Web server that is integrated with a Servlet container. The purpose of this Web server implementation is to provide a commercially viable, purely Java-based scalable Web server that is fully HTTP 1.1-compliant and integrated with a Servlet container that is fully compliant with the Java Servlet Development Kit (JSDK) 2.1.

This Web server, the functionally equivalent to the Apache Web server, supports the latest HTTP protocol (version 1.1) and incorporates a built-in Servlet container that is the equivalent of a JServ or Tomcat Servlet container. The built-in Servlet container supports CGI scripts by using a CGI handler Servlet. The scalability of this Web server implementation considers two ways of scaling: Transparent and Redirect. In both cases, it acts as a master and can use any Web server or itself as a slave.

Acknowledgements

I would like to express my cordial gratitude to Dr. Lixin Tao. Dr. Tao has given me great support in my studies and maximum scheduling flexibility. Most of my consultations and discussions with Dr. Tao occurred after working hours and on weekends. This thesis would not have been completed without Dr. Tao's compassionate support, advice and encouragement.

My thanks also go to the Comp 628 team: Dr. Lixin Tao, the Instructor; Li An, Aimin Han, Man Bao, Cheng Xing, Daofeng Zhang, and Liusong Yang. They provided a good basis for the Graphic User Interface design work used in the project Implementation.

I am grateful to the professors and administrative staff in the department of Computer Science, especially Dr. Ahmed Seffah, who gave lectures in interface design, and Ms. Halina Monkiewicz, whose friendliness and administrative support have made my academic life easier and more rewarding.

I would also like to thank my parents for providing the emotional and financial support I required.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Most applications running on the Internet are client/server applications. A server application is a system program that uses a pre-defined protocol to serve a number of clients and to complete certain tasks. Some commonly used server types on the Internet are the HTTP server, the FTP server, the mail server and the news server. Each kind of server uses a different protocol. For example, HTTP servers use the HTTP protocol, and FTP servers use the File Transfer Protocol (FTP) protocol. The current World Wide Web is based on a protocol that uses the HTTP server as the server type and a Web browser as the "client" software.

1.1 HTTP Server

HTTP servers use the Hypertext Transfer Protocol (HTTP), which is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless protocol that can be used for many tasks other than hypertext, such as name servers and distributed-object management systems and, through extension of its request methods, error codes and headers [1].

The HTTP server, also called Web server, is the server that provides Web service. Any Web page we see through a Web browser is fetched from an HTTP server by specifying its URL (Uniform Resource Locator). HTTP protocol is designed to be the communication protocol between the servers and the clients (the browsers). The Apache Web server and the Microsoft Internet Information Server (IIS) are two typical and widely used Web servers. Apache is mainly used on the Unix/Linux platform; Microsoft IIS is mainly used on the Window NT/9X platform. We discuss details of these Web servers in Chapter 2 (*Literature Survey).*

HTTP servers are designed to serve many clients. Browser applications, which run on the client side, use the HTTP protocol to communicate with these servers. A browser first sends a Web page request to the server through the network. Then the server fetches the Web page from the local disk or calls a Servlet to construct a dynamic page. Finally, the Web page is sent back to the browser.

1.2 Application Server

The requirements for Web site development have grown considerably more complex over the past few years. Web sites started out as Internet destinations that offered only static content, mainly words and images, and Web site development meant no more than creating HTML files and simple CGI scripts. Today, however, the World Wide Web has become the preferred user interface to handle a host of new application types. As a result, Web developers have to deal with many of the same requirements and problems associated with traditional application development.

Many of the problems that current Web site developers face have nothing to do with deployment rather than development. The challenge of building Web sites with qualities of reliability, scalability, stability, and manageability is just now being addressed. As Web sites begin to handle more business-critical applications, the systems management and operational issues associated with Web development is becoming crucial.

Dynamic contents on a Web page had proved attractive for users of the Internet, but the old Web server approach is not sufficient to support dynamic contents. For example, chat rooms and online BBS need to be supported by dynamically updated Web pages. All valid user inputs need to be shown on the Web page after the users submit their inputs. Some real-time content such as stock prices and real-time news updates are presently popular on the World Wide Web. Since this content is presented in real time, it is not possible to create a static page every time the stock price changes or news breaks. Some techniques are necessary to create this dynamic, real time content.

At the same time, today's business applications deeply rely on the support of databases. A business Web site usually needs to support large amount of data that can be organized in response to a database query. For example, a Web site providing stock quotes must be supported by large amount of historical stock data that is stored in a database, and can be queried to construct diagram of stock histories for research purposes. The old Web server model, which can only support static HTML pages, is not enough to support such dynamic SQL queries on a Web page.

Moreover, because of the stateless nature of HTTP, static pages alone cannot meet the demands of a "session". Today's e-business solutions involve a highly flexible exploitation of a session. For example, while shopping an online e-commerce Web site, a user may need to create a shopping cart and add an item to that cart, then continue to browse other pages on the same site. Information about the previous additions to the cart needs to be kept until the user logs out. Using a traditional static Web server alone cannot achieve this.

A new model for Web development has evolved to address these development and deployment issues. New types of Web application, such as Netscape Application Server (formerly Kiva Server) 2.1 and NetDynamics 4.0, have come to market. They are called "application servers" because they form a clear level of separation between the Web server and data access layers. As Figure 1 shows, Web sites built using the application server model consist of at least three back-end layers: the Web server layer, the application server layer, and the data layer. In this model, most or all application logic exists in the middle tier, where the application servers handle all data manipulation and HTML page-creation functions.

Application servers, the next logical step in the development of commercial Web sites, developed from the need to have mission critical applications consistently available to an ever-growing number of clients. These applications needed to be secure and reliable so that regardless of the number of people accessing the system or the source of data, the application server would always be up and running. Prior to the introduction of application servers, Web applications often ran on Web servers that were only designed to serve Web pages. Running and developing applications was slow and complex.

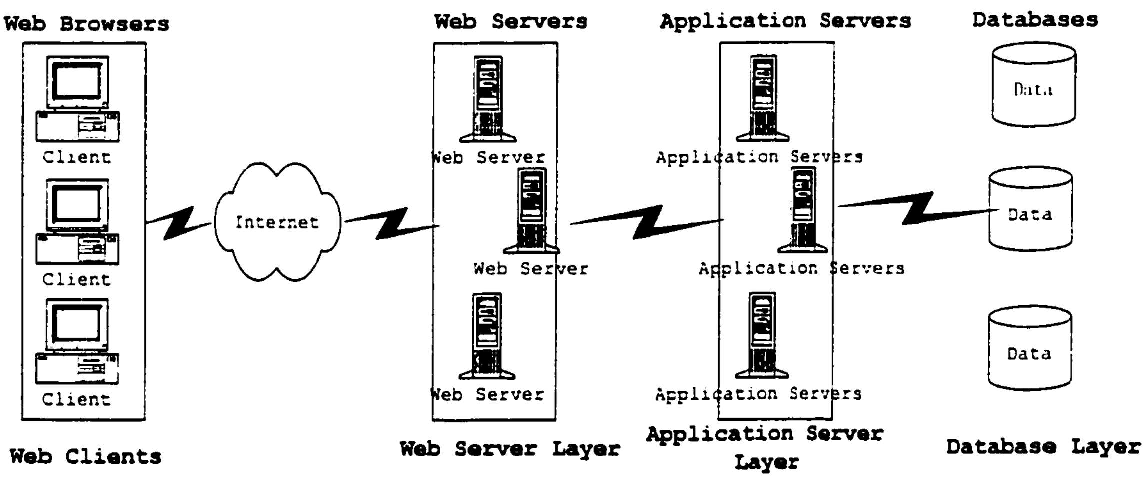

Figure 1 The Application Server Model

Application servers are part of a multi-tiered architecture in which there is a physical separation between the client that requests information, the programs that process the request and the data that is operated upon. The multi-tiered architecture evolved from the mainframe where client, data and process were centralized in the one place. GUI (Graphical User Interface) interfaces were rare and remote multiple database access was difficult. The mid 1980's saw the introduction of client/server computing model, which divided processing between a client (a PC) and a server (a mainframe computer). A relational database system on the mainframe usually handled requests in queries and the PC applied the presentation and business logic after receiving processed data from the mainframe. This system allowed for modular development and a GUI but deployment proved problematic.

Three-tier computing then separated the presentation logic from the business logic [8]. This separation meant that the business code was independent of how and where it was presented. The business logic layer, now in the middle tier, was not concerned with what type of client displayed the data. Three-tier computing was more portable, worked across different types of platforms and allowed for the balancing of requests from clients across multiple servers. Security was easier to implement since the application software was no longer on the client side and costs were substantially reduced. However, providing the underlying functions of the middle layer – functions such as transaction processing, security and accessing of the data layer – was still complex. The emergence of development tools and a runtime environment to solve this problem came together in the Application Server.

Application Server is the middleware between enterprise data sources and the clients that access those data sources. Business code is stored and processed on the Application Server rather than on the clients. An application is deployed and managed in a single location, and the application is accessible to large numbers of heterogeneous clients.

Application Server applications run in a distributed, multi-tiered environment. This means that an enterprise system might consist of several application servers —

computers running the application server software — along with multiple database servers and Web servers. Application code can be distributed among the application servers. Overall, the machines and software involved are divided into three tiers, as shown in Figure 2:

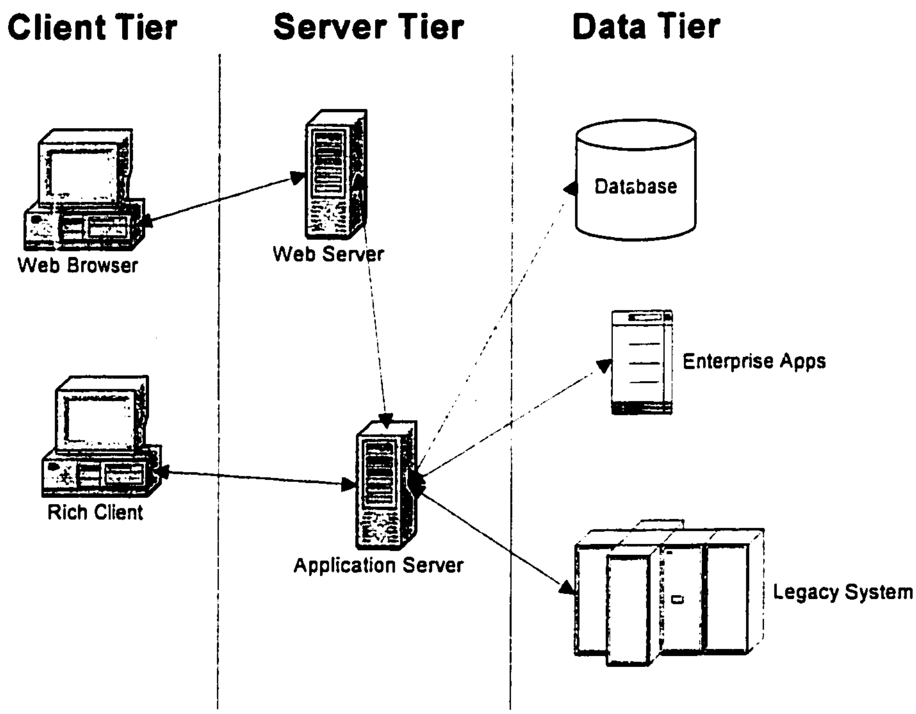

Figure 2 Three Tier Structure of an Application Server

- A client tier; the user interface. Requests for data originate here, represented by Web browsers or thin-clients (such as a Java application).

- A server tier represented by a Web server such as Apache Web Server and an application server, such as Netscape Application Server, that runs the business code.

- A data tier, represented by databases or other back-end data sources.

1.3 Servlet and Servlet Container

Abao Web Server (AWS) is a web server implementation that extends its capability by selects Java Servlets as the application server layer. We will cover our reasons for selecting Java Servlets in Subsection 2.2.2. The following chapters of this thesis are in two parts, those that deal with the HTTP server and those that deal with the Servlet container.

Servlets are small Java programs that run on the server side of a Web server/Web client connection. The Servlets employed here use the standard HTTP communications that ordinary static Web pages use to communicate with the client's Web browser. From the point of view of the browser, Web pages produced by Servlets look no different from ordinary static HTML pages.

Java Servlets are small, platform-independent Java programs that can be used to extend the functionality of a Web server in a variety of ways. Java applets are small programs compiled to binary code that can be loaded dynamically to extend the capabilities of the browser. Servlets are to the server what applets are to the client.

Servlets differ from applets in that Servlets do not run in a Web browser or with a graphical user interface. Instead, Servlets interact through requests and responses with the Servlet container running on the Web server. This request-response paradigm is modeled on the behavior of the Hypertext Transfer Protocol (HTTP).

A client program, which could be a Web browser or some other program that can make connections across the Internet, accesses a Web server and makes a request. Once the Web server gets the request, it parses the request to determine if it is a Servlet request or not. Usually this is done by comparing the request URL to some pre-registered pattern. If it is a Servlet request, the Web server calls a Servlet container to handle the incoming request. The Servlet container is a server-side program that manages all Servlets. It selects a Servlet that handles the request. The Servlet then processes the request and returns the response to the client.

From the point of view of the client, the dynamic HTML generated by a Servlet is indistinguishable from static HTML pages. In functionality, Servlets lie somewhere between Common Gateway Interface (CGI) programs and proprietary server extensions such as the Netscape Server Application Programming Interface (NSAPI). Unlike CGI programs and NSAPI modules, one does not need to modify Servlets to make them conform to either a platform or a server.

A Servlet container is a server side program that needs to work together with a Web server to provide Servlet API. There are many Servlet APIs, however, Sun's Servlet API — the JAVA Servlet Development Kit (JSDK) — is the most popular Servlet API on the market. The AWS project implements the JSDK 2.1 standard, the latest Servlet API when the project began. However, at the time of my writing of this thesis, Sun has deployed the new JSDK 2.2 standard, which is slightly different from version 2.1.

The application server is designed to make it easier for developers to isolate the business logic in their projects — usually through components — and develop three-tier applications. Many application servers also offer additional features such as transaction management, clustering and fail-over, and load balancing.

The AWS project (see Section 5.1) integrates an HTTP server and a Servlet container to make an application server. The goal of the AWS project is to provide application server solutions based on the Java platform and Java Servlet technology. It is part of the first stage of my ASP OS project as described in Section 10.2 of this thesis.

A number of companies, including Microsoft, Netscape, Sun, and IBM, make application servers, but user's choices may be limited by both the platform on which the application server is to run, and the types of components the application server supports.

Most application servers support code written to their API using C++, Java, or both. However, if the developer's code is encapsulated in a standard component (typically CORBA or Microsoft DNA as discussed in Subsection 2.2.2), then, in many cases, this approach will work too. Usually migrate an existing application to applications running on an application server depends on how the code is packaged and which application server is chosen.

Mission-critical software aimed at the enterprise is generally more expensive than software designed for consumers. For this reason, the pricing of application servers varies widely, with those offering clustering or fail-over capabilities costing more. Microsoft's Transaction Server comes free with Windows NT Server, and other lower-end application servers are also not costly.

1.4 Challenge, Objective and Contribution

Application hosting servers need to support tens of thousands of concurrent service sessions with high availability and low response time. Unlike Web servers that are mainly used to support stateless HTTP connections requesting Web contents, ASP application servers need to support connection sessions during which some state information (usually session data) must be kept on the servers. Such sessions may last hours or days, and servers cannot predict the connection patterns.

For Web servers, the main techniques currently used to improve server performance include Redundant Array of Independent Disks (RAID); server farms based on dozens of processors interconnected by buses or shared memories; and extensive caching. For example, Yahoo uses an array of around 50 high-performance server processors, and Windows 2000 can support a limited number of server processors. Non-preemptive scheduling algorithms are used to balance the workload among the processors. Some application service providers also support external caching as a generic approach to boosting Web server performance. Today's Web server market is mainly based on proprietary *ad hoc* techniques, which cannot support the level of service quality needed by ASP.

A server is designed to serve many clients at the same time and produce fast response time and large amount of throughput. With the revolution of the Internet and World Wide Web, commercial Web sites like Yahoo! now serve millions of Web requests per day, thousands of requests per second. Web servers can easily become bottlenecks in network performance, and for this reason performance is probably the most important factor to be considered when choosing a Web server application.

Web servers and application servers play the central role in all Internet applications. For future studies, we need to know how Web servers and application servers work. However, due to copyright issues and a competitive business environment, the codes employed by many commercial Web servers are not open source. Documentations and books about these servers are mainly based on their installation, configuration and user manual. Therefore, in order to study the architecture of Web servers and application servers, I decided to base my thesis on the survey and implementation of an application server. I designed and implemented the AWS, which integrated a small Java Web server and a Java Servlet container. Based on the AWS project, I designed the four-stage architecture of the ASPOS project, which is a complete solution for Application Service Providers. I have had discussions with some venture capital sources, and I am looking forward to undertaking this project.

AWS is implemented using pure Java; and can therefore be compiled and run on any platform that supports Java. It supports the latest HTTP standard (HTTP 1.1), which introduces the "persistent connection" whereby one connection is used to fulfill many HTTP requests. A client and a server can use persistent connection to pipeline requests and responses. AWS also uses the newest Servlet API from Sun, JSDK 2.1.

AWS is designed and tested to handle hundreds or even thousands of connections per second. It uses multi-thread instead of multi-process to achieve that goal. Multi-thread is more efficient that multi-process since thread consumes less system resources than process. Many other commercial Web servers such as the Sun Java Web Server and Apache server also use multi-thread approach.

AWS also integrates Servlet container and CGI support to achieve better performance. The Servlet container API is integrated with the connection class of the AWS Web server; thereby sharing variables and other resources and improving the Servlet container's performance. Therefore, less memory is consumed for every thread. If there are one thousand concurrent connections, a 1k increase in memory for each connection should results a 1M increase for the whole process.

Parsing of incoming requests is also fine-tuned. All the protocol logic is pushed directly to a deeper level of the AWS server by implementation of a specially designed inputstream object. This means a request is parsed immediately after it is read. An extra storage structure and at least one loop through the incoming stream can be saved thereby.

Scaling can improve the server's performance by sharing computing resources among a group of machines. AWS has two scalable modules that can be easily integrated with a customer-designed load-balance technique and provides two types of scaling that can be extended to more customer types.

Chapter 2 Literature Survey

As the Internet grows and e-commerce becomes more popular, the World Wide Web has increasingly become part of our daily lives. The Web server lies at the core of WWW technology, and as a critical part of all commercial Web sites, directly affects their overall performance. Giant Web sites such as those operated by Yahoo!, AOL, and Amazon, deal with thousands or even tens of thousands of requests per second. This volume poses a challenge to both hardware and software. Hardware development and improvement is a long and relatively costly process, and improvements may not be that obvious. The evolution of Pentium 100M to Pentium III 1G took more than 5 years, though the performance of a Web server running on the newer machine may not have improved by as much as a factor of 10. However, within a specific hardware environment, appropriately chosen OS and Web server, may mean performance improvements as great as 100 times. The Microsoft Internet Information Server (IIS) benefits from its integration with the Windows NT/2000 platform, therefore outperformed all other Web servers in our survey.

2.1 Web Servers

ZD Net's lab conducted a study to compare ten Web servers from nine vendors running eight operating systems on five hardware platforms [13]. On Intel-based hardware and under Windows NT 4.0, they evaluated the Lotus Domino, the Luckman's Web Commander, the Microsoft Internet Information Server, the Netscape Enterprise, the Netscape FastTrack Server, and the O'Reilly & Associates' WebSite Professional. They also considered Enterprise and FastTrack running Solaris 2.5.1 on a Sun Netra, running

Irix on an SGI WebForce Origin200, and running Digital Unix on a Digital Alpha. They tested the public-domain version of Apache under RedHat Linux on a Compaq with Intel processors. They also looked at IBM Internet Connection Secure Server on Intel hardware running OS/2 Warp, Novell Web Server under IntranetWare 4.11, and StarNine's WebStar on a PowerMac running Mac OS.

2.1.1 Price and operating system of commercial Web servers

Table 1 Commercial Web server OSs and prices

Web Server	OS	Price
Apache	NT/Unix	Free
Microsoft Internet Information Server	NT/Unix	Incl. Win NT
Lotus Domino	Win NT	$995
Netscape Enterprise	Win NT	$1295
O'Reilly & Associates' WebSite	Win NT	$499
IBM Internet Connection Secure Server	OS/2 Warp	Incl. OS/2 Warp
Novell Web Server	IntranetWare	Incl. IntranetWare
StarNine's WebStar	Mac OS	$799

2.1.2 Commercial Web Server Performance Analysis

Figure 3 and Figure 4 shows the performance analysis done by ZD net lab. Figure 3 shows the performance of 1-processor platforms. Figure 4 shows the performance of 2-processor platforms. The x-axis of each diagram represents the number of clients accessing the server at the same time. The y-axis of each diagram represents the number of kilobytes per second total throughput by the server platform.

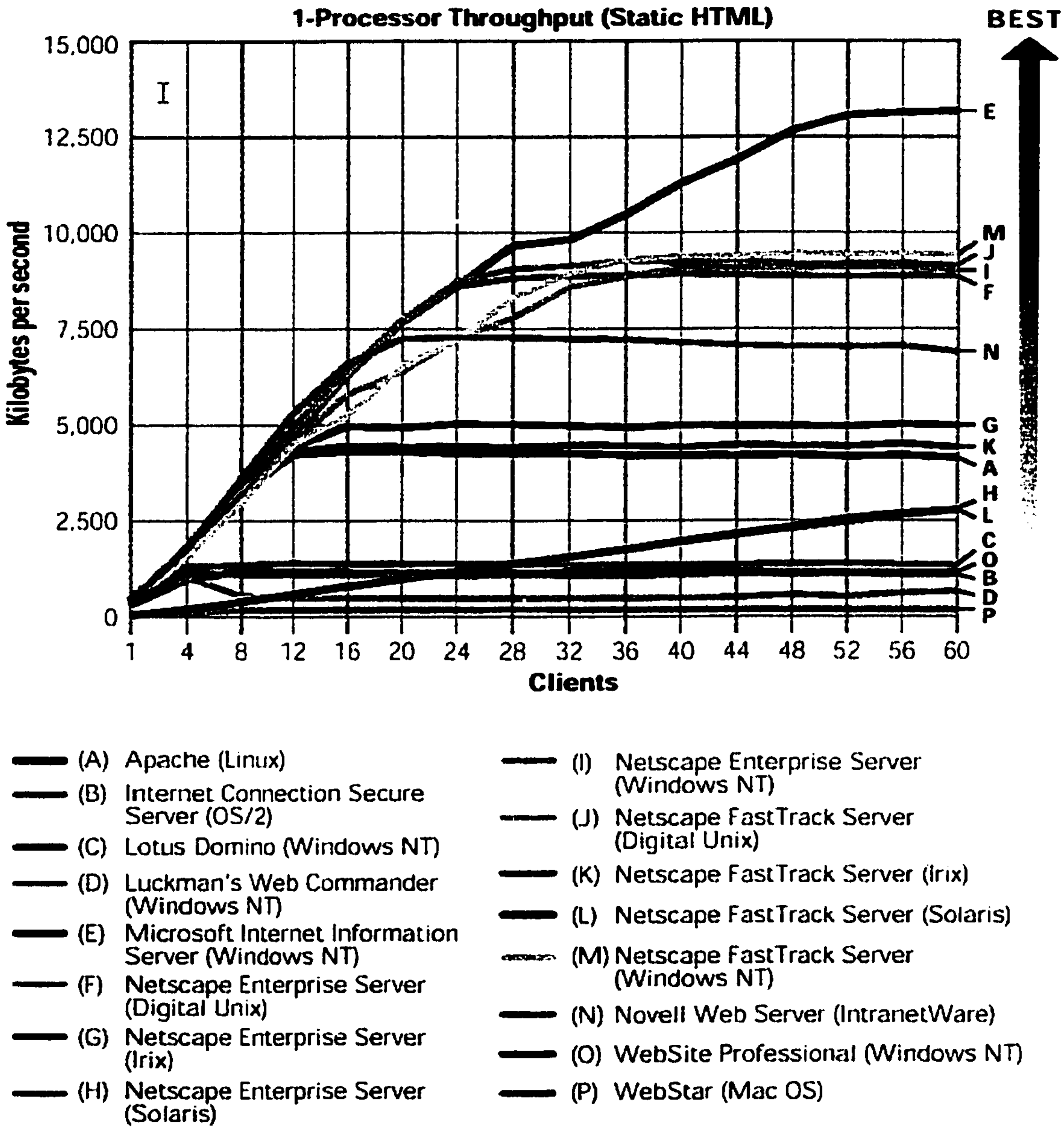

Figure 3 Performance analysis of commercial Web server (1-Processor)

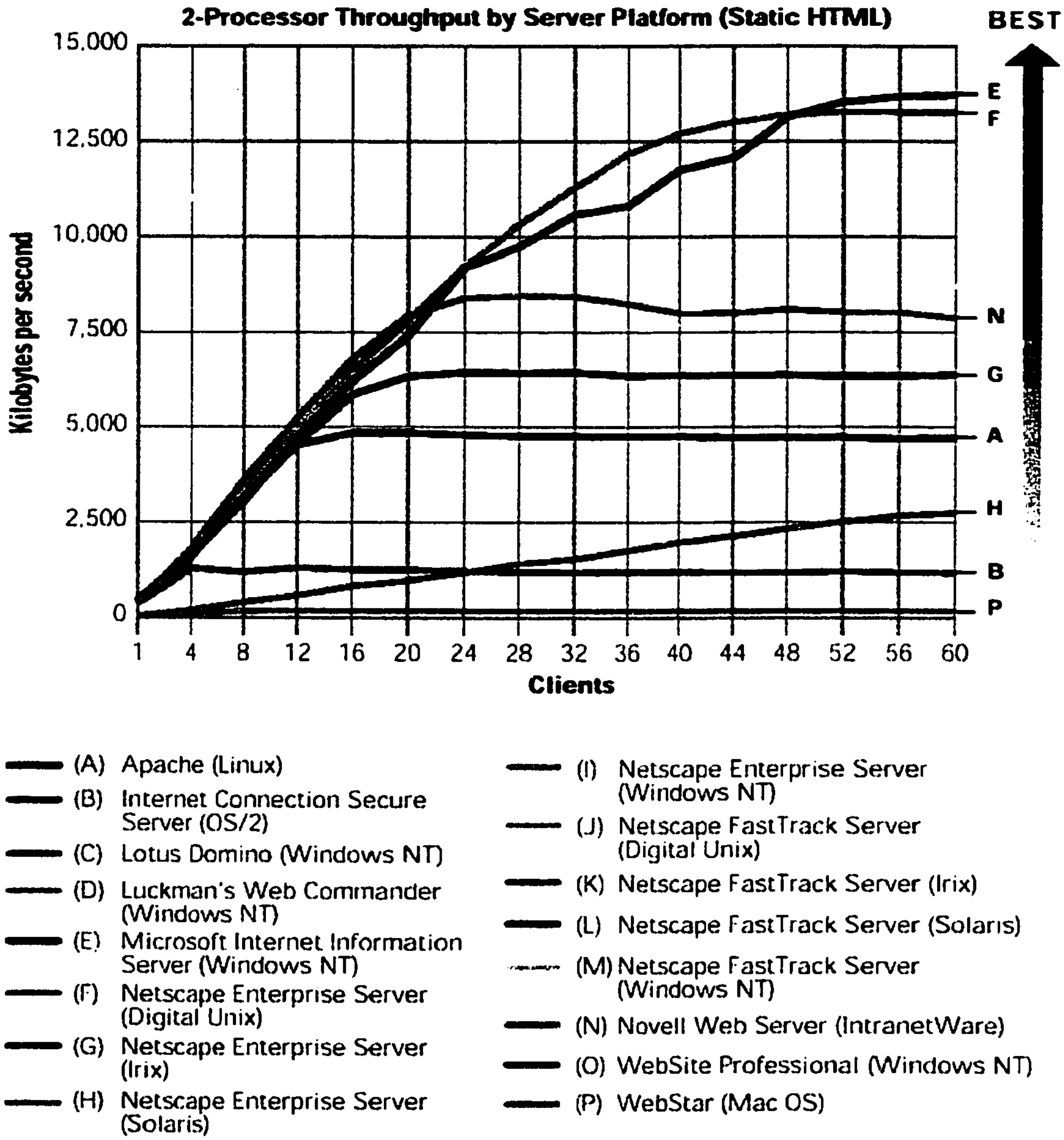

Figure 4 Performance analysis of commercial Web server (2-Processor)

From this analysis, we see that in both diagrams, Microsoft Internet Information Server achieves the best result. When client numbers are low, all Web servers exhibit about the same result. When the number of clients increases to 30, only Microsoft IIS's result continues to rise until the client number reaches 50. We can also see from this diagram that 2-processor architecture does not make significant change to results of most Web servers. Only Netscape Enterprise Server on NT platform showed a real improvement; at

30 clients, it achieves results that are almost the best in this test.

We can also see from this diagram that the operating system is very important to the Web server's performance. Netscape Enterprise Server performed very differently as it ran on 3 different platforms — Window NT, Irix and Solaris. It best results were seen on the Windows NT, where it almost doubled its performance compared to that on Irix, and tripled its performance compared to that on Solaris.

2.1.3 Apache Web Server

The Apache Web server benefits greatly from the Linux and open source revolution, and has become the most widely used Web server on the Internet. It offers a powerful and customizable approach for any Unix/Linux-based server. However, Apache's greatest strength is also its biggest shortcoming. Experienced Unix users enjoy the control they have over the Web server, which has its own API. Users may download Apache from their Web site and get all the Apache core and module source code, which can be modified to suit their needs. Creating their own Web servers by rewriting code are not, however, what newcomers to Unix are looking for.

Table 2 Apache Server Features

	Power	Ease
Documentation	Poor	Fair
Installation/configuration	Good	Poor
Management/administration	Good	Poor
Content and site management	N/A	N/A
Security	Fair	Fair

Web development	Fair	Poor

The Apache Project is a collaborative software development effort aimed at creating a robust, commercial-grade, featureful, and freely available source code implementation of an HTTP (Web) server. The project is jointly managed by a group of volunteers, located around the world, who use the Internet and the Web to communicate, plan, and develop the server and its related documentation. These volunteers are known as the Apache Group. In addition, hundreds of users have contributed ideas, code, and documentation to the project [9].

The newest version of Apache, Apache 2.0, improved the scalability problems we saw in the previous performance study. The 2.0 delivers POSIX threads support on the Unix platform; it can now run in a hybrid multiprocess, multithreaded mode. The system has been rewritten from scratch to be based on autoconf and libtool. These modifications makes Apache's configuration system easier to use and more similar to that of other packages.

Table 2 gives a ZD Net evaluation on the Apache server based on Apache 1.1.3 [14]. Even though the 2.0 release of Apache greatly improved the server in these aspects, it remains weak when compare to Microsoft IIS. Since Apache is an open-source project, support for it is not that great, documentation is not well organized, and overall, the system is not intended for the general user.

2.1.4 Microsoft Internet Information Server

Microsoft Internet Information Server (IIS) benefits tremendously from its integration with the Microsoft Windows NT operating system [15]. The result is a comprehensive solution that will surely help Windows NT take a bite out of the Unix-dominated Web server pie. IIS uses Windows NT's User Manager to maintain users and groups, saving the Web administrator the trouble of maintaining multiple sets of network and Web site users. The server also utilizes Windows NT's Event Viewer and Performance Monitor to view such items as bytes sent per second and current CGI requests. Web administrators get tighter security, better reporting, and the ability to use their Windows NT user database for Web server access control. The integrated package includes a rich suite of bundled tools, including FrontPage 2000 for Web page creation and management, Crystal Reports for reporting, and Index Server and Net Show for search capabilities and multimedia, respectively.

	Power	Ease
Documentation	Excellent	Good
Installation/configuration	Good	Excellent
Management/administration	Excellent	Excellent
Content and site management	Good	Excellent
Security	Good	Excellent
Web development	Excellent	Excellent

Table 3 Microsoft IIS Features

IIS is an extremely capable performer all around, one that would suit the requirements of any Web site. It performed well serving static pages and handling ISAPI (Information Server API) processing on the server side. Depending on the client load, IIS held its own against or outperformed Netscape servers on any platform with static pages and ISAPI.

IIS began to outshine its competitors as we increased the load to 56 and then 60 clients with static pages. This bodes well for its scalability.

IIS needs an external package to support Servlets. Only few packages can be found on the Internet that providers IIS Servlet support. Performance of these packages is relatively poor. Documentation and support for these packages is poor. IIS, designed and integrated to run Microsoft series of Web applications, is not a good choice for running a Servlet. Active Server Page is one possible substitute for Servlet on IIS.The latest version of IIS is version 5.0. It comes free with Windows 2000. Compared to other products, it is well documented and well supported. Support can be found on both the Microsoft Web site and through the worldwide Microsoft customer supporting service. IIS benefits from the new release of Windows NT 2000, is more reliable and robust and delivers a good performance. It provides developers of commercial Web sites (which mostly running on Unix/Linux platform) a choice on both OS and Web Server.

2.2 Application Servers

Application Servers run the software between browsers and data. For example, when a customer fills out and enters an order from a browser, a Web server sends the request to the application server, which executes the logic and also retrieves and updates customer data from back-end sources. The application server, rather than the client (browser, rich client), Web server or back-end system, runs the business programs in the middle between a client and an enterprise's data and other applications. The application server physically separates the business logic from the client and the data, and creates an architecture known as multi-tier computing. Application servers enable businesses to develop and deploy applications quickly and easily and increase the quantity of users without reprogramming. The separate tier is the key to this additional functionality.

2.2.1 Application Server Evolution

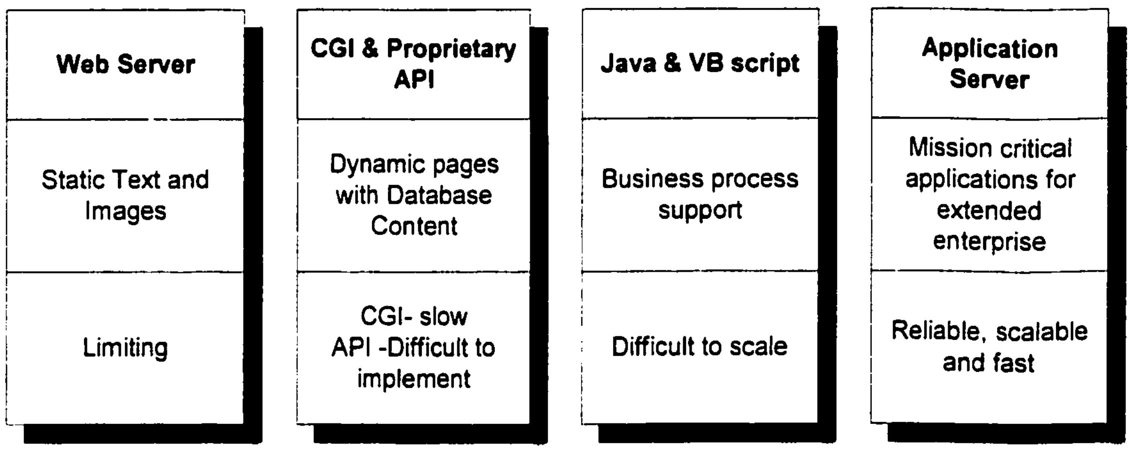

Figure 5 Application Server Evolution

As shown in Figure 5, before application servers were introduced, Web servers served only static text and images. Later, in response to fast growing Web application needs, CGI and server API were added to Web servers to allow dynamic pages and database content. However, CGI was slow and server API, with no general standard, was relatively difficult to implement. Server side Java script and VB script were introduced in an effort to replace CGI and server API and, indeed, server side Java script and VB script can suit business process needs in case of small applications. They are difficult to scale, however. The application server soon became the major solution for commercial Web application development, and provided reliable and scalable support for mission-critical applications. We will discuss a few commonly used Web application server approaches in the following section.

2.2.2 Web Application Approaches

Most applications on the Web today are one of following general types:

- Server-based CGI forms
- Web server APIs
- Active Server Pages
- Client-based Java applets
- Java Server Pages
- COBRA
- Enterprise JavaBeans (EJB)
- Microsoft DNA
- Servlets

CGI

An early Web solution was the CGI interface. Developers wrote individual programs to this interface, and Web-based applications called the programs through the Web server. This solution has significant scalability problems — each new CGI request launches a new process on the server. If multiple users access the program concurrently, these processes may consume all of the Web server's available resources and the performance slows down considerably.

One typical example of CGI is CGI forms, CGI forms are quick to download and allow developers to use heavy processes located on back-end servers to calculate/query results. Their disadvantage is that they require the user to hit the Submit button to get anything to happen, don't offer intuitive navigations between data fields, and are slow to regenerate the results screen. In general they are not friendly to use.

Security is another big concern. Most CGI scripts written in Perl use the command shell to execute OS commands with user-supplied data, for instance to send mail, search for information in a file, or just leverage OS commands in general. This use of a shell opens up many opportunities for a creative hacker to make the script remove all files on the server, mail the server's password file to a secret account, or other undesirables outcomes.

Web Server Proprietary APIs

The Web server vendors defined APIs to solve some of these problems, but an application written to a proprietary APIs such as Apache API is dependent upon its particular server vendor. If a developer needs to migrate this application to a server from another vendor, the development process would have to start from scratch. A further problem is that APIs typically support C/C++ code executing in the Web server process. If the application should crash, due to, say, a bad pointer or division by zero, it brings the Web server down with it.

Active Server Pages

Microsoft's Active Server Pages™ (ASP) technology makes it easier to create dynamic content on a Web page, but only works with Microsoft IIS or Microsoft Personal Web Server. ASP is a proprietary technology developed by Microsoft, and limits deployment to their own Web server. It utilizes a subset of the non-portable Visual Basic language, and is not suitable for commercial-quality development of Web applications for a number of reasons, including performance, database portability and access, and the structure of the language itself.

Java Applets

Java applets, on the other hand, usually download the business logic to the client machine and run what is essentially a little application there. Once downloaded they are quick to respond and can offer a more application-like user interface. However, applets can be painfully slow to download initially and they cannot easily access server-side processes. User must endure a slow download for each new piece of functionality in the overall application.

Java Server Pages (JSP) Technology

For the creation of pages with dynamically generated content. Java Server Pages (JSP) constitutes a good choice. This solution addresses the limitations of other alternatives by:

- Working on any Web or application server.
- Separating the application logic from the appearance of the page.
- Allowing fast development and testing
- Simplifying the process of developing interactive Web-based applications.

The JSP specification is the result of extensive industry cooperation between vendors of Web servers, application servers, transactional systems, and development tools. Sun Microsystems developed the specification to integrate with and leverage existing expertise and tools that support the Java programming environment, such as Java Servlets and JavaBeans™. The result is a new approach to the development of Web-based applications that extends powerful capabilities to page designers through the use of component-based application logic.

Java Servlets

Java Servlets is a technology that is rapidly replacing CGI scripts as the preferred choice for dynamic Web sites server-side processing. As Web sites get more and more interactive, customized and dynamic, developers turn to technologies that allow them to do more than what the first-generation of Web tools could handle.

One of the advantages over CGI is that a Servlet can retain information between requests and share common resources. When CGI is used, performance and scalability are big problems since a new process is created for each request, quickly draining a busy server of resources. Sharing resources such as database connections between scripts or multiple calls to the same script leads to repeated execution of expensive operations.

Java Servlets are based on a simple API supported by virtually all Web servers and even load-balancing, fault-tolerant application servers. They solve performance problems by executing all requests as threads in one process, or, in the case of a load-balanced system, in one process per server in the cluster. Servlets can easily share resources.

Security is improved by using Servlet in many ways when compared to CGI. First of all, the Java APIs provide access to all commonly used functions, and we rarely require that a shell execute commands with user-supplied data. We can use Java Mail to read and send email; Java Database Connect (JDBC) to access databases; the File class and related classes to access the file system; RMI, CORBA and Enterprise Java Beans (EJB) to access legacy systems. The Java security model makes it possible to implement fine-grained access controls, for instance, allowing access only to a well-defined part of the file system. Java's exception handling also makes a Servlet more reliable than proprietary C/C++ APIs: a divide by zero is reported as an error instead of crashing the Web server.

Servlet-based applications avoid a lot of processing overhead. Using threads instead of processes means that a Servlet can retain data between requests. For instance, multiple requests can share a pool of database connections and frequently requested information can be cached. Threading and persistence make it easier to develop high performance solutions.

Performance and scalability are important concerns when selecting a technology for Web site development. A popular site can receive an enormous number of requests per day. With inherited threading and techniques such as database connection pooling and caching, Servlet-based solutions are well able to handle the pressure.

CORBA

Common Object Request Broker Architecture (CORBA) [2] is the dominant distributed component model for ASP applications for which the components need to be deployed across various types of networks and on various platforms. The Object Management Group (OMG), an industry consortium consisting of over 800 IT companies, with the notable exception of Microsoft, specified CORBA.

CORBA uses Object Request Broker (ORB) to provide network connectivity for its components, and uses a neutral Interface Definition Language (IDL) to separate interface specification and the implementation of a component. CORBA components support universal reference, network interoperability, introspection. Currently all Netscape Web browsers have a built-in ORB to support CORBA-based applications embedded in Web contents.

A special feature of CORBA is that it can easily wrap up legacy code in CORBA wrapper components, thus providing a fast-track approach to adapt legacy code to the ASP model.

The ultimate goal of CORBA is system integration. OMG uses IDL to standardize the specification of vertical and horizontal common facilities for system integration [5].

Enterprise JavaBeans (EJB)

Java, developed by Sun Microsystems, has become the foundation of a powerful environment for developing and running server-based applications [3]. Enterprise

JavaBeans (EJB) is a component-based infrastructure framework that forms the basis of many high-end application servers. While JavaBean is the Java component model that mainly runs on a client machine, an EJB is a specialized, non-visual JavaBean that runs on a server. The EJB server-side component architecture brings together most of the properties and services we described for our component reference model.

Since its introduction over two years ago, Enterprise JavaBeans technology has maintained unprecedented momentum among platform providers and enterprise development teams alike. That's because the EJB server-side component model simplifies development of middleware components that are transactional, scalable, and portable. Enterprise JavaBeans servers reduce the complexity of developing middleware by providing automatic support for middleware services such as transactions, security, database connectivity, and more.

EJB by itself can only support the integration of Java components. But EJB and CORBA are complementary. CORBA has become the implementation technique of EJB Remote Method Invocations (RMI). EJB has provided CORBA with a friendlier user-interface. EJB augments CORBA with declarative transactions, a server-side component framework, and tool-oriented deployment and security descriptors. CORBA augments EJB with a distributed object framework, multilingual client support, and IIOP (Internet-Inter-ORB Protocol) interoperability.

To integrate an existing CORBA component into an EJB framework, we only need to use the IDL-generated JavaBean proxy to represents the original CORBA component. It is very easy, therefore, to take advantage of both these component models [5].

Microsoft DNA

While CORBA and EJB are both based on open standards, Microsoft Distributed interNetwork Applications (DNA) is a proprietary technology based on Microsoft's COM+ [4]. DNA represents Microsoft's vision of networked computing; it is an application architecture to compete with CORBA and EJB. The objective of DNA is to fully embrace and integrate the Internet, client/server, and PC models of computing to support the development of scalable, multi-tier business applications that can be delivered over any network.

The most significant difference between Windows DNA and CORBA/EJB is that Windows DNA is a proprietary product supported by a single vendor, whereas CORBA/EJB is an open industry standard supported by a variety of middleware vendors, each of which are providing implementations for the standard. The cornerstone of Windows DNA is COM+, a language-independent technology used to build re-usable components. COM+ is an enhanced version of (Distributed) Component Object Model (COM/DCOM) integrated with Microsoft Transaction Server (MTS), the supporting environment for COM components. Unlike CORBA and EJB, COM is a binary standard to support the interaction among component instances with pointers to interfaces.

The main strengths of COM+ are its position as a mature core-supporting technology for Windows applications in the last decade; close integration with Windows applications; and user-friendly development environments. Therefore it is a very attractive platform for developing ASP applications for servers and clients using Windows. The disadvantages of COM+ are that it is basically native to Windows platform; that, compare with CORBA, it currently offers limited services for distributed computing and scalability; and that it lacks competition that would compel the perfection of the technology. Even though Microsoft has made effort to port COM to other platforms, MTS, the supporting environment for COM, proved to be a big obstacle to this effort due to its high platform dependency. In recent years Microsoft tried to use interoperability to compensate for COM's platform dependency. With DCOM/CORBA bridges, COM component instances running on Windows can interact with CORBA/EJB component instances running on other platforms. Microsoft supports Java, but only as a language, not as a platform [5].

2.2.3 Servlet Advantages

Servlets have the following advantages over other common server extension mechanisms:

- They are faster than CGI scripts because they use thread instead of process.
- They use a standard API that is supported by many Web servers.
- They have all of the advantages of the Java language, including ease of development and platform independence.
- They can access a large set of APIs available for the Java platform (e.g. database access, email, directory servers, CORBA, RMI, Enterprise JavaBeans and others).

All business logic runs on the back-end server, which means that it is never downloaded to the user machine and can easily access server processes, such as database queries, and multi-processor calculation engines.

"Java is a great platform for writing the server side of your Web-based application. Servlets are emerging as the preferred way of creating highly dynamic WWW services linked to real backend applications and databases. See the growing trend at *http://www.servletcentral.com*. " [7]

"Java makes it easy to develop and deploy all parts of a professional, maintainable distributed system application. The Servlet API provides you the fastest way to start using Java Server technology in your networked applications. You can start with applications that involve clients and a single server, and gradually create multi-tier enterprise applications which integrate the power and flexibility of Java throughout your existing network ... because Java Servlets run on the software and hardware you've already installed." [7]

The AWS project selected Java Servlet as an application server approach due to the above reasons. After all, AWS is not a commercial product and we have designed AWS more for research purpose: it is a prototype that will allow readers of this thesis to understand application servers better. Java is a pure object oriented language. Java applications are easy to implement and easy to read. Our use of Java Servlets as an application server approach will help readers better read and understand the implementation and structure of AWS.

Chapter 3 Web Server Concept

The term "Internet" applies to a collection of networks, including the Arpanet, NSFnet, regional networks such as NYsernet, local networks at a number of University and research institutions, and a number of military networks. The subset of them managed by the US Department of Defense is referred to as the "DDN" (Defense Data Network), and includes some research-oriented networks, such as the Arpanet, as well as those of a purely military character. (Because the DDN organization provides much of the funding for Internet protocol development, the terms Internet and DDN can sometimes seem equivalent.)

All of these networks are connected to one another. Users can send messages from any of them to any other, except where there are security or other policy restrictions on access. TCP/IP (Telecommunication Protocol/Internet Protocol) is a family of protocols that provide low-level network functions needed for many applications. Other protocols perform specific tasks, e.g. transferring files between computers, sending mail, or finding out who is logged in on another computer. Initially TCP/IP was used mostly in communications between minicomputers or mainframes. These machines had their own disks, and generally were self-contained.

3.1 Network and DNS Structure

As shown in Figure 6, a sample site called samplesite.com server, with an IP address of 123.456.789.123, is connected to the Internet. A Web server application is running on this machine. Many client machines are also connected to the Internet, and a Web browser application is running on each of these machine, The client machines may use two methods to access the samplesit.com Web server: by using the IP address 123.456.789.123 or by using the domain name samplesite.com. If a client uses the IP address, the connection is done directly, if the client uses domain name, a Domain Name Server (DNS) will translate the domain name to the IP address before connecting to the site.

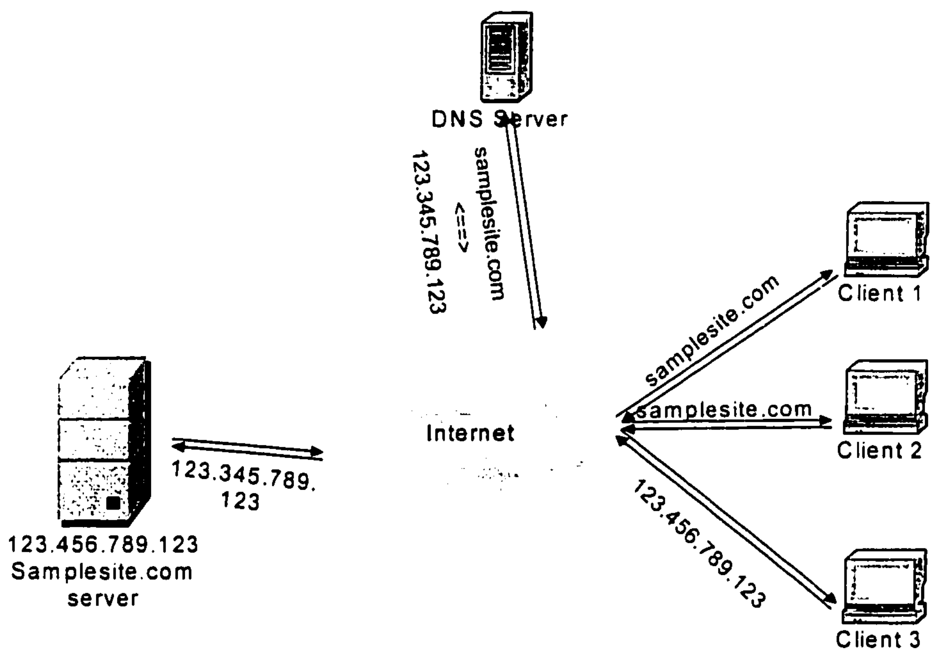

Figure 6 Physic Structure of Web and DNS Translate process

3.2 TCP/IP Connection

An Internet TCP/IP connection is a point-to-point bi-directional connection. When Client

1 requests a Web page on the samplesite.com server, a connection between Client 1 and the samplesite.com server will be established. Client 1 can send the page request and receive the response page data through this connection.

3.2.1 Socket

A socket is an endpoint for communication between two machines. There are in general two types of sockets. One is connection socket, which is an established connection between a client and a server. This connection socket can be read from and written to just like a file. The other type of socket is listening socket, which is a server-side socket that is not connected to any specific client, but rather listens to a certain port (see Subsection 3.2.2). The listening socket listens for any incoming new requests from that port. Once a new request comes in, a peer-to-peer connection is established. This creates a new connection socket on another port, and keeps the original socket listening for incoming connections on the defined port.

3.2.2 Port

A port is a communication channel through which client and server can be connected and exchange data. It is a "logical connection place" and specifically, when we use the Internet's protocol (TCP/IP), a port is the way a client program specifies a particular server program on a computer in a network. Higher-level applications that use TCP/IP, such as the Web protocol, HTTP, have ports with preassigned numbers. These "well-known ports" have port number assigned by the Internet Assigned Numbers Authority. Other application processes are given port numbers dynamically for each connection. When an Internet service is initially started, it is said to bind to its designated port number. Any client program that wants to use that server must request to bind to the designated port number [5].

Port numbers are from 0 to 65536. Ports 0 to 1024 are reserved for use by certain privileged services. For the HTTP service, port 80 is defined as a default and it does not have to be specified in the Uniform Resource Identifier (URI). The most common form of URI is the Web page address, which is a particular form or subset of URI called a Uniform Resource Locator (URL) [[16]].

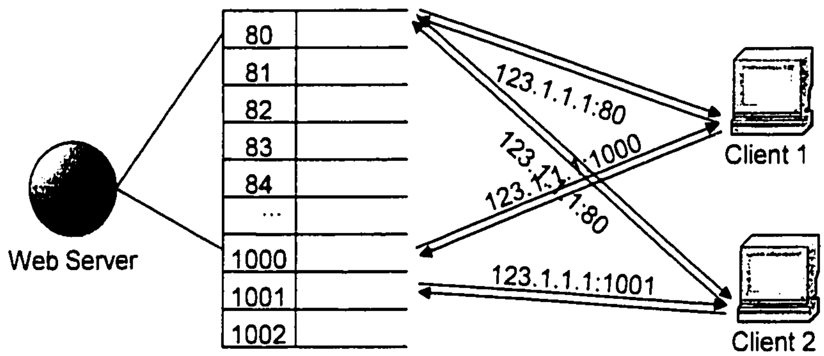

Figure 7 Web Server port assignment

In the Example shown in Figure 7, a Web server is running on samplesite.com and listening to port 80. Client 1 requests a Web page from samplesite.com by sending an HTTP request through port 80 of IP 123.1.1.1. The listening socket on the server will accept this request and assign a free port (usually randomly selected) to the client. Client 1 receives the number of the free port, port 1000 in this case, and establishes a socket connection to port 1000 of IP 123.1.1.1. Then client and server can communicate with each other through this communication channel.

3.3 Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol that can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

3.3.1 Overview

HTTP has been in use by the World Wide Web global information initiative since 1990. . The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0, as defined by RFC 1945, improved the protocol by allowing messages to be in the format of MIME-like messages, containing Meta information about the data transferred and modifiers on the request/response semantics. However, HTTP/1.0 did not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts. In addition, the proliferation of incompletely implemented applications calling themselves "HTTP/1.0" necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities. The current version, HTTP 1.1, includes more stringent requirements than HTTP/1.0 in order to ensure reliable implementation of its features. Practical information systems require more functionality, including search, update, and annotation, than simple retrieval. HTTP allows an open-ended set of methods and headers that indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI), as a location (URL) or name (URN), to indicate the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail as defined by the Multipurpose Internet Mail Extensions (MIME).

HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by the Simple Mail Transfer Protocol (SMTP), Network News Transfer Protocol (NNTP), FTP, Gopher, and Wide Area Information Server (WAIS) protocols. In this way, HTTP allows basic hypermedia access to resources available from diverse applications.

3.3.2 HTTP 1.1 versus HTTP 1.0HTTP 1.0

1. HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, and virtual hosts.
2. Closing the connection causes loss of congestion information
3. Connection opens may congest low bandwidth links, due to a lack of flow control on TCP opens and closes
4. Poor user-perceived performance (most connections in slow-start)
5. Workaround has been to open multiple simultaneous connections, with resulting congestion problems
6. Servers have thousands of connections in close & wait state
7. Cost is primarily memory, on systems running reasonable TCP implementations
8. Large scale servers with HTTP result in big servers using 100's of I.P. addresses and consequential routing headaches
9. Caching model is primitive, and broken enough that content providers often defeat caching**HTTP 1.1**

1. Reduces HTTP's impact on the Internet, and make HTTP a 'well behaved'

Internet protocol

2. Improves user-perceived performance

3. Reduces load and increase performance of HTTP servers

4. Enables reliable applications in the face of caching

5. Should help congestion and other operational problems of parts of the Internet

6. Decreases load of HTTP on the Internet

3.3.3 HTTP Connection

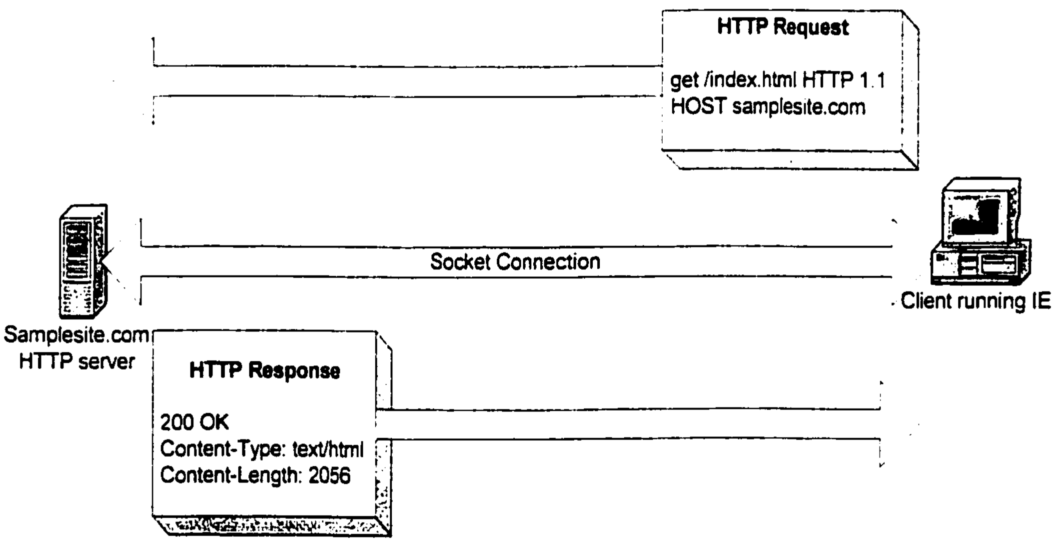

Figure 8 HTTP request and HTTP response headers

Once a socket connection is established, two sockets, on both the server side and the client side, communicate with each other by using HTTP protocol. HTTP protocol can be separated into two parts: HTTP request and HTTP response. An HTTP request is send to the HTTP server by a client that contains the client's request information; HTTP response is the server's response to that request. Figure 8 shows a typical HTTP request and response set. Through a socket connection, an HTTP request has been sent to the server requesting a Web page: /index.html. The HTTP server finds the Web page and informs the client that this resource is available and will be sent after the response header. Content type of this Web page is text and html; content length of this Web page is 2056.

3.3.4 Persistent Connection

Prior to persistent connections, a separate TCP connection was established to fetch each URL, increasing the load on HTTP servers and causing congestion on the Internet. The use of inline images and other associated data often require a client to make multiple requests of the same server in a short period of time.

A typical Web page in current days contains at least 5 files. In HTTP 1.0, to fetch these 5 files, at least 5 connections will be established and 5 file requests will be send through these 5 connections. With HTTP 1.1 support, in a default setting, a client browser will open at most 2 connections to one server location. Client and server may pipeline the request and response. **Persistent HTTP connections have a number of advantages:**

- Opening and closing fewer TCP connections saves CPU time in routers and hosts (clients, servers, proxies, gateways, tunnels, or caches), and, in hosts, saves memory used for TCP protocol control blocks.

- HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently, with much less elapsed time.

- Network congestion is reduced by reducing the number of packets caused by

TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.

- Latency on subsequent requests is reduced since there is no time spent in TCP's connection opening handshake.

- HTTP can evolve more gracefully, since errors can be reported without the penalty of closing the TCP connection. Clients using future versions of HTTP might optimistically try a new feature, but when communicating with an older server, retry with old semantics after an error is reported.

3.3.5 HTTP Request

An HTTP request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

Request = **Request-Line**
***(header *CRLF*)**
CRLF
[message-body]

Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with ***CRLF***. The elements are separated by ***SP*** (space) characters. No ***CR*** or ***LF*** is allowed except in the final CRLF sequence.

Request-Line **= Method** ***SP*** **Request-URI** ***SP*** **HTTP-Version** ***CRLF***

Example of a request line:

get /index.html

post /counter.cgi HTTP 1.1

HTTP 0.9 does not implement the HTTP version token. Therefore the first line of the example is using HTTP 0.9. The later versions, HTTP 1.0 and HTTP 1.1, support the token and a request line will be like the second line of the example.

Method

The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.

Method **= "GET"| "HEAD"| "POST"| "PUT"| "DELETE"**

All general-purpose servers MUST support the methods GET and HEAD. All other methods are OPTIONAL.

GET

The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data that shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since field. A conditional GET method requests that the entity be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.

HEAD

The HEAD method is identical to GET except that the server must not return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request may be cacheable in the sense that the information contained in the response may be used to update a previously cached entity from that resource. If the new field values indicate that the cached entity differs from the current entity (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache must treat the cached entry as stale.

POST

The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotating existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI. The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.

The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result.

Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource.

Header Fields

Header Fields could be combination of request headers, entity headers and general header.

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation.

request-header = Accept| Accept-Charset| Accept-Encoding| Accept-Language

| Authorization| Expect| From| Host| If-Match| If-Modified-Since

| If-None-Match| If-Range| If-Unmodified-Since| Max-Forwards

| Proxy-Authorization| Range| Referer| TE| User-Agent

Request-header field names can be extended reliably only in combination with a change in the protocol version. However. new or experimental header fields may be given the semantics of request- header fields if all parties in the communication recognize them to be request-header fields. Unrecognized header fields are treated as entity-header fields.

Entity-header fields define metainformation about the entity-body or. if no body is present, about the resource identified by the request. Some of this metainformation is optional: some might be required by portions of this specification.

Entity-header = Allow| Content-Encoding| Content-Language| Content-Length|

Content-Location| Content-MD5| Content-Range| Content-Type|

Expires| Last-Modified| Extension-Header

The extension-header mechanism allows additional entity-header fields to be defined without changing the protocol, but these fields cannot be assumed to be recognizable by the recipient. Unrecognized header fields SHOULD be ignored by the recipient and MUST be forwarded by transparent proxies.

There are a few header fields which have general applicability for both request and response messages, but which do not apply to the entity being transferred. These header fields apply only to the message being transmitted.

General-header = Cache-Control| Connection| Date| Pragma| Trailer|

Transfer-Encoding | Upgrade| Via| Warning

General-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields may be given the semantics of general header fields if all parties in the communication recognize them to be general-header fields. Unrecognized header fields are treated as entity-header fields.

Examples of Request Header:

Host Header

The Host request-header field specifies the Internet host and port number of the resource being requested, as obtained from the original URI given by the user or referring resource (generally an HTTP URL). The Host field value must represent the naming authority of the originating server or gateway given by the original URL. This allows the origin server or gateway to differentiate between internally ambiguous URLs, such as the root "/" URL of a server for multiple host names on a single IP address.

Host = "Host: " host [":" port]

A "host" without any trailing port information implies the default port for the service requested (e.g., "80" for an HTTP URL). For example, a request on the origin server for <http://www.w3.org/pub/WWW/> would properly include:

GET /pub/WWW/ HTTP/1.1

Host: www.w3.org

A client must include a Host header field in all HTTP/1.1 request messages. If the requested URI does not include an Internet host name for the service being requested, then the Host header field must be given with an empty value. An HTTP/1.1 proxy must ensure that any request message it forwards does contain an appropriate Host header field that identifies the service being requested by the proxy. All Internet-based HTTP/1.1 servers must respond with a 400 (Bad Request) status code to any HTTP/1.1 request message that lacks a Host header field.

If-Modified-Since

The If-Modified-Since request-header field is used with a method to make it conditional: if the requested data has not been modified since the time specified in this field, an entity will not be returned from the server; instead, a 304 (not modified) response will be returned without any message-body.

If-Modified-Since = "If-Modified-Since: " HTTP-date

An example of the field is:

If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

A GET method with an If-Modified-Since header and no Range header requests that the identified entity be transferred only if it has been modified since the date given by the If-Modified-Since header. The algorithm for determining this includes the following cases:

- If the request would normally result in anything other than a 200 (OK) status, or if the passed If-Modified-Since date is invalid, the response is exactly the same as for a normal GET. A date that is later than the server's current time is invalid.

- If the data has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

- If the data has not been modified since a valid If-Modified-Since date, the server should return a 304 (Not Modified) response.

The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead.

Note: If-Modified-Since times are interpreted by the server, whose clock might not be synchronized with the client.

Note: When handling an If-Modified-Since header field, some servers will use an exact date comparison function, rather than a less-than function, for deciding whether to send a 304 (Not Modified) response. To get best results when sending an If-Modified-Since header field for cache validation, clients are advised to use the exact date string received in a previous Last-Modified header field whenever possible.

Note: If a client uses an arbitrary date in the If-Modified-Since header instead of a date taken from the Last-Modified header for the same request, the client should be aware of the fact that this date is interpreted in the server's understanding of time. The client should consider unsynchronized clocks and rounding problems due to the different encoding of time between the client and server. This includes the possibility of race conditions if the document has changed between the time it was first requested and the If-Modified-Since date of a subsequent request, and the possibility of clock-skew-related problems if the If-Modified-Since date is derived from the client's clock without correction to the server's clock. Corrections for different time bases between client and server are at best approximate due to network latency.

3.3.6 HTTP Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

Response = Status-Line
***(header CRLF)**
CRLF
[message-body]

Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

-1xx: Informational - Request received, continuing process

- 2xx: Success - The action was successfully received, understood, and accepted

- 3xx: Redirection - Further action must be taken in order to complete the request

- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled

-5xx: Server Error - The server failed to fulfill an apparently valid request

The individual values of the numeric status codes defined for HTTP/1.1, and a sample set of corresponding Reason-Phrase's, are presented below. The reason phrases listed here are only recommendations -- they may be replaced by local equivalents without affecting the protocol.

Status-Code	**Reason-Phrase**	**Status-Code**	**Reason-Phrase**
100	Continue	101	Switching Protocols
200	OK	201	Created
202	Accepted	203	Non-Authoritative Information
204	No Content	205	Reset Content
206	Partial Content	300	Multiple Choices
301	Moved Permanently	302	Found
303	See Other	304	Not Modified

305	Use Proxy	307	Temporary Redirect
400	Bad Request	401	Unauthorized
402	Payment Required	403	Forbidden
404	Not Found	405	Method Not Allowed
406	Not Acceptable	407	Proxy Authentication Required
408	Request Time-out	409	Conflict
410	Gone	411	Length Required
412	Precondition Failed	413	Request Entity Too Large
414	Request-URI Too Large	415	Unsupported Media Type
416	Requested range not satisfiable	417	Expectation Failed
500	Internal Server Error	501	Not Implemented
502	Bad Gateway	503	Service Unavailable
504	Gateway Time-out	505	HTTP Version not supported

Table 4 HTTP response status code and Reason-Phrase

HTTP status codes are extensible. HTTP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications must understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response must not be cached. For example, if the client receives an unrecognized status code 431, the client can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code.

200 OK

The request has succeeded. The information returned with the response is dependent on the method used in the request, for example:

GET: An entity corresponding to the requested resource is sent in the response.

HEAD: The entity-header fields corresponding to the requested resource are sent in the response without any message-body.

POST: An entity describing or containing the result of the action.

TRACE: An entity containing the request message as received by the end server.

304 Not Modified

If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server should respond with this status code. The 304 response must not contain a message-body, and thus is always terminated by the first empty line after the header fields.

400 Bad Request

The server could not understand the request due to malformed syntax. The client should not repeat the request without modifications.

500 Internal Server Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

Response Header Fields

The response-header fields allow the server to pass additional information about the response that cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

response-header = Accept-Ranges| Age| ETag| Location| Proxy-Authenticate

| Retry-After| Server| Vary| WWW-Authenticate

3.3.7 Stateless feature of HTTP

HTTP is a stateless protocol. A protocol is said to be stateless if it has no memory of prior connections and cannot distinguish one client's request from that of another. FTP is a stateful protocol because the connections is not opened and closed with every request. After the initial login, the FTP server maintains the user's credentials throughout the session. On the other hand, due to its stateless nature, there is no inherent method in

HTTP for tracking a client's traversal of a Web site. Every request uses a new connection from an anonymous client. State is extremely useful for secure sites that require a user to login or for electronic commerce sites that provide customers with a virtual shopping cart.

The stateless nature of HTTP is both its strength and a weakness. It is strength in that its stateless nature keeps the protocol simple and straightforward. It also consumes fewer resources on the server and can support more simultaneous users since there are no client credentials and connections to maintain. The disadvantage is the increased overhead required to create a new connection with each request and the inability to track a single user as he traverses a Web site.

Each command of HTTP is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement Web sites that react intelligently to user input. This shortcoming of HTTP is being addressed by a number of new technologies, including ActiveX, Java, JavaScript and cookies.

When a user browses the World Wide Web, which is based on HTTP protocol, each request for a new Web page is processed without any knowledge of previous pages requested. This is one of the chief drawbacks of the HTTP protocol. Because maintaining state is extremely useful, programmers have developed a number of techniques to add state to the World Wide Web. These include server APIs, such as NSAPI and ISAPI, the use of cookies (see Section 4.2.2) and Servlet sessions (see Section4.2.1).

After the server has responded to the client's request, the connection between client and server is dropped and forgotten. There is no "memory" between client connections. The pure HTTP server implementation treats every request as if it was brand-new, i.e. without context.

CGI applications (see Section 2.2.2) get around this by encoding the state (or a state identifier) in hidden fields, in the path information, or in URLs in the form being returned to the browser. In a CGI form submission, the first two methods always return the state or (its id) when the user click the submit button. On the other hand, the method of encoding state into hyperlinks (URLs) in the form only returns the state (or id) if the user clicks on the link and the link is back to the originating server.

It's often advisable not to encode the whole state but to save it, for instance, in a file, and identify it by means of a unique identifier, such as a sequential integer. Visitor counter programs can be adapted very nicely for this - and thereby become useful. It is then necessary only to send the state identifier in the form, which is advisable if the state vector becomes large, thus saving network traffic. However the Web administrator must then take care of the periodic housekeeping of the state files.

Chapter 4 Servlet Container Concept

4.1 Servlet Server

Just as regular Web pages are "served" by a Web server application, Servlets are also provided to the client via a Web server. This can be the same Web server used for static pages (as is the case with Sun's Java Web Server), or an entirely different product (such as JRun) that runs on either the same or a different server. Running Servlets on a Web server requires a Servlet container, which can also be called as Servlet container.

In a typical situation, the user of the system first browses through static Web pages, which act as a navigation guide to the various parts of the system (much like menus in legacy systems). The user then clicks a link that refers to a Servlet, and some action takes place as a result of the link being activated.

Today, Servlets are a popular choice for building interactive Web applications. Third-party Servlet containers are available for Apache Web Server, iPlanet Web Server, Microsoft IIS, and others. Servlet containers can also be integrated with Web-enabled application servers, such as BEA WebLogic Application Server, IBM WebSphere, iPlanet Application Server, and others.

A Servlet container should implement the Java Servlet Development Kit (JSDK) API. The latest version of JSDK is 2.1. Sun will continue responsibility for the Servlet API and JavaServer Pages (see Subsection 2.2.2) specifications, which are being developed under the Java Community Process.

The Apache Server does not support Servlets directly; the Jserv project is designed as add-on to the Apache as a Servlet container. However, Jserv supports only Java Servlet Development Kit API 2.0. The Apache group rewrote the Jserv from scratch with JSDK 2.1 support and named it Jakarta Tomcat. It is the most popular Servlet container on the Apache server. It is the official reference Implementation for the Java Servlet 2.1 and JavaServer Pages 1.1 Technologies.

AWS supports Servlets with a build-in Servlet container, which is easier to maintain and configure. The Servlet container benefits from its integration with the Web server and performs very well in the ZD Net Web Bench test. It supports the latest JSDK 2.1. However, it does not support the JSP API. The next version of AWS will have this feature.

4.2 Servlet State (Session) Management Approaches

There is more than one definition of the term "session". In the traditional sense, a session is a persistent network connection between two hosts (usually a client and a server) that facilitates the exchange of information. When the connection is closed, the session is over. However, a Servlet session is not this kind of session. It is also called a virtual session, because it involves a virtual connection between the client and the server rather than a physical connection. A typical HTTP transaction goes like this: the client establishes a connection to the server, issues a request, receives a response, and then closes the connection. The server's tie to the client is serverd once the connection is closed. If the same client issues a new request, the server cannot associate this connection with the client's previous one. The fact that the server "forgets about" the client after the connection is closed presents significant challenges to the HTTP protocol. If not remedied, this disconnect between client and server can lead to the following problems and limitations:

- If the server requires client authentication (e.g. a client must log in), the client must reauthenticate with every request. The server does not realize that it has already authenticated this client because the connection between the two was lost.

- Storing user-specific information such as the contents of a shopping cart or user-defined preferences is not possible because the server cannot distinguish one client from another.

The solution to these problems is to establish a persistent "virtual connection" between the client and the server. A virtual connection associates each request received by the server with the client that issued it. This association is accomplished by requiring the client to return a piece of state information with each new request. The server uses this piece of information (usually called a session ID or user ID) to uniquely identify the client and associate the current request with the client's previous requests. By allowing the server to identify the client, virtual connections alleviate the problems presented above. These virtual connections are commonly referred to as sessions. Sessions are used to maintain state and client identity across multiple requests.

4.2.1 HTTP Session

HTTP Session vs. Traditional Session

In a traditional session, all client requests are associated by virtue of the fact that they share the same network connection. In contrast, an HTTP session associates client requests by virtue of the fact that they all share the same session ID. A traditional session refers to the duration of time that a network connection is open. Similarly, an HTTP session refers to the duration of time that a virtual connection is active. In short, an HTTP session is a series of associated requests in which the client can be uniquely identified. This association between the HTTP client and the HTTP server persists across multiple requests and/or connections for a specific period of time.

The traditional type of session expires when the network connection is closed. However, since an HTTP session persists between requests (with each request opening and closing a connection), the duration of an HTTP session is configurable on the server. A common approach is to instruct the server to discontinue any sessions that have been inactive for more than a specified amount of time, say fifteen minutes. In this case, the session will persist between a client's requests as long as they are received no more than fifteen minutes apart. If the client waits more than fifteen minutes before issuing a new request, the server will end the client's session and the client will be required to log in again (The server usually issues a valid session ID with each login). Automatically ending sessions after a period of inactivity ensures that server resources are not consumed by old sessions.

When a session expires, it means that the unique session ID used to identify a client is removed from storage along with any associated data (e.g., shopping cart contents). Session IDs are usually stored in memory or in a database. Once a session ID expires and is removed from storage, the client that utilized this session ID must log in again in order to be assigned a new session ID.

In AWS implementation, a session manager automatically creates new session objects whenever a new session starts. In some circumstances, clients do not join the session, for example, if the session manager uses cookies and the client does not accept cookies.

The session ID generator, which is used for Servlet sessions, employs a unique random number generation algorithm. AWS uses the same approach as Apache Jserv to generate session IDs.

How Do Servlets Access Session Data?

To access the state information stored in a session object, your Servlet can create a new session as follows:

```
// request is an HttpServletRequest that is passed to the Servlet
SessionClass session = request.getSession(true);
```

The Servlet can call any of the public methods in javax.servlet.http.HttpSession on the session object.

4.2.2 Cookie

Cookies present the simplest way to store state information on the client. A cookie is a simple mechanism that allows the server to instruct the client to store small amount of state information. When an HTTP server receives a request, in addition to the requested document, the server may choose to return some state information that is stored by a cookie-enabled client. This state information includes a URL range within which the information should be returned to the server. The URL range is composed of the server's domain and some path information. Whenever the client issues an HTTP request, it checks the URL of the request against the URL ranges of all stored cookies. If a match is found, the state information is included in the client's request. In this way, the server can effectively overcome the stateless nature of the Web and track a client from request to request.

The advantages of using cookies are many. First, cookies are the simplest way to store state information on the client because this information need only be stored once. The server does not have to keep returning this information to the client, as is the case with rewritten URLs and hidden variables. And unlike storing the session ID in the URL path, either relative or absolute URLs may be used without losing state. Also, in contrast to either of the previous two methods, cookies do not require paring of the requested URL or the HTML document. Cookie information can be extracted from the client request using a very simple Servlet API method. Thus, server-side processing is kept to a minimum. Cookies provide a simple, low-overhead method of maintaining state and session.

Despite the many benefits provided by cookies, there is one distinct disadvantage: they are not supported by all browsers. This lack of cookie support may result from one of two causes: either the client is using an older browser that does not recognize cookies or the user has instructed the browser not to accept cookies. Some users may, often out of ignorance, turn off cookie support in their browser for fear that cookies may somehow compromise security or exhaust their system's resources (or some other unfounded fear). Considering that virtually all current browsers support cookies, the chance that a user is running a browser that does not recognize cookies is small. The possibility that a user might manually disable cookie support and, in turn, disable a Web site's mechanism for state and session management is of greater concern. Some sites may therefore choose to use an alternate session management mechanism.

Memory-based cookies

There are two types of cookies, the memory-based cookie does not save any information to your hard disk once your browser is terminated or closed. Or the disk-based cookie will be stored in a predefined cookie directory. The default settings for most IE and

Netscape browsers have memory-based cookies enabled. Even by increasing the level of security to the maximum, memory-based cookies are still accepted without notification (only by going into custom settings can memory-based cookies be disabled in IE).

How cookie works

The main purpose of cookies is to identify users and possibly prepare customized Web pages for them. In a typical situation, when you enter a Web site using cookies, you may be asked to fill out a form providing such information as your name and interests. This information is packaged into a cookie and sent to your Web browser, which stores it as a text file for later use. The next time you go to the same Web site, your browser will send the cookie to the Web server. The server can use this information to present you with custom Web pages. For example, instead of seeing just a generic welcome page, you might see a welcome page with your name on it.

How do Servlets create and set cookie data?

After a cookie is created, it can be set by using the addCookie method in HttpServletResponse.

```
// response is an HttpServletResponse that is passed to the Servlet

Cookie cookie =new Cookie("cookiename", "cookievalue");
response.addCookie(cookie);
```

The Servlet can call any of the public methods in javax.servlet.http.Cookie on the Cookie object.

How do Servlets access cookie data?

After a cookie is set, when the next request from the same client comes in, the cookie can be fetched by using the getCookies method in HttpServletRequest.

// request is an HttpServletResponse that is passed to the Servlet

Cookie [] cookies = request. getCookies();

The Servlet can call any of the public methods in javax.servlet.http.Cookie on the Cookie object.

Chapter 5 Web Server Implementation

5.1 Project Overview

5.1.1 Objective:

In order to understand Web server and HTTP better, I implemented a pure Java Web server — the Abao Web Server (AWS). The objective of the AWS project is to develop a commercial-grade, full Java compliant, easy to use and easy to configure, open-source Web server.

5.1.2 Releases

The project began in January 2000; the first Beta release of AWS was available in middle of May 2000. Version 1.0 was released in late August and in November 2000 the latest release of AWS, version 1.5 was released. This version has enhanced support for Servlets and sessions. It fully supports most HTTP 1.1 features: persistent connection, CGI, Servlet, etc. It also has a GUI when running on the Windows platform to make user configuration easier. AWS does the same job as Apache + Jserv or Apache + Jakarta Tomcat. It has a build-in Servlet container that can speed up the Servlet response.

5.1.3 Java Support

This package is done with JDK 1.3. It uses and supports the Java Servlet development kit

(JSDK) 2.1 Standard. Documentation and Specifications of JDK 1.3 and JSDK 2.1 can be found on the Java's home page on Sun Microsystem's Web site. Because it uses pure Java, AWS can be used on any platform. It has been tested on two platforms: Windows 95/NT and Unix/Linux.

5.1.4 Installation and configuration

The completed package is about 150K in size. The deployment package contains only 4 files: 1 .jar file for complied source and library, and three .conf files for configurations. The .conf file follows the same rule as Apache .conf files. It is very simple to install and configure. Details for compiling, installation and configuration are in Appendix B AWS installation and configuration manual).

5.2 Project UML and Package structure

5.2.1 Package com.chinapromotion.aws

Figure 9 shows the structure of the AWS project. The AWS class and all other packages shown on this diagram is under the package com.chinapromotion.aws. The name of package follows the general rule of com + company name + project name. China Promotion is a company I registered in Hong Kong. The source of this project and future projects will be supplied as open source projects on the company's Web site once I go back to Hong Kong.

In this diagram, we can see one class and six packages. The AWS class, which is the main class of the project, controls the initialization of program and loading of configuration. It responsible for restarting and shutdown the program, and also contains the implementation of main class, which will parse the command line parameter. The only command line parameter AWS supports now is "-c" parameter, which gives out the path of the installation path of the AWS project. This parameter is later used to load configuration files and log any event and error to a log file. The program will exit and the usage function will be called to print out the correct format of command line parameter whenever there is an error while parsing the command line parameter.

Figure 9 UML diagram: Package com.chinapromotion.aws

5.2.2 Package Servletmanagement

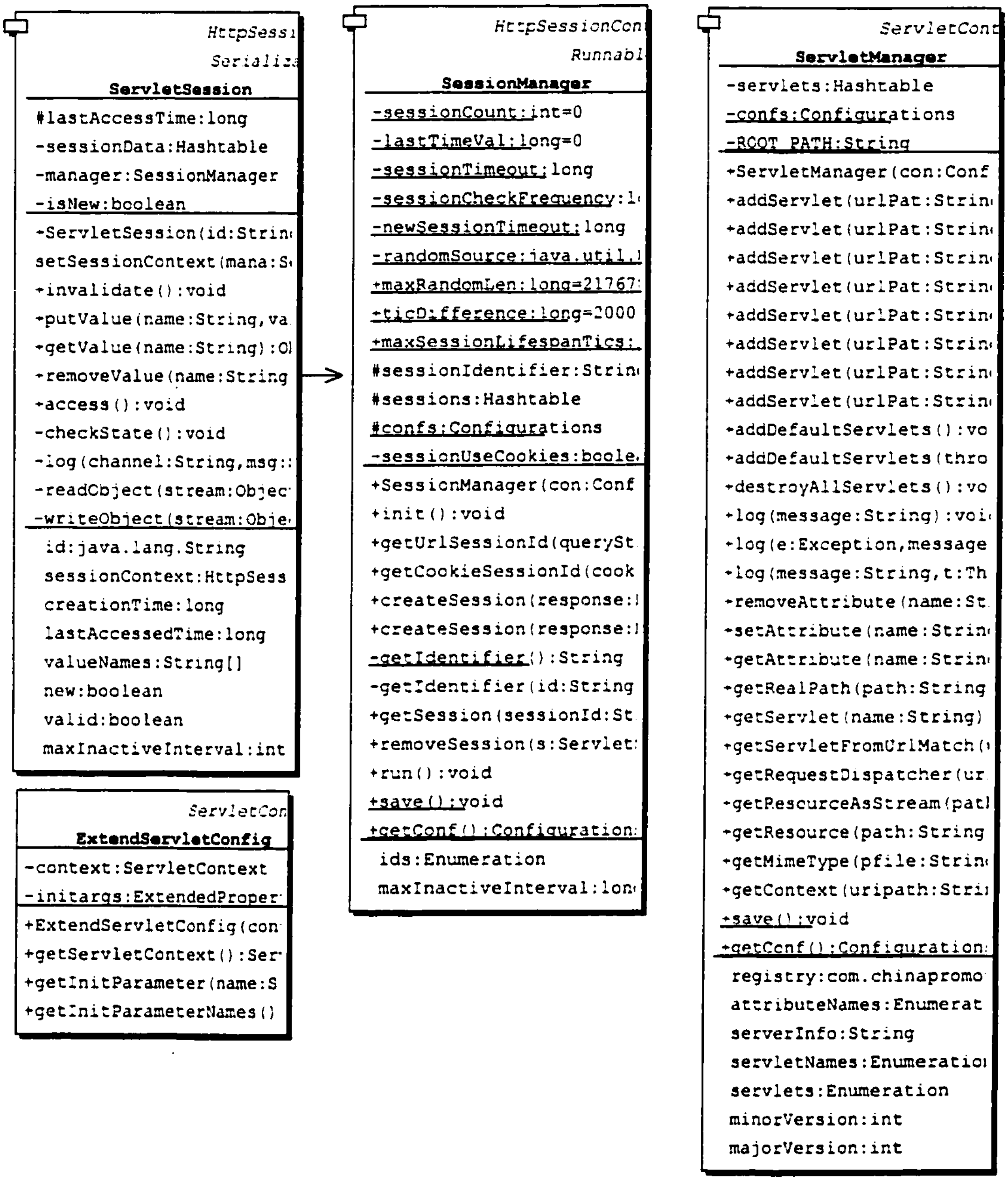

Figure 10 UML diagram: Package Servletmanagement

As Figure 10 showes, the servletmanagement package contains four classes:This package is responsible for management of the session and the Servlet container.

ExtendServletConfig is a class that extends the standard JSDK ServletConfig class. It contains information and initial parameters about the Servlet.

The ServletManager class is the class that manages all Servlets. It loads and initializes all the Servlets when the Web server starts. Information about these Servlets is stored inside the servlets.conf file. Usually two parameters about a Servlet are required: the Servlet class name and the Servlet URL pattern. A Servlet class must be a subclass of the HttpServlet class. It is a standard Java Servlet API class. A Servlet URL pattern is used to match the request URL to the particular Servlet. The ServletManager class maintains a wildcard dictionary that maps Servlets' URL pattern to its initialized instance. The URL pattern is the key to this dictionary. It can be a wildcard that allows the user to use "*" and "?" to replace all strings and characters. A utility function inside the Utility package can match an input wildcard string with a wildcard pattern. When an HTTP request comes in, the server will call the utility function to match the incoming URL path with all registered Servlets. If a match is found, this means it is a Servlet request. The corresponding Servlet is called to handle the incoming request.

The ServletSession class is the class that extends standard HTTPSession from Java Servlet API. It maintains a hashtable to keep all objects related to the session. After a Servlet gets a ServletSession object, it can get and put objects into the session. It also contains function to create and invalidate a session.

Session Manager is the class that manages all the sessions created by servlets. The AWS class contains a private instance of the session manager. It is initialized when the AWS class starts; it is shut down when the AWS class shuts down. The init function of this class loads all parameter from the session configuration object, which is already loaded from the Session.conf file by the AWS class. The save method is called when AWS is shutting down or restarting. This method saves back to the Session.conf file all parameters that might be changed during the run. It implements the HttpSessionContext interface that is defined in the JSDK 2.1 API. It also has several createSession methods for creating a ServletSession object with a different input parameter. ServletSession is the object that stores the actual data of a servlet session. A Hashtable is maintained to keep a reference to all created ServletSession objects. The SessionManger class has its own housekeeping thread that uses the **session.checkFrequency** parameter to detect session timeouts. The value stored in the **session.checkFrequency** parameter is the activation frequency of the housekeeping thread, which checks the validation of all existing sessions stored in the Hashtable.

5.2.3 Package com.chinapromotion.aws.Util

Figure 11 shows the com.chinapromotion.aws.Util package, This package contains several utility classes and classes for configurations.ConfigurationRepository, Configuration and ExtendedProperties are three classes used to manage configuration files. AWS supports the same format of configuration files as Apache. Three configuration files are used by AWS: one for general setting, one for Servlets settings and one for sessions. These files are text files; each line of the file represents a directive. The format of each line is directive name = directive value. These parameters are used to change some variable without recompiling the code. Configuration files are loaded when AWS starts, and it can be modified by a text editor or through the GUI of AWS under Windows. The wildcard dictionary is a utility class that extends the Dictionary class; it supports the '*' and '?' as wildcard references to the dictionary. CookieUtils contains utility functions on cookie manipulation, such as a function for parsing cookie headers.

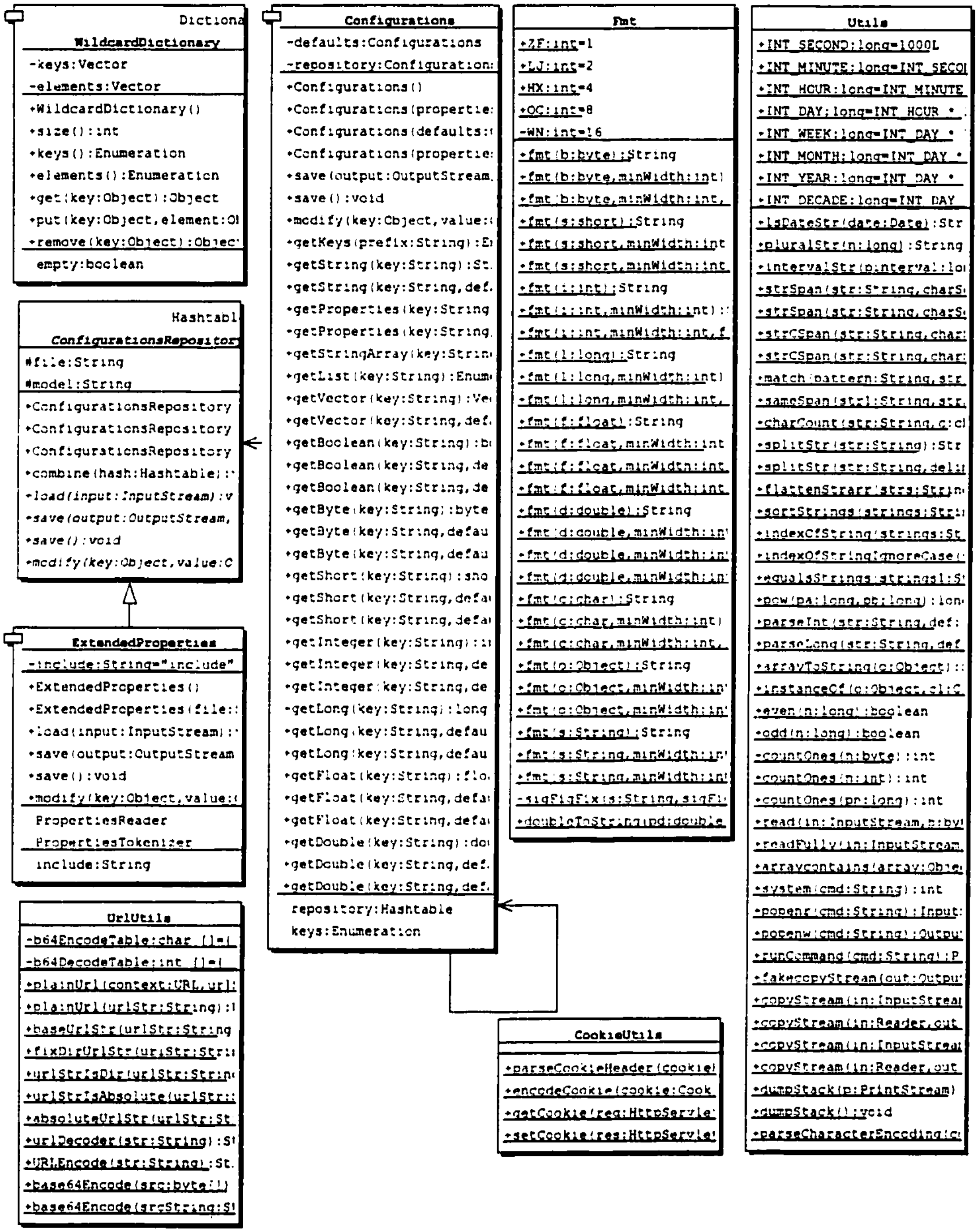

Figure 11 UML diagram: Package com.chinapromotion.aws.Util

5.2.4 Package com.chinapromotion.aws.servlets

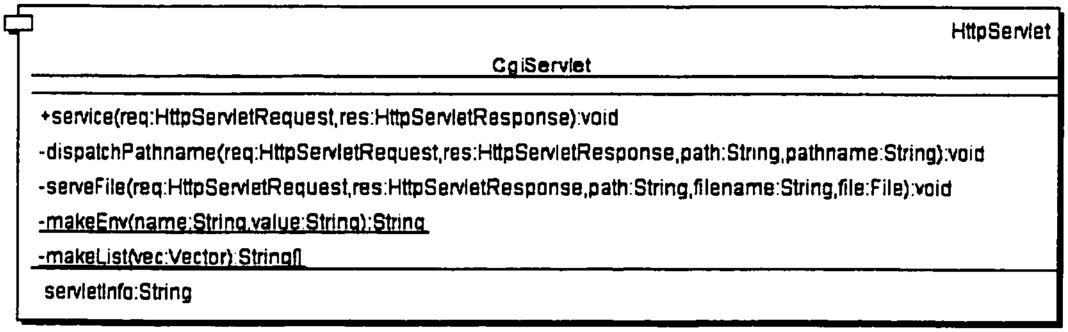

Figure 12 UML diagram: Package com.chinapromotion.aws.servlets

The Server.Servlet package contains only one class CgiServlet. It is the only default Servlet deployed with the release V1.5. This class handles all CGI requests. It uses the URL pattern *.cgi, which means all requests ending in .cgi will be handled by this Servlet.

This package is also the place to put all later implementation of other default Servlets, such as JSP supporting Servlet.

5.2.5 Package com.chinapromotion.aws.service

As Figure 13 shows, the com.chinapromotion.aws.service package contains the scaling module of AWS. There are two kinds of services that AWS could provide. LocalService class implements the local service that parses and handles incoming requests on a local machine. RemoteService class implements the remote service that handles an incoming request on a slave machine. Both service classes implements the ConnnectionService interface, which has an abstract method named startService. Once a slave machine is registered, a new RemoteService object is initialized and added to the service list that the ServiceManager object maintains. When a request comes in, with a slave assignment method, the ServiceManager object decides if a local service or remote service will be invoked. Then the startService of the assigned service object will be called to handle the incoming request. Details of the scaling module can be found in Chapter 8.

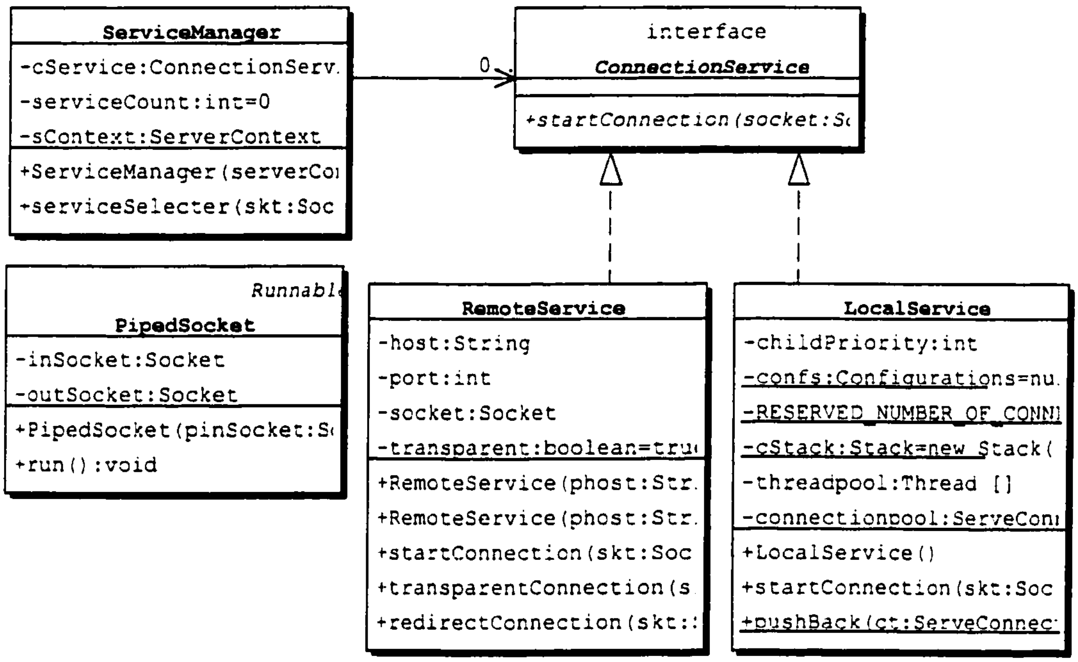

Figure 13 UML diagram: Package com.chinapromotion.aws.service

5.3 Web Server Execution Flow

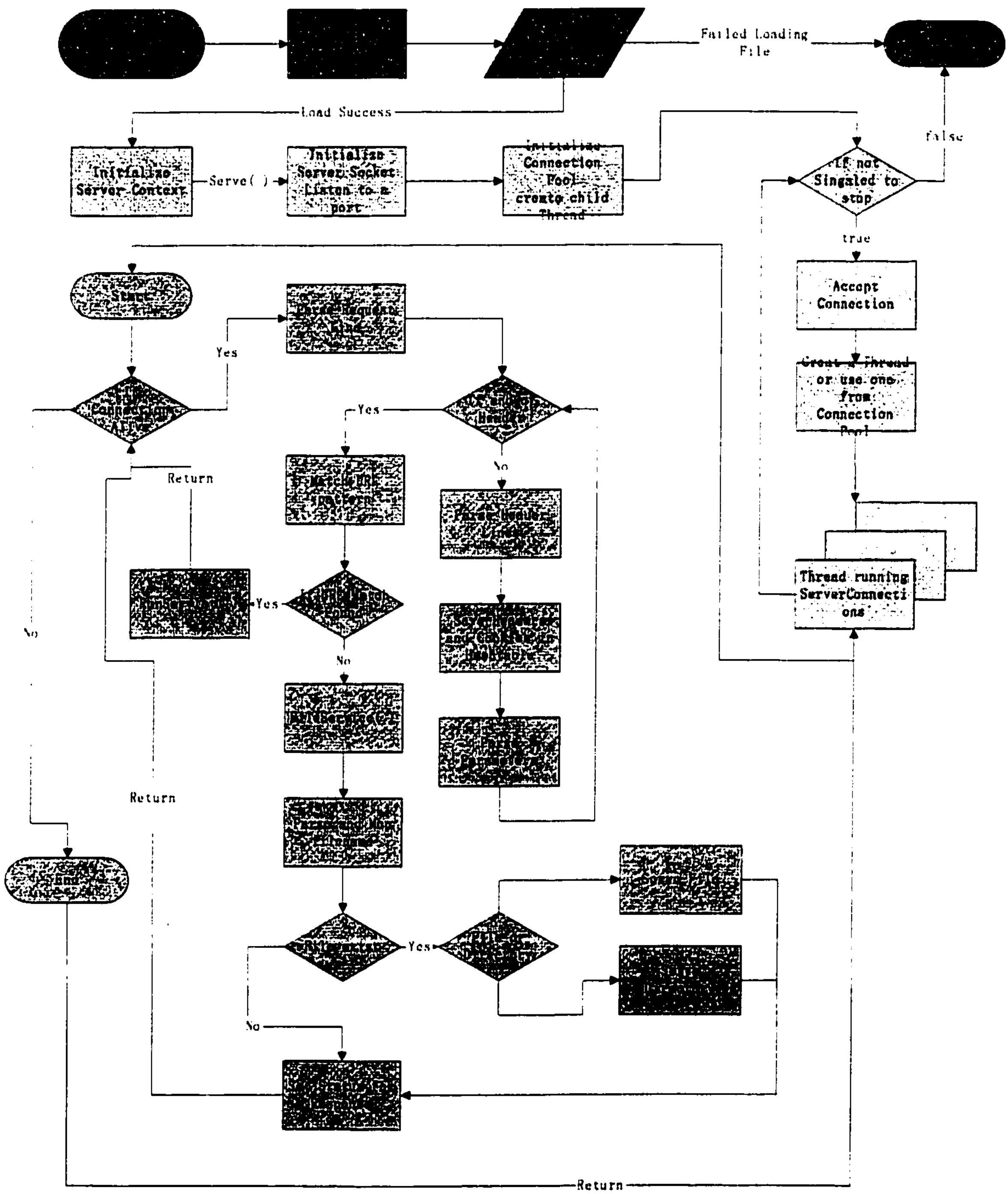

Figure 14 Project Flow Chart

5.3.2 Web Server Flow Chart

Under the Server directory, there are three files: AWS.java, ServerContext.java, and ServerConnection.java. These three classes make up of the whole process of a Web server. Figure 14 illustraes the flow chart of that process, with the three classes shown in different colors.

AWS.java

AWS is the main class that runs when the program starts. It parses the command line arguments and initializes all the variables. There is only one command line argument in AWS release 1.5, "-c", which indicate the configuration file path. This argument is used if the configuration file is not in the default location. Then AWS loads the configuration file. If the configuration file is not found or contains errors, AWS will exit the program. After successfully loading the configuration file, AWS initializes the ServerContext object and calls the server( ) function of that object.

ServerContext.java

ServerContext is responsible for accepting and handling socket connections. It maintains a connection pool if the connection pool size directive is set in the configuration file. Every ServerConnection object is bound to a thread. For all pool connections, both the ServerConnection object and the Thread object are initialized and kept in a stack. Therefore when a new connection comes in, this class will check if there is a connection available in the stack. If the stack is not empty, a connection thread is popped out of the stack and been assigned to handle the incoming connection. If the stack is empty, a new ServerConnection object and Thread object will be created to handle the connection. All pool connections are recycled after completing the response; they will be pushed back to the connection stack. All other connections and threads will be destroyed and collected by the garbage collector. Once a ServerConnection is in a service state, the thread which hasbeen bound to it will take charge of the later process and return control to the main thread. The main thread will then free to handle the next incoming connection. This accept–service-return process will loop until the AWS server is instructed to shut down. Priority of the main thread and all child threads can be set up as a directive in the AWS.conf file.

ServerConnection.java

Every ServerConnection object is bound to a Thread object. Threads are used to simultaneously handle multiple connections. Every ServerConnection handles a socket connection, reads input from the input stream of the socket and writes output to the output stream of the socket. The ServerConnection object also handles persistent connection: once a connection is established, it will loop forever unless the connection is closed by the client or the client uses the header:

Connection: Close

The client can pipeline requests to speed up the process. For every request, the following steps are performed:

Step 1: Parsing of the request line.

The HTTP request line format is shown in Section 3.3.5 (HTTP Request). AWS fine-tunes the parsing of the request line by pushing the error handling and parsing process down to the stream reading phase. The URL line parameter is also parsed and stored in vectors.

Step 2: Parsing of the request headers.

Since request headers have an unknown number of lines, an empty line indicates the end of headers. This process loops until an empty line is read. Details of the request header can be found in Section 3.3.6 (HTTP Response). As in Step 1, this process is also pushed down to the stream reading phase. A special inner class ServeInputStream is designed to optimize the performance. This process stores the request method and request URL path for later usage.

Step 3: Parse parameters and cookies

There are two ways that the client can use to pass parameters to the server. First, client can append "name=value" pairs in URL after a '?' symbol. For example:

http://www.samplesite.com/counter.cgi?color=red&font=large

Second, if a Content-Length header is read from step one, which indicates that the incoming request line comes with some parameter, these parameters are in name = value format. These name and value pairs are read, parsed, and stored as two vectors, one for all the names and another for all the values. These parameters will be merged with the URL line parameters if necessary. The two methods of passing parameter, through URL line and through Request body, are treated identically in HTTP. Session Information and cookies are also stored as the format of parameter. Sessions and cookies are discussed in Section 4.2.

Step 4: Matching the URL pattern

After all headers and parameters are read, a utility function will be called to determine if the income request is a Servlet request. For instance, in AWS implementation, a default Servlet — CGIServlet — will handle all CGI request. This Servlet uses the URL pattern *.cgi. Therefore, the CGIServlet will be called to handle all request URL paths that end with .cgi. If no matches are found (which means it is a file request), the process will continue to handle it as a file request. If a match is found, the Servlet which registered this particular URL pattern will be activated by calling the Service( ) function in the HTTPServlet interface.

Step 5: File Service

In a classical Web server design, the file service of the Server should be in a module separate from the module containing the ServerConnection object. However, AWS integrates the file service module with the ServerConnection module. Because most requests are file requests, putting the file service module in the ServerConnection class can improve performance by saving a lot of function calls and passing of parameters. This placement will also improve the later implementation of file caching. In this procedure, a virtual URL path is mapped into a real path on the server. At this point, there are two cases and an exception to be considered. The two normal cases are those in which the request URL path is a file or is a directory. The exception is when the request URL path does not exist. In the configuration file, a directive called Default Filename is used when a directory is found and, in this situation, the file service appends the default filename to the end of the directory. Since this directive allows multiple filenames, the program tries file names one by one until a default file is found. If a default file is found, that page will be sent to the client. Otherwise, if the Directory Indexing directive is enabled, a Directory Indexing service is called to list all files in the directory.

Step 6: Write response to the client

A response status is sent to the client; the response status code can be found in Section 3.3.6 (Status Code and Reason Phrase). Then a few response headers are sent. If a persistent connection is used, a Content-Length header is sent to give the client information about the page size. Then, after all headers and an empty line, the requested page is sent to the client as requested.

AWS also implements many features that either optimize the overall performance or support some new functionality.

5.3.3 Client Side Caching

Client side caching is used when handling file requests. If a page is already in the client's cache, when the browser sends the request, a Last-Modified-Since Header is sent with the request. The File Service procedure in ServerConnection object first locates the requested file on local file system. The modified time is checked with the timestamp obtained from the client request header. If the file has not been modified since the time of the client's request, a 304 (No Modified) status and no body content of this file will be sent. This feature is done inside the ServerConnection object.

5.3.4 Servlet Management implementation

The ServletManager, which initializes when the AWS main class initializes, manages all the Servlets in the serveltmanagement package. The ServletManager reads Servlet registration information from the Servlet.conf file, load and initializes all the Servlets and registers their URL pattern in a wildcard dictionary. This dictionary is used by the URL pattern matching process of the ServerConnection object.

5.3.5 Session Management implementation

The SessionManager object, which initializes when the AWS main class initializes, manages the session in the serveltmanagement package. The SessionManager reads session registration information from the Session.conf file. It maintains all sessions in a hashtable used by those Servlets that request a session. We will discuss details of the Servlet and session management in the following chapter.

Chapter 6 Servlet Container Implementation

6.1 Servlet Implementation Basic

A Servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the Hypertext Transfer Protocol. To implement this interface, Java programmers can write a generic Servlet that extends javax.servlet.GenericServlet or an HTTP Servlet that extends javax.servlet.http.HttpServlet.

This interface defines methods to initialize a Servlet, to service requests, and to remove a Servlet from the server. These are known as life-cycle methods and are called in the following sequence:

1. The Servlet is created, and then initialized with the init method.

2. Any calls from clients to the service method are handled.

3. The Servlet is destroyed with the destroy method, and then finalized and garbage collected.

In addition to the life-cycle methods, this interface provides the getServletConfig method, which the Servlet can use to get any startup information, and the getServletInfo method, which allows the Servlet to return basic information about itself, such as author, version, and copyright.

6.1.1 GenericServlet

GenericServlet is an abstract class, which defines a generic, protocol-independent Servlet. To write an HTTP Servlet to use with a Web site, HttpServlet must be extended. GenericServlet implements the Servlet and ServletConfig interfaces. A Servlet usually extend GenericServlet or its subclass HttpServlet, unless the Servlet needs another superclass. If a Servlet needs to extend a class other than GenericServlet or HttpServlet, the Servlet must implement the Servlet interface directly [11].

GenericServlet makes writing Servlets easier. It provides simple versions of the lifecycle methods "init" and "destroy" and of the methods in the ServletConfig interface. GenericServlet also implements the log method, declared in the ServletContext interface.

To write a generic Servlet, it is necessary only to override the service method, which is declared as an abstract method with no body. When writing a Servlet container, it should override getServletInfo and specialize the init and destroy methods if the engine will manage expensive Servlet-wide resources.

6.1.2 HttpServlet

HTTP Servlet provides an abstract class that users can subclass to create an HTTP Servlet, which receives requests from, and sends responses to, a Web site. When users subclass an HttpServlet, they must override at least one method, usually one of these:

- doGet, if the Servlet supports HTTP GET requests
- doPost, for HTTP POST requests
- doPut, for HTTP PUT requests

- doDelete, for HTTP DELETE requests
- init and destroy, which you override as a pair if you need to manage resources that are held for the life of the Servlet
- getServletInfo, which the Servlet uses to provide information about itself

The service method in HTTP Servlets usually does not need to be overridden. Service handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the do*xxx* methods listed above).

Likewise, doOptions and doTrace methods usually also do not need to be overridden. The service method supports HTTP 1.1 TRACE and OPTIONS requests by dispatching them to doTrace and doOptions.

Servlets typically run on multithreaded servers, so a Servlet must be prepared to handle concurrent requests and synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections. See the Java Tutorial on Multithreaded Programming on Sun's Web site for more information on handling multiple threads in a Java program.

6.2 Handle HTTP Servlet Request and Response

The service method in HTTP Servlets requests two parameters, an HttpServletRequest and an HttpServletResponse. These two parameters are interfaces defined by JSDK API. In the AWS project, the ServerConnection object implements both these methods. HttpServletRequest contains all information about the HTTP request, which can be used by the Servlet through a pre-defined method of this interface. HttpServletResponse contains information about the Http response.

Figure 15 shows the relationship among HttpServletRequest, HttpServletResponse, Servlet and client. In AWS implementation, the ServerConnection class implements both HttpServletRequest and HttpServletResponse interfaces. It is a thread that will call the service function of the Servlet to handle a Servlet request. When the Servlet needs information about the Request, it calls the getReader and/or other functions of the HttpServletRequest interface. When the Servlet needs to write information to the Response, it calls the getWriter and/or other functions of the HttpServletResponse. Reader and Writer are then mapped to Input and Output Streams of the socket connection. In this way a Servlet can communicate with the client through a Servlet container, which in the case of AWS, is the ServerConnection object.

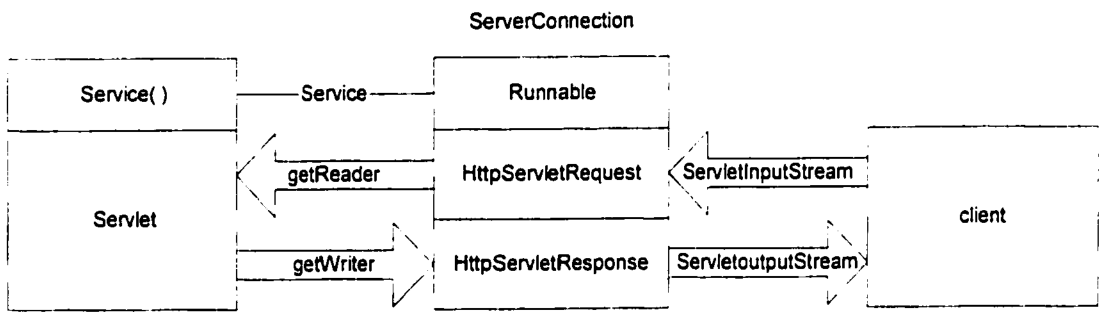

Figure 15 HttpServletRequest and HttpServletResponse

A few useful methods in HttpServletRequest API are:

getCookies:

Returns an array containing all of the Cookie objects the browser sent with this request. This method returns null if the browser did not send any cookies.

getHeader:

Returns the value of the specified request header as a String. If the named header wasn't sent with the request, this method returns null. The header name is case insensitive. You can use this method with any request header.

getMethod:

Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.

GetRequestURI:

Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.

getSession:

Returns the current session associated with this request, or if the request does not have a session, creates one.

getReader:

Returns the body of the request as a BufferedReader that translates character set encodings.

A few useful methods in HttpServletResponse API are:

addCookie:

Adds the specified cookie to the response. It can be called multiple times to set more than one cookie.

sendError:

Sends an error response to the client using the specified status code and descriptive message. If setStatus has previously been called, it is reset to the error status code. The message is sent as the body of an HTML page, which is returned to the user to describe the problem. The page is sent with a default HTML header; the message is enclosed in simple body tags (<body></body>).

setHeader:

Adds a field to the response header with the given name and value. If the field had already been set, the new value overwrites the previous one. The containsHeader method can be used to test for the presence of a header before setting its value.

setStatus:

Sets the status code for this response. This method is used to set the return status code when there is no error (for example, for the status codes SC_OK or SC_MOVED_TEMPORARILY). If there is an error, the sendError method should be used instead.

getWriter:

Returns a PrintWriter object that you can use to send character text to the client.

6.3 State Management of AWS

6.3.1 Cookie:

AWS supports cookies by using two data structure in ServerConnection class. One is an array of Cookie objects that stores incoming cookies from the client; another is a vector of cookies that stores outgoing cookies from Servlets and other server side programs. AWS uses different structures for incoming and outgoing cookies because it knows the number of incoming cookies but not the number of outgoing cookies. As long as the request-response process is not finished. the number of cookies from the server side is not determined. New cookies may be added to the outgoing cookie vector at anytime by a server-side program. Therefore using an array for incoming cookies and a vector for outgoing cookies is more efficient in respect to both memory and performance.

Incoming cookies are received in terms of cookie header. The cookie header has the following format:

Cookie: name=value;name=value;name=value

This string is parsed by the parseCookieHeader method in the Util class. This method parses the cookie header into a array of Cookie objects. Every Cookie object inside the array contains a name=value pair from the cookie header. This array is then stored as a private member of the ServerConnection class and is available to all Servlets through the standard getCookie method of JSDK 2.1 API.

When the Servlet needs to send the client a cookie, the cookie is added to the outgoing cookie vector through the setCookie method of JSDK API. This vector is kept in the ServerConnection class until headers are sent. At that time, a Set-Cookie header is generated and all name=value pairs inside the cookie vector are appended to the header. Then the header, together with other headers, is sent to the client. If the client receives the cookie header, cookies are parsed and stored on the client side. The cookie will be sent back to the server when client next requests the same URL.

6.3.2 Session:

A session is more a complex technique than a cookie. Cookies still use the stateless HTTP protocol and all state information is stored on the client side as text files. A session stores state information on the server side, which allows server side programs to store more complex data types in a session. A typical session may have a database connection object attached, which is very hard to be serialized into a string object that might be used by cookie. Therefore the implementation of session handling requires a server side data structure to manage all client sessions. The session manager object inside the servletmanagement package manages a hashtable that stores all session objects with the session ID as a key. Session ID is generated by applying the current time and IP address to a special algorithm: every session ID is unique.

For a typical client session, all user-state objects are kept in a session object; the API of session object is defined by the JSDK standard. The session manager uses the session ID to hash all session objects into a hashtable. Each session object maintains a hashtable to keep a reference to all user-state objects. The session object has a timeout value that can be setup by the session.timeout parameter in Session.conf file. This value represents the number of milliseconds that elapse (during which no request on that session occurs) before the session times out. In AWS implementation, a housekeeping thread is used to do session-validation checks at frequent intervals. The frequency of validation checks can be defined through the session.checkFrequency directive in Session.conf file.

Once a session is created by any Servlet, a session ID is generated and passed on to the client in the form of URL line parameter or cookie. The client will send this session ID back to the server when requesting the same URL. The server receives the session ID and can use it as a key to fetch a session from the hashtable, which is managed by the session manager object. Whenever a Servlet makes a getSession request, the found session will be returned. The Servlet can then access any object associated with the client's previous session.

Chapter 7 Graphical User Interface For Windows

7.1 General Features and Usages

Once the program executes, AWS displays the main window (Figure 16). The main window is a tabbed panel where users can choose the option. "Connection" is the default option panel, which contains information about the port number that AWS is listening; the maximum number of connections allowed, the connection pool size, the main thread priority and priority for child threads.

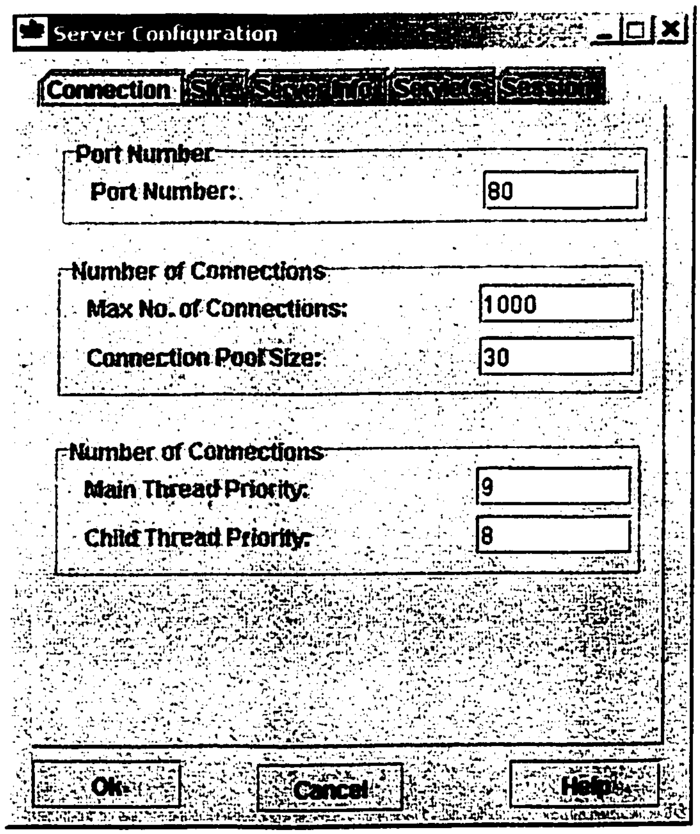

Figure 16 The connection properties tab

Buttons

As Figure 16 illustrates, there are three buttons at the bottom of the main window: OK, Cancel and Help.

Button "OK"

The "OK" button is used to save the server parameter values in the boxes. After changing the values of any boxes, the user presses the "OK" button to save the values. If the user modifies the values of a parameter text box and does not click "OK", the values that the user has entered will not be saved, even though the users can see changes of the related text boxes.

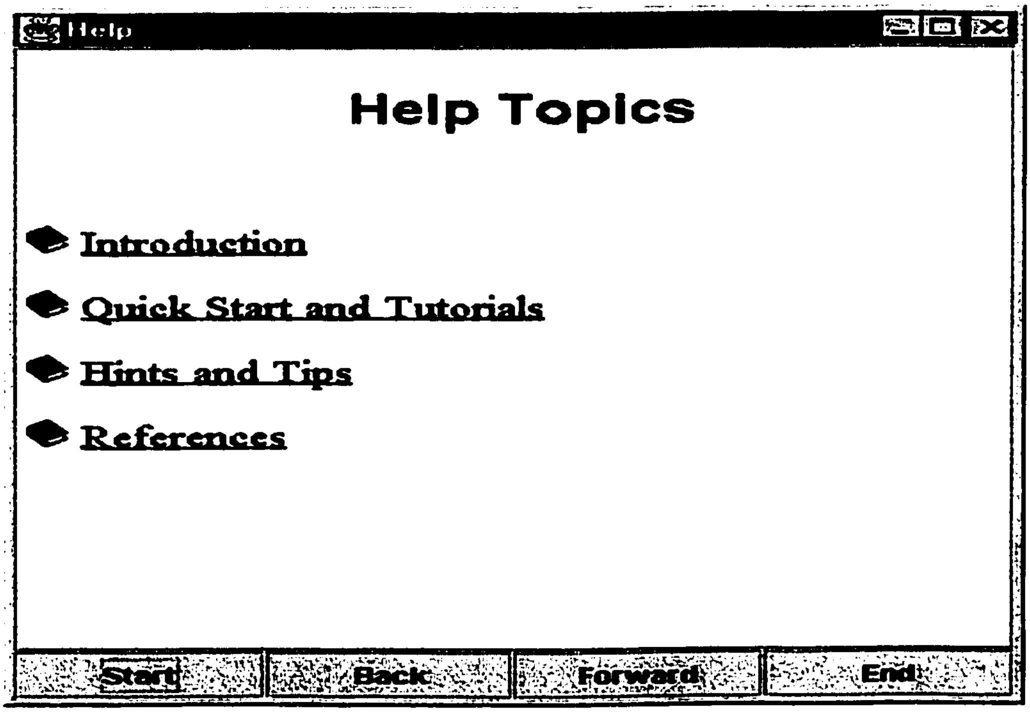

Figure 17 The Help Window

Button "Cancel"

After users have changed the values in the boxes, they can cancel, that is, not save the change by pressing the "Cancel" button.

Note: the "Cancel" button is disabled if nothing is changed in any box, but enabled if any change has been made in the text boxes.

Button "Help"

The "Help" button is used to help familiarize users with this program. After pressing the "Help" button, a help window is displayed as Figure 17.

From the help window, users can choose available help pages by clicking the button "Start", "Back", "Forward", and "End". Users can also click the links in each page to get the help they require.

Close the Main Window

By clicking the closing button of the main window, it will be minimized to the system tray as an icon, shown in Figure 18.

The tool tip of the icon indicates that the server is running (the icon is a red maple leaf).

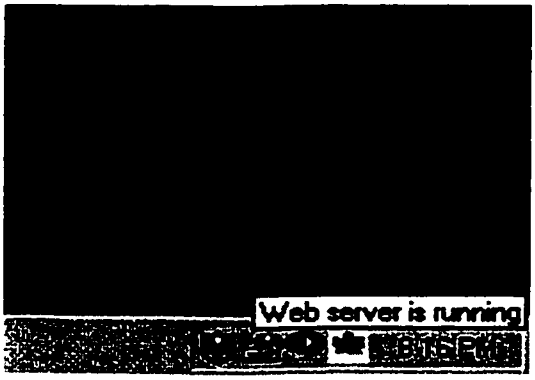

Figure 18 Tool tip of tray icon

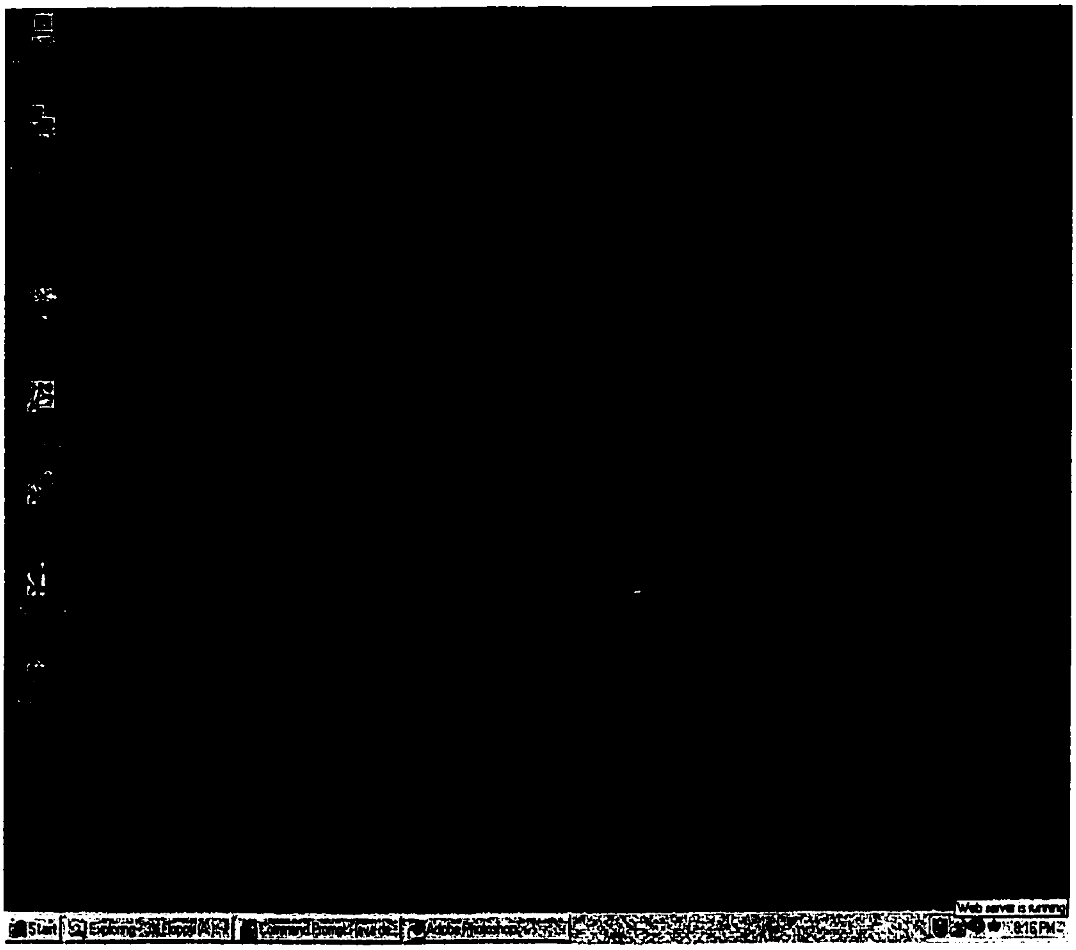

Figure 19 The tray icon in system window

By clicking the right mouse button on the tray icon, a menu will pop up, as Figure 20

Figure 20 The Pop Menu and the About Window

From the pop menu,

- **Show**: The show option restores the main window.

- **About**: The About window appears, as in Figure 20;

- **Restart AWS:** The server will restart and load new settings.

- **Exit**: The server will exit execution.

7.2 Connection Tab

As shown in 7.1Figure 16, all parameters and corresponding values are displayed in this window. Users can modify these parameters by changing the values in the corresponding text fields. The Connection Tab contains settings about the connection.

Port Number

"Port Number" is the number of the port that the server is currently using to listen. A valid value for Port Number is an unsigned positive integer. Characters cannot be entered or displayed. Nothing appears in this box if users enter characters.

Number of Connections

"Max Connections" refers to the maximum number of connections the server is allowed and the "Connection Pool Size" means the number of connections that have been reserved.

Valid values of "Max Connections" and "Reserved Connections" are unsigned integers. Characters cannot be entered or displayed. The value in the box of Reserved Connections box should always be less than the value in the Max

Connections box.

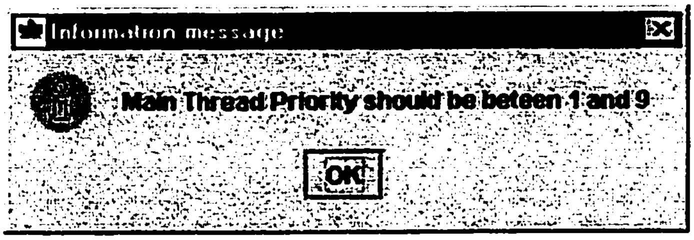

Figure 21 Warning message for thread priority

Thread Priority

Thread Priority is used to assign a priority value to threads. The valid values of both main threads and child threads are 1 to 9. If users enter a character, there will be no response in the boxes. If users enter a number less than 1 or greater than 9, for example, -1 or 10, a warning message window will pop up (Figure 21)

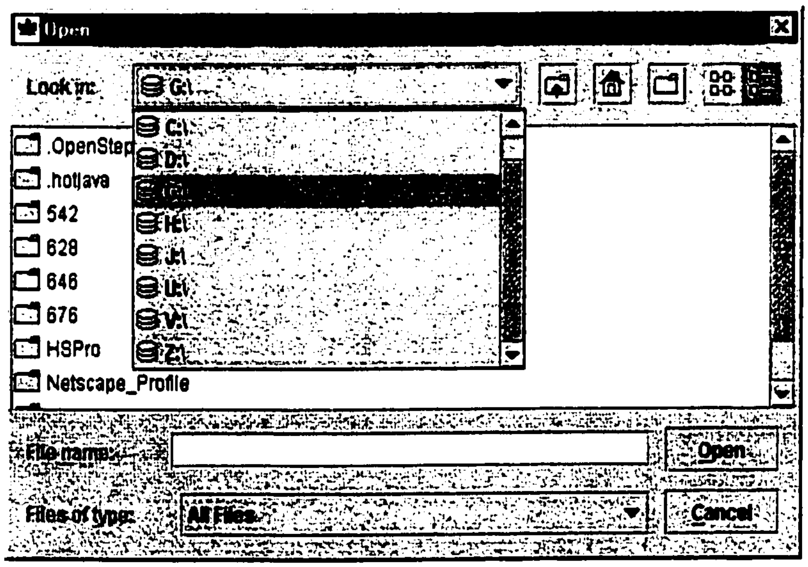

Figure 22 File Chooser

7.3 Site Tab

As Figure 23 shows, the site properties tab contain the following settings:

Host Name

Host Name is the IP address of this server. The default value is displayed in the box and users can change it according to their needs.

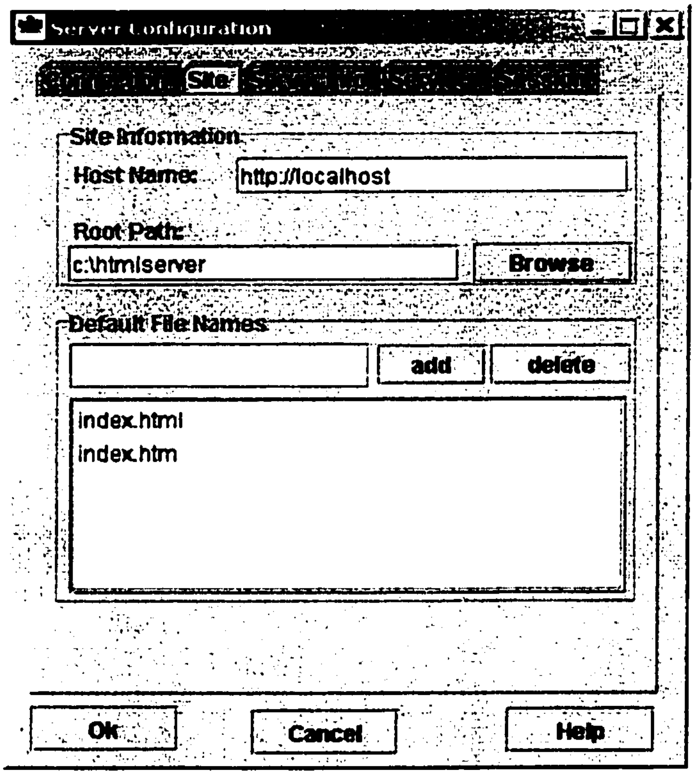

Figure 23 Site properties tab

Root Path

A Root Path can be selected through a file chooser. It is the directory where, mapped as "/", the root directory of this Web site is located. For example, if a root path is set to be "c:\htmlserver", then the HTTP request "get /index.html" is mapped to "c:\htmlserver\index.html".

Default Filename

Default filename is a collection of ordered filenames that is used when a directory request is received. For example, if two default filenames are registered, "index.htm" and "index.html", when a request "get /" comes in, "/index.htm" is checked. If no file is found, the next filename, "/index.html" is tried.

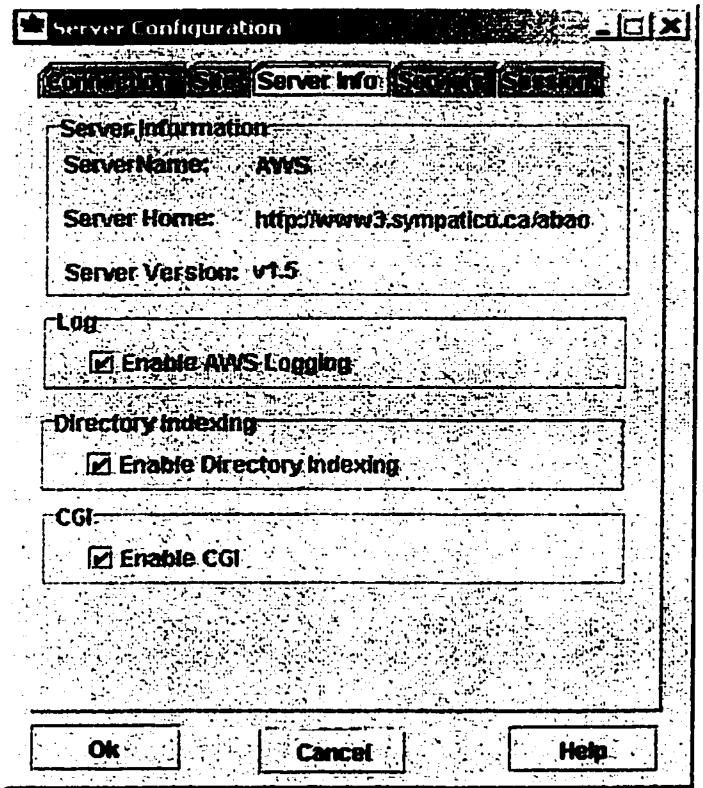

Figure 24 Server Info Tab

7.4 Server Info Tab

Logging

As shown in 7.3Figure 24, the logging check box represents a boolean value determining whether or not to enable system logging. If logging is enabled, errors and exceptions will be written to a text file under the log directory of the AWS installation directory. Logging requires both CPU and IO resources and will reduce the overall Web server performance.

Directory Indexing

If directory indexing is enabled, AWS will generate a directory indexing when no default files are found for a directory request.

CGI

When CGI is enabled, the cgi Servlet is loaded and ready to handle CGI request. Otherwise, CGI requests will be treated as normal file requests.

7.5 Servlets Tab

As Figure 25 shows, the Servlets tab contains all information needed to register a Servlet on the AWS server.

URL Pattern

This is a wildcard pattern that maps an incoming URL to the Servlet, i.e. *.CGI will map to the CGI Servlet all URL ending with ".cgi".

Servlet Class Name

This field is for the class name of the Servlet, for example, com.chinapromotion.AWS.CGIservlet. In order to run a Servlet on AWS, users must to ensure that the path of the Servlet class is included in the classpath environment variable.

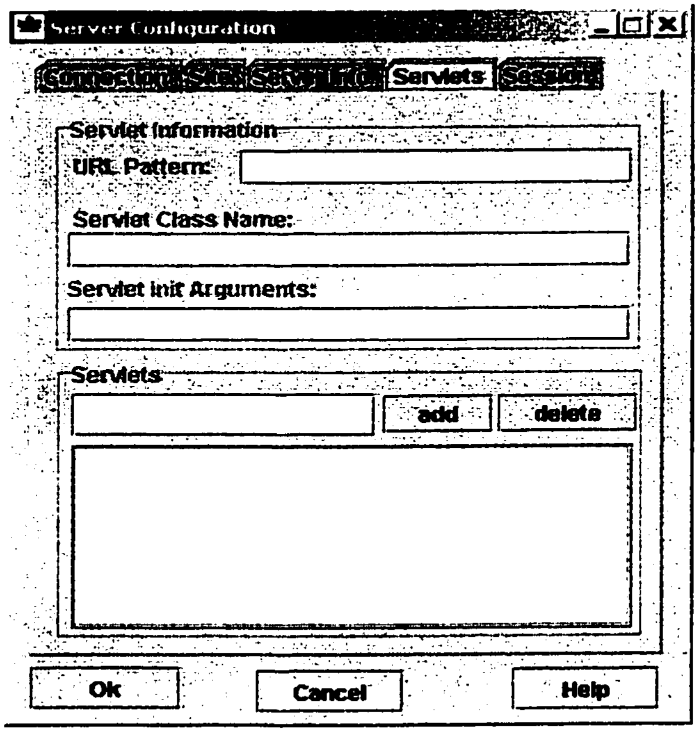

Figure 25 Servlets Tab

Servlet Initialize Argument

Here the user can input the initializing argument for the particular Servlet.

Servlets

This is where the user can select, add, and delete Servlets. After a Servlet is selected. the user can modify its URL pattern, classname, and initialize arguments.

7.6 Session Tab

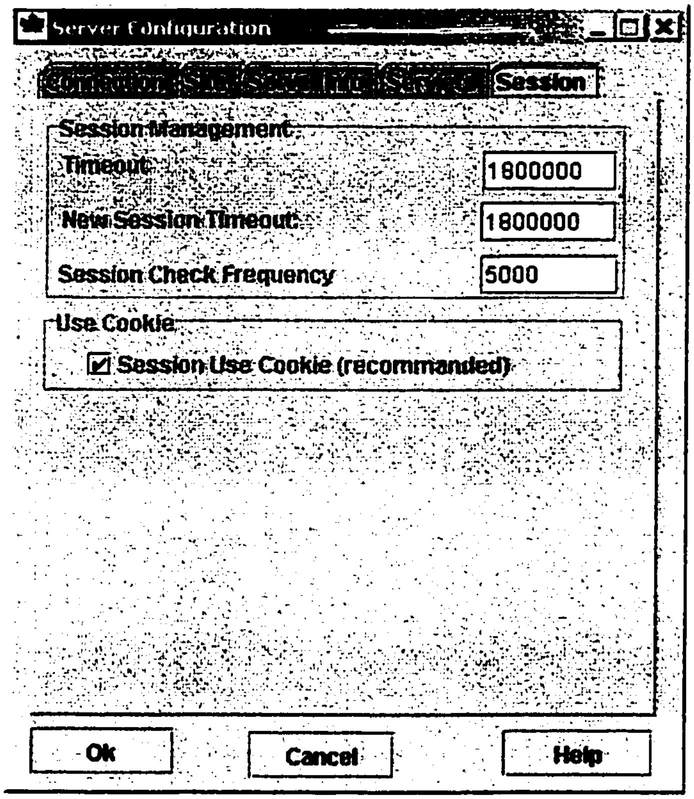

Figure 26 Session tab

Timeout

This integer value indicates the number of seconds until the session times out.

New Session Timeout

This integer value indicates the number of seconds until a new session times out. A new session is a session that is created but not accessed.

Session Check Frequency

This determines the number of seconds that the housekeeping thread of the session manager object is scheduled to run. Every time the housekeeping thread runs, it traverses all the sessions, discovering any invalid sessions and garbage collecting them. The housekeeping thread consumes system resources. If its frequency is set too high, it will lower the system's performance. If its frequency is set too low, a session may remain valid for a long time after it had timed out.

Chapter 8 Scalability Model

8.1 Transparent Scaling

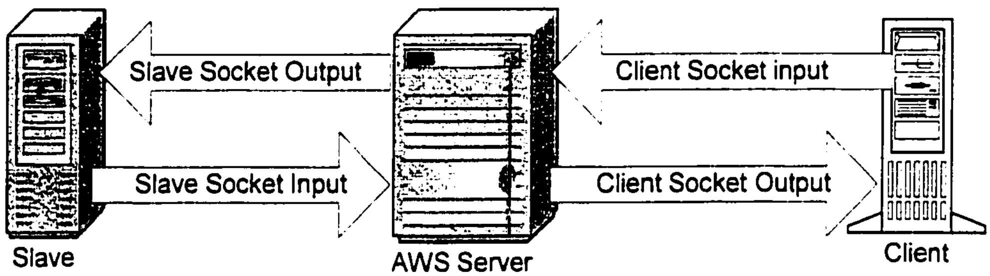

Figure 27 Transparent Scaling

With respect to transparent scaling, the AWS server acts like a proxy: the AWS server is "transparent" to clients. This means that the actual request from the client is sent to a slave through another socket connection, and the response from the slave machine is received by the AWS server and sent back to the client. There can be many slave machines registered on the AWS server and each of them can have a Web server running. Whenever a client connection comes in, the AWS server will assigns a slave machine to handle the incoming request. It establishes a socket connection to the assigned slave machine. The AWS acts like a server to the client, and like a client to the slave machine. All requests that AWS receives from the client will be sent the slave machine. All responses from the slave machine will be sent back to the client by AWS. For each slave the AWS server act like two pipes, with two new threads created to handle both pipes, one for connecting the client output to the slave machine's input, the other one for connecting the slave output to the client machine's input. Throughout the whole process, the AWS server does not parse any request or response. It does not need to understand the meaning of either requests or responses.

8.2 Redirect Scaling

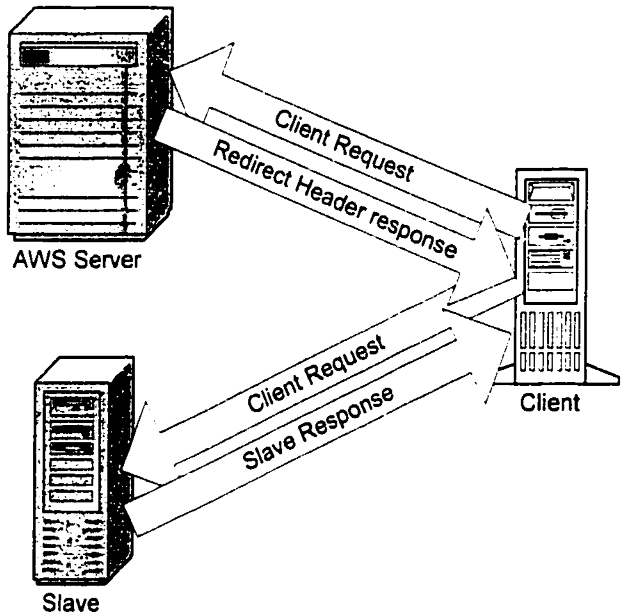

Figure 28 Redirect Scaling

In redirect scaling, AWS server acts like a redirecting server. The AWS server redirects clients to a new location, just as, when we using http://yahoo.com to access YAHOO!, the server redirects us to a new location http://www.yahoo.com. The user on the client side will not notice the redirection since the client Web browser does it automatically. There can be many slave machines registered on the AWS server, and each one of them may have a Web server running. However, every client machine must have an Internet connection and an individual IP address. Whenever a client connection comes in, the

AWS server assigns a slave machine to handle the incoming request and redirects the client to the assigned slave machine's new location by responding to the client with a redirection header. Once the redirection header is sent, the client will contact the slave machine directly by the redirection IP included in the redirection header.

Typical redirect header:

HTTP/1.1 302 RD
Location: http://www.yahoo.com/

8.3 Transparent vs Redirect

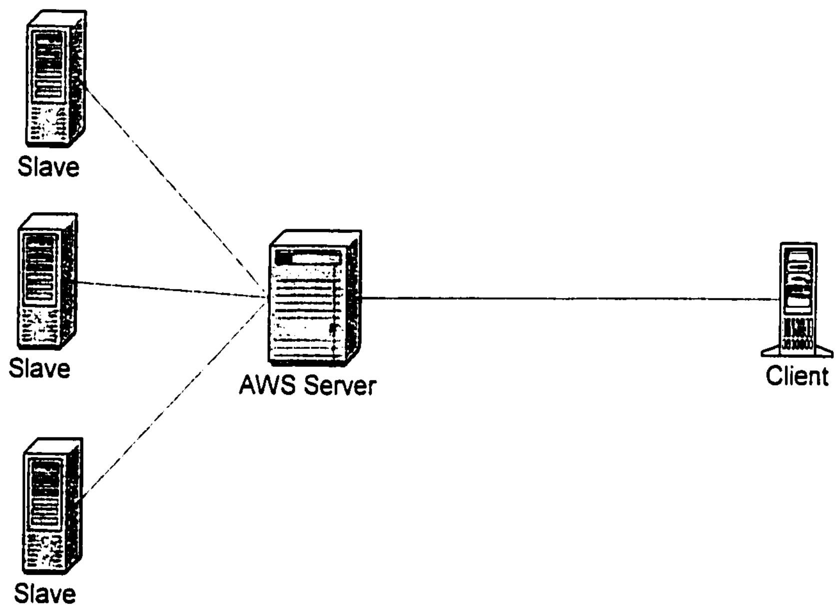

Figure 29 Transparent Scaling Connection Model

Figure 29 depicts the connection model of the transparent scaling approach. From this figure we can see that the major advantage of transparent scaling is that the slave machines do not need to have an Internet connection. The slave machines and AWS must be on the same Local Area Network (LAN). All slaves share the same IP as

AWS on the Internet but a different IP on the LAN. This approach involves a lot of computing resources on the AWS server, since all clients and slaves communicate through the AWS server. With this approach the number of threads running on the AWS server is doubled since AWS must use two threads to handle two communication channels. The hardware requirement for the AWS server is therefore relatively high. This approach is best for run AWS on a gateway machine that all slave machine connect to through a LAN connection. This approach is more efficient if number of concurrent connections is low, but each connection consumes a lot of CPU and memory resources. As the number of connections grows, the load on the AWS server increases since more threads must be created and more CPU and memory resources are consumed on the AWS server. Since slave machines handle the actual requests, requests that demand large amounts of computing resources benefits from this scaling model.

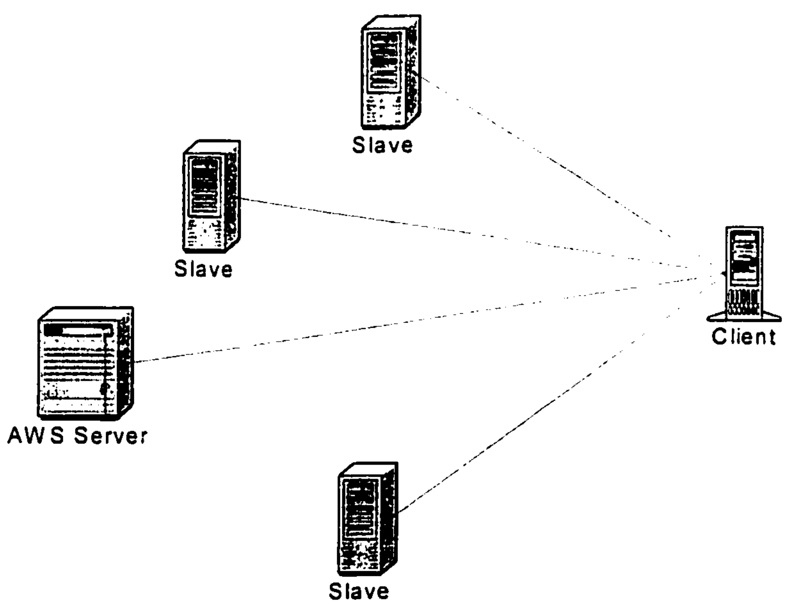

Figure 30 Redirect Scaling Connection Model

Figure 30 depicts the connection model of the redirect scaling approach. The major advantage of this approach is that it requires minimum resources on the AWS server. No extra threads need to be created; no parsing and processing of requests need to be done by the AWS server. The AWS server simply writes the redirect header to the client without any knowledge of the incoming request. However, with this approach, every slave machine needs to have an Internet connection and a static IP address on the Internet. This is usually costly because static IP address on the Internet is a very expensive resource. This approach does not require the AWS server to be strong in terms of performance. For requests that require both large and small amount of CPU time and memory, this approach does not make any difference. The overall performance of this approach is more closely related to the strength of slave machines.

8.4 Slave Assignment Algorithm

The slave assignment algorithm is mission-critical. AWS uses the random algorithm to select slave machines for handling requests. Different applications running on different slave machines need to be evaluated before implementing an assignment algorithm. A load-balanced algorithm can be implemented by custom according to the performance analysis of that particular computing environment. For serving static pages, with relatively equal distributed page sizes and relatively equal computing power for each slave machine, random algorithms might be the best, since load-balance algorithms incur a large amount of overhead. Overhead results from both the load-balance algorithm itself and network transmission. A typical load-balance algorithm will require the slave machine to send a "heart beat" to the master machine in order to provide updated CPU and memory usage information to the master machine. Fetching system information every few seconds and sending it to the master may require a new thread. Moreover, if one slave is loaded with many tasks (threads) and needs to transfer a few tasks to another slave machine that is idle, there is extra overhead entailed in copying all the state and partially completed data to the other machine. Most PCs of today can handle more than 100 typical Web requests per second, which means 1 request uses only 1% of the CPU resources when the CPU is busy. Migrating a request from one machine to another requires the slave machine to prepare data, the master machine to determine which slave is idle, another slave machine to receive the data. This may result in far more load on the system than would be entailed in allowing the request to be processed by the slave machine alone. However, the overhead is almost fixed for any request and transferring a small static Web page request needs approximately the same amount of overhead as transferring a large computational request. As the resource requirement gets larger and larger, the load-balance algorithm becomes more and more efficient. For example, if an application server can handle 1000 processes at the same time, each process requires 0.001 seconds to complete, using a random algorithm might be more efficient. If, on the other hand, the server can handle 5 processes at the same time, and each process requires 2 seconds to complete, a load-balance approach might be more efficient.

No matter what algorithm the server uses, a performance analysis and a prediction of incoming requests is necessary before we can design a good slave assignment algorithm.

Chapter 9 Performance Analysis

9.1 Usability Test

A usability test is done to test the following feature of AWS:

	Description of test	*Result of test*
Serving small static Web pages (HTML)	After installing AWS, try to access a static HTML page, with graphics, cascaded style sheet, java script or other Web content. These files are very small in size, usually a few k. These tests can show if the server is set up and running.	Passed
Serving large static files (.zip)	After the first test, try to download a large static file test.zip, which is about 30M in size. This test is similar to the first test. In addition, test the read and write buffers of the server for boundary conditions.	Passed
Serving different port	AWS is able to run on ports other than the default port 80. To change the port number, the AWS conf file needs to be modified. In this test, I changed the port number to 1000 and 10000 to see if I could access the server through these ports. This test is to show if the modification of conf will effect the AWS server running results.	Passed
Serving different Web root directory	In the AWS conf file, the user can specify the root directory of the HTTP servers. Usually I use c:\htmlserver as my root path. In this test, I renamed the c:\htmlserver to c:\html or other names and changed the ROOT_PATH directive in	Passed

	the AWS.conf as well. This test is to determine if the AWS server has variability when using a local path. No matter what the root path is, the client won't see any difference and the source code does not need to be re-compiled.	
Multiple default pages	AWS supports multiple default pages. In this test, I set up three directories with three different default pages: Index.html, index.htm, home.html. By adding these three names to the AWS.conf one by one, I was able to see all three pages by requesting their directory name instead of the full URL.	Passed
Directory Listing	Directory listing is also a feature of AWS that can be enabled and disabled through the AWS.conf file. I put all default page names inside the AWS.conf file in comments, and request the same directory again. When the directory-listing feature is enabled, I see a generated Web page that shows all files under that directory. When the feature is disabled, I see the "page not found" error on my browser.	Passed
Serving multiple client at same time	This test is done during the performance test. I use two machines to run my performance analysis program at the same time. These two client machines access the server machine at the time with large number of requests. This test verifies the server's ability to handle multiple requests at the same time.	Passed. After 3 minutes of testing not connection was rejected.
Connection pooling	AWS supports connection pooling. The connection pool size can be set through the AWS.conf file. I set the connection pool size to 50 and change the code of my ServerConnection class to print a unique sequential number of pool	Passed - From the print out result on the console I see the same sequential number of the first few connections again and

	connection. Then I do a few requests to the server. This test is to ensure that the connection pool is functioning and a connection can be recycled after been used.	again. This means the connection object is recycled and reused.
Support of Persistent Connection	Persistent connection is one import feature of HTTP 1.1. I test this feature by modifying the ServerConnection class to print out the port number of the connection every time it gets a request. If multiple requests come in using persistent connection, the port number of all requests using the same connection should be the same.	Passed. I see many request port numbers are the same. Which means these requests are sent through one single connection.
Serving CGI	I run one simple CGI program to test if the CGI handler is working. The simcgi.cgi displays information about the server. This is to test the server's standard CGI API.	Passed.
Serving Servlet	Install and run VWCL Servlet, which is the first stage of my ASP_OS project. The description of this Servlet can be found in Chapter 10. It is a general purpose Servlet. This test is to show the server's ability of support Servlets.	Passed. Servlet runs successfully after the registered URL is called. A dynamically generated HTML login page is shown.
Serving Servlet with session	The VWCL Servlet has used sessions to maintain user specific information such as the user's access right and user's name. It requires the user to log on to a session. After a set time period, the session should timeout and user will be required to login again.	Passed, after entering a password and username on the login page, the user is able to logon to the session. Different users will see different contents in response to the users' setting and access right. After a few minutes, when a user try to click on a link, the

	session time-out page appears and asks the user to login again.

Table 5 Usability Test

9.2 Performance Analysis

This performance test is in the following test environment:

AMD Athlon 700Mhz, 256M Ram, Dlink-10/100 Network card, Quantum 7200RPM Hard Drive.

Windows 2000 professional, IIS 5.0, Apache Tomcat 3.2.1, Redhat Linux 6.1, JSDK web server 2.1

In all tests, two machines are used, one for the server and one for the client. Numbers in the table are the numbers of requests completed per 5 minutes. Number of thread in each test means number of attacking thread generated by the stress test software.

	AWS(Sun VM)	AWS(MS VM)	IIS 5.0	Apache Tomcat
Single static file, 1 threads	44135	37629	55423	45098
Single static file, 5 threads	44575	37635	56017	44637

Single static file, 100 Threads	44359	37485	55873	44062
Servlet generated page, 1 thread	41825	35528	N/A	40751
Direct reject connection*	51772(Rejected)	47049(Rejected)	N/A	N/A

* **Note:** "Direct reject" means reject connection directly without sending any contents after a socket connection is accepted.

Table 6 Performance Analysis on Windows platform

In the Windows platform performance analysis, I compared AWS with Micorsoft IIS and Apache Tomcat. We also used two different Java virtual machines run AWS: the Sun Java VM comes with JDK 1.3, and the Microsoft Java VM comes with Visual J++ 6.0.

Sun Java VM vs Microsoft Java VM

The result of AWS running on different virtual machines was quite surprising. The Sun Java virtual machine outperformed the Microsoft one by almost 20%. Even through the AWS project is compiled into .exe format by using J++, the result made no improvement at all. This is a proof that the J++ executable Java format does not compile the Java source into machine code. It may be still in Java binary format and simply run by a integrated Java VM executable.

AWS vs Microsoft IIS 5.0

Even through Microsoft has poor support for Java and slow Java virtual machine, its Internet Information Server gives unbelievable performance results. AWS and Apache Tomcat did not even come close to its results in all tests. A gap in performance excess of 25% between AWS and Microsoft IIS pertains to many aspects. Java itself is not as efficient as C++. Moreover, IIS benefits tremendously from its integration with the Microsoft Windows NT operating system. However, IIS does not support Servlets directly. Support for Servlet on IIS needs an external Servlet container. Not many Servlet containers could be found on the Internet supporting IIS. None of them are free or open source.

AWS vs Apache Tomcat

Apache Tomcat is the most commonly used Servlet container. The results between Tomcat and AWS were not that obviously different. In most static page tests, Tomcat outperformed AWS. However, AWS got better results in Servlet-generated pages due to its integration of Servlet container and Web server: there is no interface between Web server and Servlet container. Parameter passing and communication between these two are minimized in AWS. AWS and Tomcat both have connection pooling, which may smooth the performance in test with large number of threads. An increase in number of attacking threads does not produce a dramatic change in result.

Windows vs Linux

Both AWS and Apache Tomcat were tested on both Windows 2000 and the Linux platform. Results for the Linux platform were much better than the results for Window 2000 platform. By using Linux Java virtual machine both server result about 10% gain in all tests. Since both servers uses the same set of source codes on both platforms, this performance gain is solely due to fast Linux kernel and Java virtual machine.

AWS vs Sun JSDK Web server

AWS outperformed Sun JSDK Web server in all tests. Since the JSDK Web server has no Servlet container, we did not use it to test the Servlet-generated page. When we compared three JAVA Web server implementations, AWS, JSDKws, and Tomcat, we found their performance lie in the almost the same level. Only about 10% variance among the three is due to implementation. Most important, however, was the performance of Java virtual machines. In contrast to the 10% implementation difference, we compared results using AWS on three virtual machines, Sun on Windows, Microsoft on Window, and Sun on Linux, it gives us more than 30% difference in performance.

Performance Limit

Performance of the virtual machine is the most important aspect of the Java Web server performance. No matter how good the Web server design is, there is a limit to its performance. The direct reject connection test is done to determine this limit of the virtual machine. In our test, a connection is rejected right after it is accepted without any manipulation, which means it represents the performance result of an ideal Web server that can use zero instruction to handle an incoming HTTP request. This perfection cannot be reached no matter how well the Web server is designed. Within 300 seconds of test period, Microsoft's Java virtual machine completed 47049 requests, Sun virtual machine completed 51772 requests on Windows platform and 59426 requests on the Linux platform. This test shows exactly how fast the virtual machine is. The best performance is resulted by to Sun virtual machine on a Linux platform, exceeding the Microsoft virtual machine on Windows platform by almost 30%.

	AWS	Sun JSDK Web Server	Apache Tomcat
Single static file, 1 threads	48923	44582	50098
Single static file, 5 threads	48575	44672	49749
Single static file, 100 Threads	48155	43785	48507
Servlet generated page, 1 thread	46436	N/A	56163
Direct reject connection*	59426 (Rejected)	N/A	N/A

Table 7 Performance Analysis on Linux platform

Chapter 10 Conclusion and Future Work

10.1 Conclusion

We may conclude that the AWS project is a successful project that delivers a Web server and Servlet container. It is an application server that supports JAVA Servlets, CGI and the latest HTTP and Java Servlet API 2.1. It supports scalability and is ready for further changes.

In the AWS project, my major contributions are:

1) Research on, and a survey of, a Web server and an application server.

2) Evaluation of application server technology

3) Implementation of a pure Java Web server

4) Integration of the Java Web server with a Servlet container

5) Implementation of two scaling model of the application server

10.2 Future Work

AWS is part of a bigger project — the ASP OS — that I am working on with Mr. Frederic Simard for the University of Quebec at Montreal. We are seeking venture capital to finance this project. The goal of the ASP OS project is to develop a low cost, easy-to-implement, commercial-grade solution for Application Service Providers (ASP). Before we source funds to complete the project at full throttle, we plan to proceed in four stages.

The application server model used by most companies is a Web server plus an application service container, as shown in Figure 31.

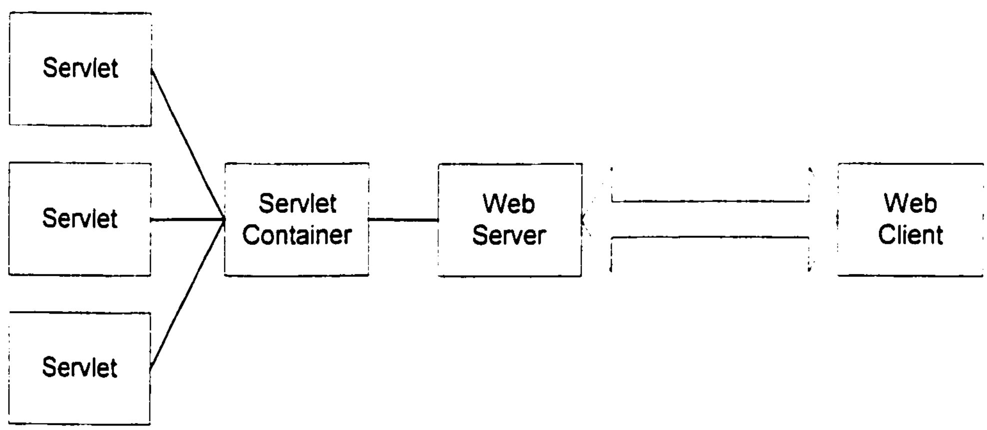

Figure 31 Classic Web Server and Servlet Container model

This model is also used by AWS, though it is not sufficient to meet the rapidly growing needs of the ASP industry, especially small and medium size companies. These ASP companies need hundreds of thousands of dollars to develop ASP applications and millions of dollars to cover operating and marketing costs. By using the ASP OS model, which requires less development and management, these companies could save more than 80% of the cost of developing their Web applications.

Four Stages of the ASPOS Project

Stage 1: Visual Web Component Library (VWCL)

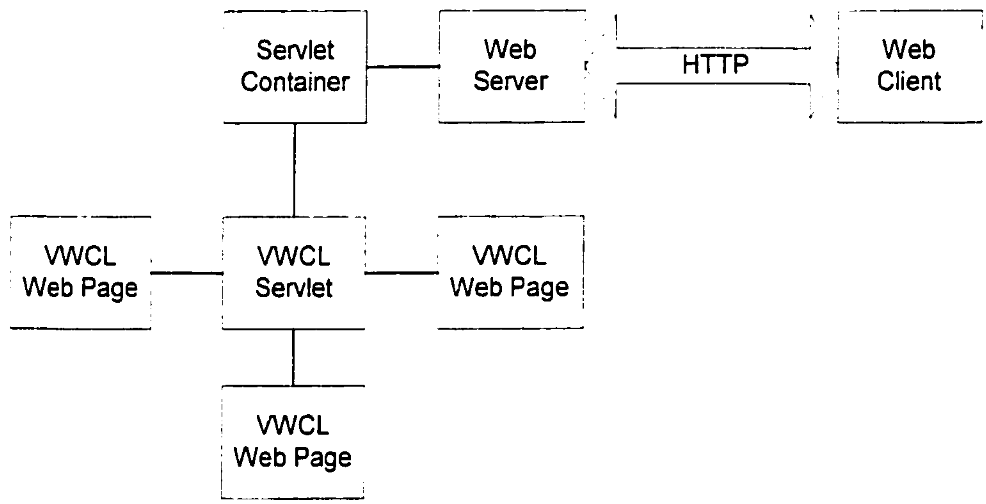

Figure 32 ASPOS stage 1

In Stage 1, nothing on the client side changes. A client would still use a Web browser such as IE and Netscape to browse a Web page. On the server side, the Web server and Servlet container would also remain unchanged. Any general-purpose Web server and Servlet container could do the job. In addition, a Visual Web Component Library (VWCL) — a Servlet that supports the JSDK 2.1 API so that it can be run as any other Servlet — is designed to support VWCL pages. VWCL is a Servlet that supports the JSDK 2.1 API so that it can be run as any other Servlet. The VWCL reads a XML page that contains layout information about a dynamic Web page, where the dynamic parts of the page are

SQL queries will replace these by the query result with a designed format. VWCL supports multi-user and multi-language, so different users will see different content on the same page, even in different language. For example, if a page contains some information about a user's curriculum, a different login name will result in a different query to the database, which will generate different results. If two users use different languages, the curriculum page will be displayed in a different language. The function of VWCL is similar to function of JSP, Servlet and ASP, which have become the current standard on the Web. However, it delivers stronger features and it is much easier to use. It does not need the user to have any programming background. A user who knows HTML and XML design is competent to do the job.

Stage 2: ASPLet

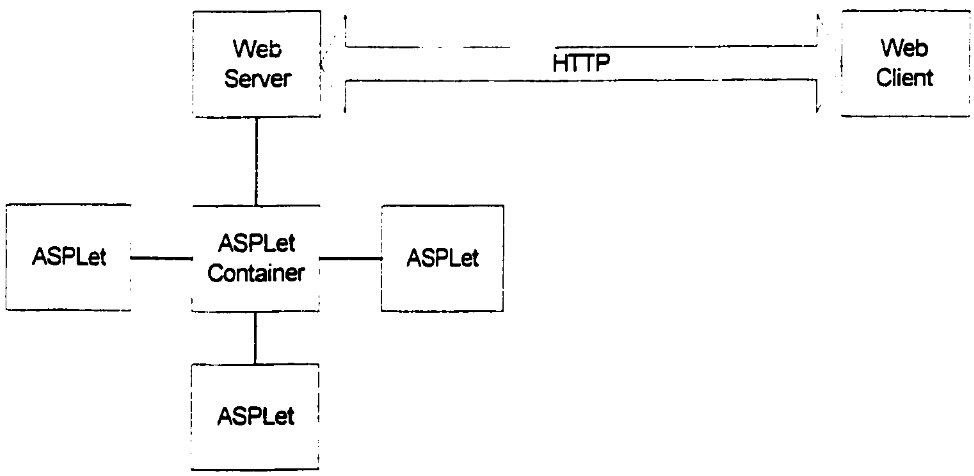

Figure 33 ASPOS Stage 2

Stage 1 does not support any action and manipulation of data on a static page. For example, a curriculum Web page might show user all courses in which a student client has registered. If the client needs to add a course or drop a course, or if he needs an alarm

10 minutes before a course starts, product from Stage 1 will not be sufficient to support these requirements. Stage 2 extends the Servlet API to an ASPLet API. Users can use the ASPLet API to design ASPLets. Like Servlets, ASPLets are small server-side programs. They use the VWCL and XML input, as Stage 1 does. They also support user-defined actions and functions on any VWCL components and HTTP requests. By using ASPLets, VWCL can handle user-designed actions and manipulations on inputs. By this means, a Web page will be more than a static page. It can contain both dynamic contents and dynamic actions. It will more like an application than a Web page. Many commercial products, such as Portal and Portlet technology from Oracle, are trying to achieve similar goals.

Stage 3: ASP Server-Client module

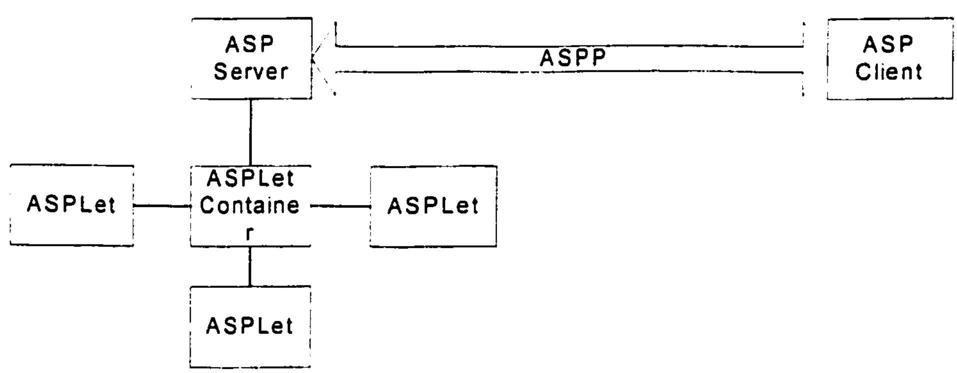

Figure 34 ASPOS stage 3

The product from Stage 2 is not sufficient to handle all ASP application needs. It will be limited by the current Web browser technology and the poor efficiency of the HTTP protocol. A new ASP client should be developed together with an ASP server, both supporting a new protocol — ASPP. ASPP will no longer be stateless and will be in binary format instead of the text format used by HTTP. By using binary format, request/response overhead is saved, therefore result less data transfer through the network. The main feature of an ASP client will be to support dynamic visual component creating. By using a Web browser, the whole graphical user interface is limited to the browser window and only HTML components can be created on the page. Web browsers do not support complex components such as tabbed list boxes. Implementing an ASP client, which will allow the server application to create any Java GUI component on the client's machine, will solve these limits. It will also transfer the event and event handling between server and client, since all GUI events occur on the client side and are handled on the server side. No commercial products currently have this feature. The industry is using Java applications to develop individual client-server applications, such as many types of banking software. The advantage of Stage 3 products is obvious: They would be easier for clients, since clients would only need to install one application for all ASPs to which they subscribe. Stage 3 products would also easier for developers, since no network communication modules are needed in an ASPLet, which is handled by ASP server and client.

Stage 4: ASP OS

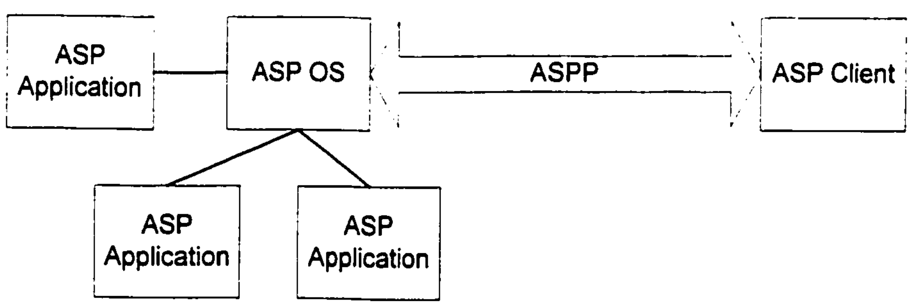

Figure 35 ASPOS Stage 4

The final goal of the project is to create an "operating system" for ASP. ASP applications can run on the server side of ASP OS. Clients can use the ASP client application to visit any ASP Web site and run any ASP application remotely. This approach can save much development time of client-server applications. Since the network communication details are handled by the ASP OS, developing an ASP application will like developing a normal application. The ASPOS will have all functions mentioned in stage 3 and, moreover, it will have several new modules such as user management, disk management, and memory management that will take care of all aspects of ASP application. This operating system is not like the classic operating systems such as Windows and Unix: it does not manage physical resources and detailed hardware. But in terms of application development, it is a platform like an operating system for which applications can be developed. Users on the client side can run the application from anywhere on the Internet. Starting an ASP client would be like "booting" a machine and loading the OS. Browsing an ASP site would be like running an application. Users will not see the details of server-client connection.

Reference

[1] UC Irvine et al., Hypertext Transfer Protocol -- HTTP/1.1 , RFC 2616, 1 April 1998.

[2] Thomas Mowbray and Ron Zahavi, "The essential CORBA: systems integration using distributed objects," Wiley, 1995

[3] James Gosling et al., "Java programming language, second edition," Addison Wesley, 1998

[4] David S. Platt, "Understanding COM+," Microsoft Press, 2000

[5] Lixin Tao, "Application Service Providers Model: Perspectives and Challenges" invited paper for International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR-2000), July 31 -- August 6, 2000, L'Aquila, Italy

[6] "Port", http://whatis.techtarget.com, 9 August 2000

[7] "Java Servlet API White Paper", http://sun.java.com, 2000

[8] Cheryl Gilbert, "3-tier application", http://whatis.techtarget.com, 02 August 2000

[9] "About the Apache HTTP Web server project", http://httpd.apache.org/ABOUT_APACHE.html, February 1999.

[10] Dustin R. Callaway, "Inside Servlets", Addison Wesley, 1999

[11] "Java Servlet Development Kit 2.1 API Documentation", http://sun.java.com, 2000

[12] "Cookie", http://www.whatis.com, 13 October 2000

[13] Todd E. Columb, "Web Servers Beyond Unix", http://www8.zdnet.com/pcmag/features/Webserver/_open.htm, PC Magazine, 1999

[14] Heath H. Herel, "Apache Web Server Review", http://www8.zdnet.com/pcmag/features/Webserver/iwsr1.htm, PC Magazine, 1999

[15] Todd E. Columb, "Microsoft Internet Information Server Review", http://www8.zdnet.com/pcmag/features/Webserver/iwsr2.htm, PC Magazine, 1999

[16] Frank Norman, "Uniform Resource Identifier", http://whatis.techtarget.com, 05 September 2000

Appendix A Extensive Survey on Application Servers[1]

Vendor	Product	Description
Allaire Corp. 888-939-2545 www.allaire.com	ColdFusion	ColdFusion is a Web application server for building and delivering scalable applications that integrate browser, server and database technologies. ColdFusion includes visual programming, database and debugging tools in an integrated development environment, ColdFusion Studio. It features open integration with databases, e-mail, directories, XML, and enterprise systems. ColdFusion Server features JIT compilation and caching. The deployment platform supports multi-server clusters with native load balancing and fail over to serve high volume, transaction intensive applications. ColdFusion offers database connectivity including support for ODBC, OLE DB and native database drivers for Oracle and Sybase, and is extensible with technologies such as COM and CORBA. ColdFusion has security services on every level from development through deployment.
Apple Computer 800-879-6398 www.apple.com	WebObjects	Apple's WebObjects is an application server environment that comes with tools for building Java-based network applications. WebObjects supports drag-and-drop construction of user interfaces and direct connections to existing applications and data resources. It features graphical development tools, a palette of prebuilt, reusable components, and seamless integration with all kinds of enterprise information systems. WebObjects handles application server requirements—including load balancing, state management, HTML generation, and Java client

[1] "Guide to Application Servers", APP Server Zone, http://www.appserver-zone.com/guide.asp, 2000

		interoperability. WebObjects 4 runs on UNIX and NT (and will also run on Mac OS X and HP-UX), and works across applications, business systems, and existing business logic.
Applied Reasoning 919-851-7677 www.appliedreasoning.com	Classic Blend	Classic Blend brings VisualWorks applications to the Web using a thin-client Java applet. It allows Smalltalk applications to be deployed over the Internet with a Java front-end. With Classic Blend, existing Smalltalk applications are re-architected from a client-server to a three-tiered application, resulting in a highly scalable, server-based distributed architecture.
Art Technology Group 617-859-1212 www.atg.com	Dynamo Application Server	Dynamo Application Server is a Java Web application server that features dynamic load distribution, automatic failover, and session recovery. The product supports assembling server applications from server-side JavaBeans components and Java Servlets, allowing an open, modular, and reusable architecture.
BEA Systems 800-817-4232 www.beasys.com	BEA WebLogic	BEA WebLogic is an extensible server for assembling, deploying, and managing distributed Java applications. With BEA WebLogic, Java business components can be interconnected with heterogeneous databases, network information resources, and other Java business components. You can manage your application components by using BEA WebLogic's graphic Java console to ensure security, scalability, performance, and transaction integrity.
Bluestone Software 609-727-4600 www.bluestone.com	Total-e-Server	Total-e-Server is an enterprise-class infrastructure that fully supports current open technology standards such as J2EE and XML. It delivers the core capabilities you need for a successful e-business. It offers scalability and fault tolerance through a combination of load balancing, data caching, state management,

		and reliable transaction processing.
Borland Inprise 408-431-1000 www.inprise.com	Inprise Application Server	The Inprise Application Server enables developers to readily build new, platform-independent applications, taking advantage of open industry standards such as CORBA and Java. It provides an integrated end-to-end solution for developing, integrating, deploying, and managing distributed-object applications that support multiple thin clients, mid-tier business logic, and standards-based database and legacy access. Features include open and extensible architecture based on industry standards such as CORBA, C++, Java, and HTML; and support for the Sun Solaris, HP-UX, IBM AIX, and Microsoft Windows NT platforms. Inprise Application Server also includes Sun's Java Web Server, and supports other major Web Servers, including Netscape, Microsoft, and Apache. VisiBroker Integrated Transaction Service (ITS), a flexible distributed-object-based transaction service that offers full support for the Java Transaction Service (JTS) standard, and AppCenter, a distributed applications-level management tool, are also included.
BulletProof Corporation 408-374-2323 www.bulletproof.com	JDesignerPro	JDesignerPro is a 100% Java development and deployment solution for building data-driven Web business applications. Visually design interactive database applications deployable through any browser. JDesignerPro includes an application server to distribute applications to users. It has a visual layout manager which allows point & click development and database access. JDesignerPro includes advanced Wizards that require no knowledge of Java to build and deploy finished applications. Other features include a user access control system, JDBC/ODBC database connectivity and connection pooling, remote management and a complete GUI designer with components.

Chili!Soft 425-957-1122 www.chilisoft.com	Chili!Soft ASP	Chili!Soft ASP is a Web application server based on the Active Server Pages (ASP) architecture. With versions for both NT and UNIX, Chili!Soft ASP allows you to build and host Web applications across multiple platforms. Chili!Soft ASP is available for Netscape, Apache, and Lotus Web servers on Windows NT, Solaris, and soon HP-UX and OS/390.
Data Access Corporation 305-238-0012 www.dataaccess.com	WebApp Server Product Suite	The WebApp Server Product Suite consists of two components: WebApp Server and WebApp Studio. WebApp Studio is a complete suite of visual tools for building Web applications. WebApp Server is a business logic engine, a powerful application server that provides Microsoft IIS/ASP developers with enormous capabilities for controlled, intelligent database integration, and centralized business rule and business process execution. WebApp Server extends Microsoft's dynamic internetworking architecture (DNA) by providing a place to store business logic in DNA's middle tier. WebApp Server provides a clean separation between the application interface and the application logic. Through the combination of a three-tier architecture, open database integration, robust business rules, and rapid development tools for high performance browser applications, the WebApp Server is truly designed for developing database applications on the Web.
Day Interactive 323-938-9888 +41 61-226-98-98 www.daynetwork.co m	Communiqué	Communiqué 2.0 is an application server with enhanced content management functionality designed to provide total Web information management solutions. Core functionality includes a browser-based, content-management system; completely dynamic Web servers; and scripting host. Additional characteristics include platform-independent, multithreaded, unlimited scalability; stable under heavy workloads; modular and object-oriented scripting language (ECMA Script, aka JavaScript); and nonforking. Features include a

		fully browser-based WYSIWYG content management system; syndicated/subscriber-oriented content replication across servers and sites; true load balancing across multiple servers and sites (even across various platforms); scriptable, ultrafast XML integration; a real-time, full-text linkchecker (site-internal and site-external); and wireless application protocol support. Supports XML-Object and FastDOM for application information interchange and database connectivity to mySQL, Oracle, Informix, Sybase, MSSWL, any ODBC source, LDAP integration, session support for e-commerce, and other profiled applications.
The dbXML Group, L.L.C. 480-421-1215 www.dbxmlgroup.com	Juggernaut Application Server	Juggernaut is an application server designed to provide a flexible Web application architecture. The architecture is a component-based, plug-and-play, and extensible framework. It is a complete product, open, customizable, and scalable. Juggernaut is primarily a platform for experimentation. Some have called it the "Anti-App Server," because it does not directly support Sun Microsystems' standards such as Servlets, JSP, and Enterprise JavaBeans. (Juggernaut Application Server was formerly maintained by Beach Dog Software.)
Delano Technology Corporation 905-947-2222 www.delanotech.com	Delano e-Business Interaction Suite	The Delano e-Business Interaction Suite is an application platform that enables companies to rapidly develop and deploy e-business applications that leverage e-mail and the Web to interact with customers, partners, suppliers, and employees. The platform includes a rapid application builder environment and a highly scalable server that enables companies to deploy e-business applications.
Dynalivery 314-205-8995 www.mobileapps.com	Parallel Crystal	Parallel Crystal Report Engine (PCRE) provides a scalable server solution for your report generation needs. Mobile Application Servers, Inc. developed PCRE under a license from Seagate Software and is

		based on the C++ and Microsoft Foundation Classes (MFC) in the Crystal Reports run-time or "print engine."
Esemplare Development 718-698-0070 esemplare.com	Galileo Application Server	The Galileo Application Server is a development system designed to allow developers to create dynamic database-driven applications through the use of custom HTML tags. It has been designed to simplify the design, development, deployment and maintenance of Web based applications. Galileo was written in Java and will run on any platform that supports the Java Virtual Machine, including Windows 95/98, Windows NT, SPARC Solaris, HP-UX and Linux.
Fujitsu Siemens Computers +49 (89) 636-47691 www.fujitsu-siemens.com/en/	PRIMERGY Servers	Fujitsu Siemens Computers' PRIMERGY server systems are based on Intel architecture, ensuring a high level of standardization and a good cost-benefit ratio. They also have a wide range of standard applications.
GemStone Systems 800-243-9369 www.gemstone.com	GemStone/J GemStone/S	GemStone/J is GemStone's application server for deploying and managing distributed Java applications in a three-tier environment. It is an integrated transactional server for Java and CORBA based on distributed JavaBeans and an array of enterprise services. It uses standards such as CORBA/IIOP and JDBC to ensure interoperability with other systems and integration with all data sources. GemStone/J is a JavaSoft-certified, Java-compatible system that can run all 100% Pure Java applications on the server. GemStone/S application server provides a highly advanced platform for developing, deploying, and managing scalable, high-performance, multi-tier applications. GemStone/S is based on a mature industry-standard language—Smalltalk. GemStone/S provides seamless integration with existing Smalltalk applications because its object model is the same as

		the Smalltalk object model.
HAHT Software 888-438-4248 www.haht.com	HAHTsite Scenario Server	The HAHTsite Scenario Server is a standards-based e-business server that offers platform features such as scalability, high availability, security and extensibility. The Scenario Server also offers integration features that provide a framework for intra-enterprise and extra-enterprise integration.
Halcyon Software 408-998-1998 www.halcyonsoft.com	Instant ASP	Instant ASP is a 100% Microsoft ASP-compatible server engine, it enables Web/enterprise developers to deploy ASP applications. Develop data-driven, Web-enabled, enterprise-class applications that can be deployed across multiple Web Server and Operating System platforms. Written entirely in Java, Instant ASP runs on Linux, Novell, Sun, MacOS, HPUX, SGI, SCO, DEC Alpha, IBM OS/2, RS/6000, AS/400, S/390, and Windows. It supports Apache, FastTrack/Enterprise servers, Sun WebServer, Java WebServer, IIS, WebSphere, Lotus Domino, and most Web servers.
IBM Corp. 800-426-4968 www.ibm.com	WebSphere Application Server	IBM WebSphere Application Server combines a runtime environment for Java Servlets with connectors to common database formats, industry-standard object request brokers, and enterprise middleware. It runs on major industry HTTP servers.
ICOM Informatics 512-335-8200 www.icominfo.com	Winsurf Mainframe Access	Winsurf Mainframe Access (WMA) is a Web-to-host connectivity solution that integrates Internet, intranet, and extranet technologies into mainframe environments. WMA provides Web-based access to a range of IBM S/390, AS/400, DEC/UNIX, and Bull hosts. It also provides mainframe access connectivity without requiring changes to host applications. Installed on a Windows NT server equipped with

		Microsoft Internet Information Server (IIS), WMA ensures the administration of users who access host-based applications.
Iona Technologies 781-902-8000 www.iona.com	iPortal Application Server	The Iona iPortal Application Server is an EJB 1.1 Server/Container combined with a graphical interface for the assembly, deployment and administration of Enterprise JavaBeans. It includes all the services required by the EJB 1.1 standard, including support for session and entity beans, Java Naming and Directory Interface (JNDI), EJB Security and RMI-IIOP. Container and bean-managed persistence is supported by XA conformant JDBC 2.0 technology for connectivity with more than 100 databases.
iPlanet 888-786-8111 www.iplanet.com	iPlanet Application Server	iPlanet Application Server provides a J2EE e-commerce platform for delivering application services to servers, clients and devices. It enables rapid-time-to-market, the ability to handle unplanned success, and the ability to leverage information systems and business processes across the extended enterprise.
JADE (Aoraki Corporation Ltd.) +64 3-365-2500 www.discoverjade.com	JADE	JADE is an application programming technology used to build enterprise-class, transaction-based business systems, particularly high-volume e-commerce applications. JADE comprises both a rich, seamless development environment and a unique distributed computing environment. JADE has a single development environment that contains all the functionality necessary to build an entire system integrated together so that development integrity and quality are improved. It also uses a single language, eliminating the need for multiple languages in one system, greatly improving initial development and maintenance. JADE can integrate external parts (ActiveX controls, external methods, and relational database access) and be integrated into external systems (ODBC, multiple language APIs). JADE

		distributes and manages the execution of your system across Windows NT platforms and through LANs/WANs and the Internet. JADE Server can also run on IBM RS/6000 under the AIX operating system. JADE 5 is the recently released enterprise-class version of JADE.
Level 8 Systems, Inc. 800-499-7337 www.level8.com	Geneva Enterprise Integrator	Geneva Enterprise Integrator (EI) is a memory-resident operational resource manager, delivering real-time information integration. Geneva EI uses distributed object technology to integrate applications, information flows, and data into an active model (structure and state) of an enterprise. The active model is also a platform for implementing new enterprise applications.
Lotus Development 800-343-5414 www.lotus.com	Lotus Domino R5 Application Server	The Lotus Domino Application Server is an open, secure platform optimized to support rapid delivery of collaborative Web applications that integrate enterprise systems with dynamic business processes. It allows you to leverage current information assets with built-in connection services for live access to relational databases, transaction systems and ERP applications.
Lutris Technologies 831-471-9753 www.lutris.com	Enhydra Application Servers	The Lutris Enhydra application server offerings, Lutris Enhydra Standard and Lutris Enhydra Professional, are tools for building comprehensive Web and wireless applications. At the center of both Standard and Professional is the Enhydra Java/XML application server development and deployment environment.
Math Wizards Inc. 619-552-9031 www.mathwizards.com	MathXplorer/JS	MathXplorer/JS is a Java RMI distributed Math (application) server for the Web that provides software developers and Web designers distributed and platform-independent numerical computing services.

Merant 800-872-6265 www.merant.com	Micro Focus Server Express	Server Express, a Micro Focus COBOL product for UNIX, is a platform for deploying e-business and distributed applications. It is specifically designed to support high-volume transaction processing applications. Server Express accelerates enterprise COBOL application performance.
Microsoft 800-426-9400 www.microsoft.com	Microsoft Transaction Server Internet Information Server	Microsoft Transaction Server (MTS) is a component-based transaction processing system for building reliable, scalable, distributed Internet and Intranet applications. MTS is a feature of Microsoft Windows NT operating system that simplifies the development and deployment of server-centric applications built using Microsoft Component Object Model (COM) technologies. Internet Information Services (IIS) 5.0, the most important Web technology integrated with Windows 2000 Server, makes the platform a powerful Internet and intranet Web application server. IIS extends the Web capabilities of server operating systems by letting users share information, build and deploy business applications, and host and manage sites on the Web.
Miva Corporation 619-490-2570 www.miva.com	Miva Mia	Miva Mia is a desktop server for running applications written in Miva Script, an XML (tag based) server side scripting language. A desktop Web server and xBase compatible database are fully integrated into Miva Mia. Applications developed using Miva Mia are fully cross platform and can be run on any Windows NT or Unix Web server that has been enabled with Miva Empresa, the enterprise and hosting service edition of Miva Mia.
ObjectSpace, Inc. 800-OBJECT-1 or 972-726-4100	Voyager Application Server	Voyager Application Server offers the means to build scalable, secure, distributed, transaction-oriented applications. Because Voyager embraces the standard

www.objectspace.com		Java server component model, Enterprise JavaBeans, application developers have access to the burgeoning market of off-the-shelf, server-side application components. Voyager also allows application developers to concentrate on business logic, not infrastructure. The details of transactions, security, threading, and distribution are handled transparently. Voyager's advanced graphical tools also simplify application deployment, management, and monitoring. Voyager is the only application server that has a universal container.
Oracle Corp. 800-672-2531 www.oracle.com	Oracle Application Server	Oracle Application Server (OAS) integrates all of the core services and features required for building, deploying, and managing high performance, n-tier, transaction-oriented, Web applications within an open standards framework. These essential capabilities include HTTP Web server and support for popular Web servers; database and legacy access middleware for connection to all major databases; CORBA ORB support for scalable, cross platform, distributed object deployment; TP monitor capability for load balancing, pooling and transactions; network services like security and directory; and message-oriented middleware for extensibility that simplifies the enabling of new applications. OAS offers cross-platform support for all types of network clients (HTML, Java, CORBA), Web servers, and databases, which preserves existing investments in legacy and client/server systems.
Orbware www.orbware.com	OrCAS Enterprise Server	OrCAS Enterprise Server is a pure Java EJB application server from Orbware, a J2EE licensee. OrCAS is a full implementation of the EJB 1.1 specification and is also available as a full J2EE platform. An ASP Edition is available with support for virtual hosting of J2EE applications. The product is supported on Windows NT, Linux, and Solaris.

Persistence Software 650-372-3600 www.persistence.com	PowerTier	PowerTier is an application server with multiple patents in performance, scalability, and developer productivity. With PowerTier's shared transactional object cache, data is moved into memory and away from the database, eliminating processing strains on the back-end and speeding up response times by as much as 50 times. With PowerSync, PowerTier synchronizes the caches of distributed servers, so users have real-time, local access to current data from the closest PowerTier server. And PowerTier automatically generates EJBs, JSPs, and XML code with object modeling tools such as Rational Rose or Together Control Center reducing the learning curve for developers.
Pervasive Software 800-287-4383 www.pervasive.com	Tango 2000 Application Server	Tango 2000 provides a complete suite of products for the development, deployment and maintenance of business critical Web applications. Tango Application Server dynamically creates HTML output based on what's contained in your databases at a given moment. Think of the Application Server as a powerful broker between the client display logic and an organization's information systems—everything from the database to e-mail. Failover detection ensures site availability.
PortalSphere Inc. 613-722-7389 www.portalsphere.com	PortalSphere Application Server	PortalSphere Application Server is a powerful CORBA-based development and execution environment for e-business. It supports client-side development in HTML, ASP, Java/JavaScript, Visual Basic/VBScript, ActiveX, Macromedia, Microsoft Office (Word, Excel), and C/C++. Applications run on Linux (UNIX) with simultaneous native access to multiple databases, including Oracle, Sybase, and MySQL. It supports real-time event processing.

		Server object library includes Excel-compatible spreadsheets, XML, LDAP, and facilitates development of reusable components. Delivers full real-time event processing; benchmarks up to 100 times faster than HTTP/CGI.
Pramati Technologies Ltd. +91 40-374-3295 www.pramati.com	Pramati Server and Studio for J2EE	Proton is a Web application server written in Java. Proton supports multi-tier Web applications with HTML for presentation layer, Java objects for business logic, and interfacing to enterprise information servers. Proton now supports large-scale enterprise applications with high-performance features, dynamic load balancing, failover management, automatic restart, remote administration and monitoring. Proton provides complete access to Java objects from HTML pages. User interface developers can use simple HTML tools without worrying about the complexity of the business processing on the application layer. Proton uses 100% Pure Java and will run on any platform with Java 1.1 or higher.
Progress Software Corp. 781-280-4000 www.progress.com	Open AppServer	Progress Open AppServer supports distributed Future Proof enterprise applications that leverage existing investments, support new technologies, and communicate with other applications as needed. An integrated application server for both Progress 4GL Version 9 and WebSpeed 3 Web-based applications, Progress Open AppServer forms a middle tier between an application's user interface and its database. Invisible to application users, Open AppServer allows interoperability with virtually any client and data source.
Sagent Technology 800-782-7988 www.sagent.com	AddressBroker	AddressBroker is an enterprise server designed for client/server environments that handle high-volume customer data record requests. Features include load

www.sagent.com		balancing, multiprocessing, and a Handle Manager. C, C++, Java and ActiveX provided. Connectivity is available through HP/UX, IBM's AIX, Sun Solaris and Digital Unix, as well as Windows NT.
Secant Technologies Inc. 216-595-3830 www.secant.com	Secant Extreme Enterprise Server	Secant Extreme is an Enterprise Object Application Server environment for developing scalable multi-tier applications. Built on top of Secant's Persistent Object Manager (POM) Server, which maps Java and C++ objects to relational databases, Extreme provides a complete Enterprise JavaBeans framework and implementation of CORBA services including transactions, security, events, concurrency, and locking. Coupled with Rational Rose98, developers can use Extreme to model, partition, and generate the framework of a multi-tier application. Extreme includes deployed application management, extensive object services, automatic state, and session management for Web applications and dynamic load balancing.
Servertec 201-998-1048 www.servertec.com	iServer	Servertec iServer is an independent Web server written in Java that can serve static Web pages and generate Web pages using Java Servlets, iScript, CGI and Server Side Includes. Provides a scalable platform to establish a Web presence and deploy cross-platform Internet and extranet applications. Features include platform independence, open standards, and extensible architecture.
SilverStream 781-238-5400 www.silverstream.com	SilverStream Application Server	SilverStream Application Server is an enterprise application server that allows corporations to build and deploy complex Java and HTML applications on which they can run their businesses. It is designed and optimized for the Intra/Inter/Extranet, and delivers both client- and server-side Java and client-side HTML. SilverStream integrates business logic, extensive database access, content creation, publishing, collaboration, and communications in one

		solution.
SoftQuad Inc. 416-544-8879 www.softquad.com	HoTMetaL Application Server	SoftQuad's HoTMetaL Application Server is a server-based engine for hosting and deploying dynamic, browser-independent Web applications with complete security and low maintenance. HoTMetaL Application Server can be used for deploying data-driven customer service, e-commerce, and personalized content applications developed using Miva Markup Language (MML), a standards-based, tags-oriented language. Supports UNIX, BDSI, Solaris, Linux (CGI), Windows NT.
Sun Microsystems Inc. 512-434-1591 www.sun.com	Java Embedded Server	Java Embedded Server is a small-footprint network server that enables real-time deployment and installation of applications to remote embedded devices. Powered by the modular architecture of the JavaServer Engine and the JavaServer Services, the Java Embedded Server allows remote devices to upload, download, activate, and deploy customized services and applications precisely when they are needed. Employing the Java Embedded Server means that services can be built, rebuilt, added, removed, customized, and utilized on a real-time and variable basis.
Sybase Inc. 800-879-2273 www.sybase.com	EAServer	Sybase's EAServer, a scalable application server for e-portal and Internet solutions, provides a set of services for deploying Web and distributed applications using core Java 2, J2EE technologies. EAServer also offers broad support for applications, including those based on CORBA, XML, HTML, DHTML, any ActiveX client, PowerBuilder, COM, C and C++.
Tempest 212-624-4156 www.tempest.com	SiteShield	SiteShield is a "plug and play" software solution consisting of the SiteShield Server and the SiteShield Bridge that secures Web-based information exchange.

		The product keeps enterprise application data securely behind a firewall closed to all incoming connections. This technology allows you to close all inbound ports in your firewall while authorizing controlled access to your Web server behind your firewall. The product enables multiple partners to exchange information from these Web servers.
Tomahawk Technologies Inc. 613-257-4141 www.tomahawktech.com	SteelArrow	SteelArrow is an easy-to-use Web Application Server offering dynamic database connectivity and a feature-rich HTML embedded markup scripting language. WebHawk, the editor bundled with SteelArrow, offers Database and HTML-table wizards that allow Web developers to produce dynamic Web applications in a snap. The WebHawk editor (Win95/98/NT) also allows developers to test and debug scripts on-the-fly while editing, a true benefit for Road Warriors. SteelArrow provides automated e-mail and e-news support, timed script execution, database connectivity, database connection and session pooling, data scraping from other Web pages, dynamic file creation, user- and group-level security, conditional logic evaluation and looping, and integration with other Web technologies. SteelArrow offers developers full Web application development functionality and run-time reliability. It operates as an extension to Microsoft IIS and Netscape Enterprise servers, providing true scalabilty and cost effectiveness.
Unify Corporation 972-871-4600 www.unify.com	Unify VISION	Unify VISION App Server is an open, standards-based Internet application server that enables IT organizations to bridge legacy, custom-built and packaged applications with the Internet. Its universal client architecture enables users to access any application, anywhere, anytime. Unify VISION AppServer's Parallel Dynamic Scaleable Architecture offers server replication, load balancing, fail-over and recovery, and publish-and-subscribe

		capabilities, all based on a fully asynchronous messaging architecture. Integrated application management services lower total cost of ownership by allowing organizations to effectively manage their applications from a single point of control.
Unify Corporation 972-871-4600 www.unify.com	ServletExec	ServletExec is a Java-based Web application server that features complete implementation of the Java Servlet API 2.0, including Session Tracking; Servlet Aliases, Chains, and Response Filters; Remote Servlets. It supports JavaServer Pages (JSP) as defined by Draft Specification 0.91. It supports major Web servers and operating systems, including Microsoft IIS & PWS, Netscape FastTrack & Enterpise (UNIX & Windows), Apache (UNIX & Windows), and Mac OS Web servers.
Unify Corporation 972-871-4600 www.unify.com	Unify eWave Engine	Unify eWave Engine is a scalable, enterprise-caliber Java application server designed for e-Commerce and Web portals. It features a high-performance, scalable architecture; support for COM, DCOM, RMI, HTTP; replication; automatic load balancing; automatic failover; and role-based security.
VelociGen Inc. 858.622.1164 www.velocigen.com	VelociGen Application Server	VelociGen is a Perl-driven application server for distributing Web applications. Features include advanced templating based on custom HTML tags, script compilation and caching, persistent database connections, and built-in XML support.
Versata Inc. 800-984-7638 www.versata.com	Versata E-Business Automation	The Versata E-Business Automation System enables the rapid deployment and evolution of complex, business-to-business, transaction-based Web applications. Versata Logic Server (an open, scalable platform to compile and execute business rules), Versata Studio (a team-based development environment that defines business rules and the e-commerce presentation layer), and Versata

		Connectors (links for the e-business applications to legacy resources) comprise the Versata E-Business Automation System.
Zope 540-371-6909 www.zope.org	Zope	Zope is an open source application server and portal toolkit used for building high-performance, dynamic Web sites. This object-based Web application platform allows you to build dynamic Web applications easily. Zope is completely managed through the Web and uses the Web object model as its design. Zope's framework provides a safety net, including access control, undo, private versions, and more. Zope is based on Python, and offers support for CORBA, COM, XML, and leading databases.

Appendix B AWS installation and configuration manual

Complier and Installation of AWS

Downloading AWS

Information on the latest version of AWS can be found on the AWS Web site at http://www3.sympatico.ca/abao. If you downloaded a complied distribution, skip to Installing AWS. Otherwise read the next section for how to compile the server.

How to compile AWS

Step 1:

Make Sure a JDK1.3 and JSDK 2.1 is installed. JDK1.3 and JSDK 2.1 can be found on Sun's Web site. Set up the JAVA_HOME environment variable to be the path where JDK1.3 installed using the following DOS command:

Set JAVA_HOME=c:\JDK1.3

Step 2:

Unzip the AWS.zip file. You should see a directory called AWS.

Step 3:

Under AWS/bin type make.bat to compile the AWS server.

Step 4:

If you run AWS under windows platform, the tray icon GUI will be showed. This requires support of Dynamic Link Library. There are two .dll file located under AWS/lib which are required for supporting tray icon. Copy these two files into your Windows/system or Winnt/system32 directory to enable these libraries.

Install AWS

The next step is to edit the configuration files for the server. This consists of setting up various **directives** in up to three central configuration files. By default, these files are located in the AWS/conf directory and are called AWS.conf, Servlet.conf and Session.conf. Read the comments in each file carefully. AWS.conf contains configuration about the Web server; Servlet.conf contains configuration about the Servlet container; Session.conf contains configuration about Session. Failure to setup these files correctly could lead to your server not working or being insecure.

Configuration and directives of AWS

AWS.conf

In this file, you can set up many attributes of the AWS server.

Port:

Default format:

port = 80

This directive set the port that AWS will listen to. The default HTTP port is 80.

MAX_NUMBER_OF_CONNECTIONS:

Default format:

MAX_NUMBER_OF_CONNECTIONS=1000

This is to set up the maximum number of connections allowed at the same time. Default value is 1000

RESERVED_NUMBER_OF_CONNECTIONS:

Default format:

RESERVED_NUMBER_OF_CONNECTIONS =100

This is to set up the connection pool size. A connection pool is a set of connection objects that is already initialized and waiting for connections. Once a connection comes in, the server will assign a connection object to it. After a connection is closed, it will be return to the connection pool and ready for reuse. It is an optimization to Web server.

ROOT_PATH:

Default format:

ROOT_PATH = c:\htmlserver

This directive set the root directory of the Web server. In this default example, it uses c:\htmlerver as the root directory. Therefore if the Web site domain is http://www.AWS.com by typing http://www.AWS.com/index.html in the URL line of your Web browser. You will get the file c:\htmlserver\index.html on the server.

Host:

Default format:

HOST = http://localhost

This directive set the domain name for the Web site.

MainThreadPriority:

Default format:

MainThreadPriority = 9

MainThreadPriority directive set thread priority of the Web server main thread.

ChildThreadPriority:

Default format:

ChildThreadPriority = 6

ChildThreadPriority directive set thread priority of the Web server child thread. There is only one main thread but many child threads. Main thread controls the whole program and assign connection object to a new child thread when a connection comes in.

Session_Configuration_File:

Default format:

Session_Configuration_File=.\Session.conf

Session_Configuration_File directive gives out the path of Session.conf file. This needs to be modified to the conf path you installed AWS.

Servlet_Configuration_File:

Default format:

Servlet_Configuration_File=.\Session.conf

Servlet_Configuration_File directive gives out the path of Servlet.conf file. This needs to be modified to the conf path you installed AWS.

serverName:

Default format:

serverName = AWS

This directive does not need to be modified, it is only for changing the name of the AWS server when send response to a client and display an index page of a directory.

serverVersion:

Default format:

serverVersion = v1.50

This directive does not need to be modified, it give out a version number when displaying an index page of a directory.

serverUrl:

Default format:

serverUrl = http://www3.sympatico.ca/abao/

This directive does not need to be modified, it give out a link to the AWS home page.

Directory_Indexing:

Default format:

Directory_Indexing = true

This directive enables or disables the directory indexing when a default page cannot be found on the requested path. The default value is true. Set false will disable the directory indexing.

Default_Filename:

Default format:

Default_Filename=index.html

This directive set up the default page name, usually index.html and index.htm. When a default page cannot be found on the requested path the default page is showed. This directive can have multiple lines for different names.

ENABLE_CGI:

Default format:

ENABLE_CGI=true

This directive enables CGI support. Set to a false value will disable the server's CGI support function.

Servlet.conf

Servlet:

Default format:

servlet=SampleServlet

This directive register a name of Servlet, is name is used to fetch more information about that Servlet in a servlet.NAME.ATTRIBUTENAME format.

servlet.SampleServlet.code:

Default format:

servlet. SampleServlet.code= SampleServlet

This directive gives out a class name of the Servlet. Java Virtual Machine will look for this class in all class paths. Before you register a Servlet, you should include the path of the .class file of the Servlet in the classpath. In a default situation, you should always include the ROOT_PATH/Servlets path in your classpath. Where ROOT_PATH is setted up in AWS.conf. Then you should copy the .class file of your

Servlet into that directory.

servlet.SampleServlet.urlpattern:

Default format:

servlet.SampleServlet.urlpattern = SampleServlet

This directive gives out a URL pattern such as *.cgi and ?sample.???. It can be used as a wide card marching from a URL to the Servlet. Most Servlet uses its own name as a urlpattern.

servlet.SampleServlet.initarg:

Default format:

servlet.SampleServlet. initarg = SampleServlet.conf

This directive gives out a file name which contains the initial argument of the Servlet.

Session.conf

session.timeout:

Default format:

session.timeout = 60000

This directive set up the time out of a Servlet session in milliseconds.

session.newtimeout:

Default format:

session.timeout = 60000

This directive set up the time out of a new Servlet session in milliseconds. A new Servlet session means that session has been created but not used.

session.checkFrequency:

Default format:

session.checkFrequency = 5000

This directive set up the time period that how frequently the session manager will check all sessions for timeout.

session.useCookies:

Default format:

session.useCookies = true

There is two ways to obtain a session. One way is to save the session ID as a cookie. Another way is pass it through URL parameters. Use cookie is a preferred way, which is safer for a client to pass the session ID as a cookie.

All three configuration files are very important to set up AWS on a Web site. The server will not functioning well unless it is properly set. Usually editing a text file is a boring work to do. A graphical user interface is developed under help of the Concordia COMP 628 class group to configure the server easier.

Run AWS

Under AWS/bin type startup.bat