Connecting Enterprise Data to the Web using WebSphere

An IBM e-business Experience Report

Introduction

Many companies are now looking for ways to use the web environment to provide additional services to their customers, as well as to their employees. For example, a customer self-service application, allowing customers to view and update their own information via the Internet, can both improve customer service and reduce costs. In many cases, web applications such as this can be easily implemented by web-enabling a company's existing applications and data.

For our test application (INSCO, short for Insurance Company), we decided to model an insurance company interested in providing customer self-service via the web. In our example, this company has existing enterprise data residing in a DB2 database.

The Application Model

Our program is structured following the logical three-tier model of the Application Framework for e-business.[1] The logical three-tier model is an excellent solution for a customer self-service application. The model, shown in *Figure 1* below, typically has a web browser as the client, a middle tier web server and web application, and third tier enterprise data. Because the client is a web browser, and uses industry standard Internet protocols such as HTTP and HTTPS to communicate with the middle tier, the application is easily accessible by a wide set of customers via the Internet, without the customer needing to install any additional client software. The actual web application code is invoked by the web server, and accesses third tier enterprise data or applications. The enterprise data or applications remains on the third tier, without modification.

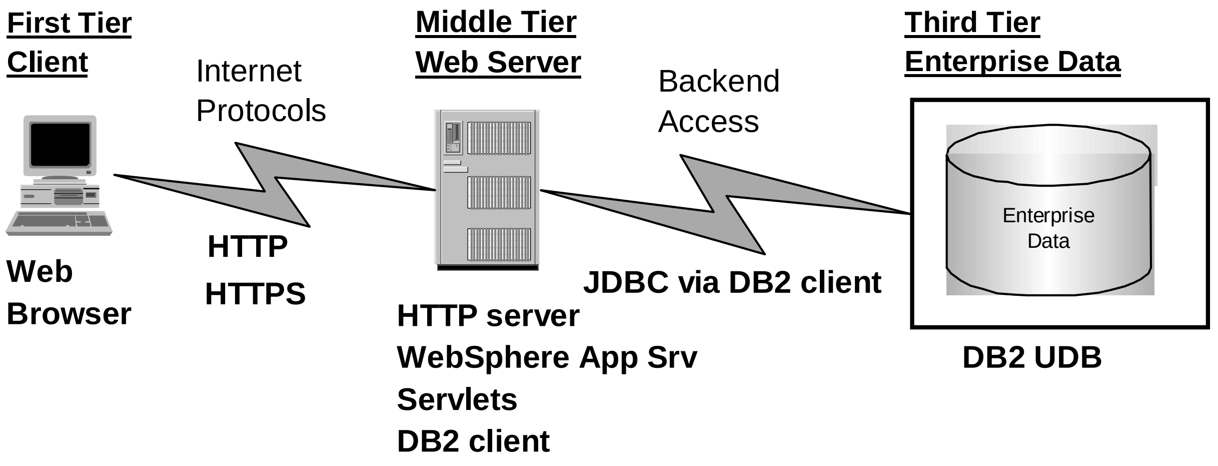

Figure 1: **Logical Three-Tier Model**

[1] For more information on the Application Framework for e-business, please visit http://www.software.ibm.com/ebusiness.

The client, middle tier, and third tier for our application are described below. We also have additional infrastructure to support security, scalability, and availability which are described in the Infrastructure section.

Client

In an effort to support Internet access for all our customers, our solution minimizes the hardware and software requirements on the client. The client system can be any web browser supporting HTTP and HTTPS. The client processing is limited to the user interface, which is simple HTML. The actual retrieval and formatting of data is done by Java servlets and Java Server Pages (JSPs) residing on the middle tier. The client system is required to support Secure Sockets Layer (SSL) to ensure that personal information transmitted is kept confidential, and to support cookies for session management support.

Middle Tier

The middle tier has the IBM WebSphere Application Server working in conjunction with an HTTP server. The WebSphere Application Server is designed to work with different HTTP servers including Domino Go, Netscape, Apache, and IIS. For this test phase, we tested our implementation with both Domino Go and Microsoft Internet Information Server IIS. We chose to implement our application logic using Java servlets. The Java servlet receives and processes requests for information from the client. It then coordinates the retrieval of the information from the DB2 Universal Database Server and the return of the information to the client. Java Server Pages are used to format the data into HTML for display on the client. The HTTP Server has been configured for SSL so that the information sent between the browser and the web server is secure.

Client-Middle Tier Internet Protocols

The communication between the client and the web server uses two of the standard Internet protocols: HTTP and HTTPS. HTTP is used to establish the initial communication between the client and the server. HTTPS is used thereafter to securely send the specific requests and information between the client and the web server.

Third Tier

The data for our application resides in a DB2 Universal Database on the third tier.

Web Server - Third Tier Access

The communication between our application (Java servlet) and the DB2 datastore is provided by the DB2 Java Database Connectivity (JDBC) drivers included with the DB2 Client Application Enabler (CAE). The middle tier Java servlet issues JDBC calls to the DB2 client. The DB2 client then issues traditional DB2 client-DB2 server requests to retrieve the data from DB2

The Infrastructure

Opening company data and systems to public access via the Internet increases the potential for these systems' security to be compromised. For security requirements, we separated the web servers handling customer requests from the Internet, from the actual enterprise databases located in the intranet. This is called a screened subnet architecture and is shown in *Figure 2* below:

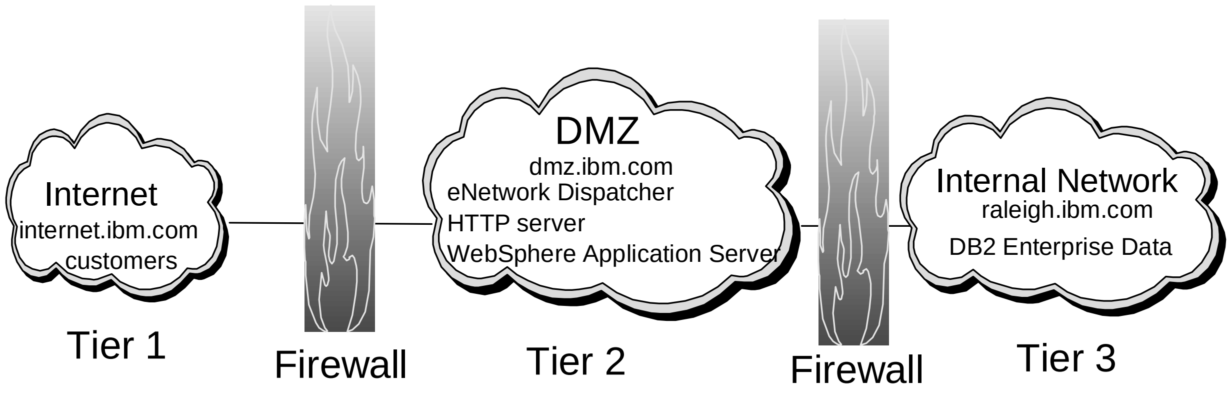

***Figure 2*: Screened Subnet Architecture**

The scalability and availability requirements were met using two components from the WebSphere Performance Pack (WSPP). We used the IBM eNetwork Dispatcher component to balance load across multiple web servers, and AFS to keep the content synchronized across these web servers.

For test purposes, we implemented our solution on a variety of platforms. We set up our browsers for testing on Windows 95, Windows NT, and Network Station platforms. We deployed WebSphere Application servers on both Windows NT and AIX, and the enterprise DB2 database on an AIX platform. The screened subnet architecture was implemented using one physical firewall on AIX. Load balancing was accomplished using eNetwork Dispatcher on Windows NT. The AFS server and replica systems were established on AIX. Our complete environment is shown in *Figure 3*.

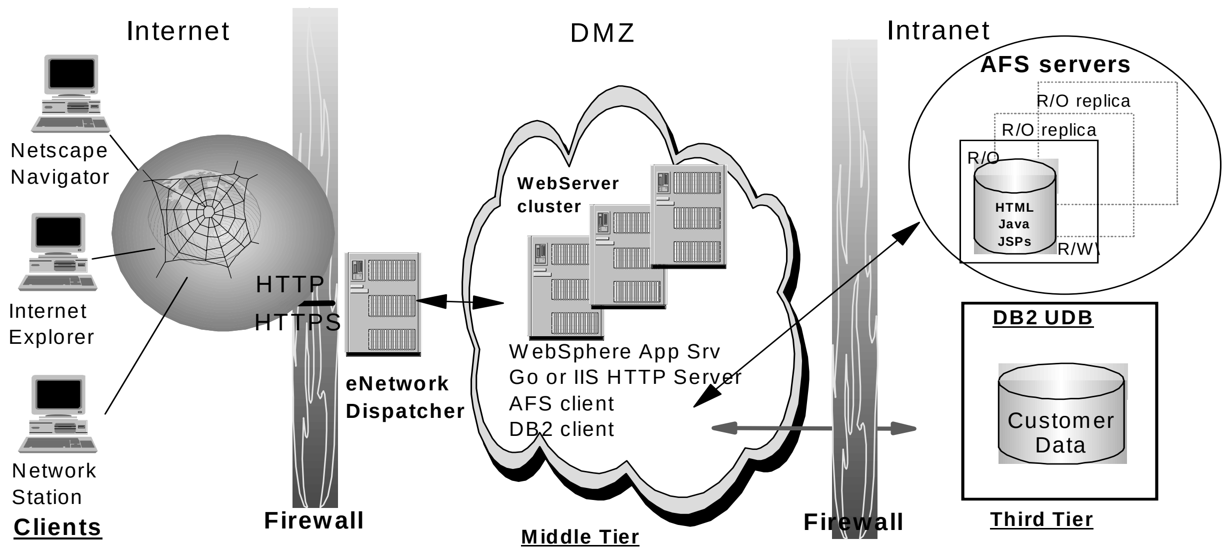

***Figure 3*: e-Connectors Internet Environment**

Application Development Environment

We developed our Insurance Company application on a machine that had the following tools: a standard editor, the WebSphere Application Server and the JDK it requires. We read through the WebSphere Application Server documentation to learn more about the Connection Manager and how it can be used in our application. We used the *IBMConnMgrTest* sample provided by WebSphere as our starting point.[2] We added several elements, such as session state tracking, Java Server Pages (JSPs), queries to the database, error handling, etc. It took us several iterations to get to the final version. After each iteration, we compiled our application with the JDK and tested our application in our test environment. This environment is shown below in *Figure 4*.

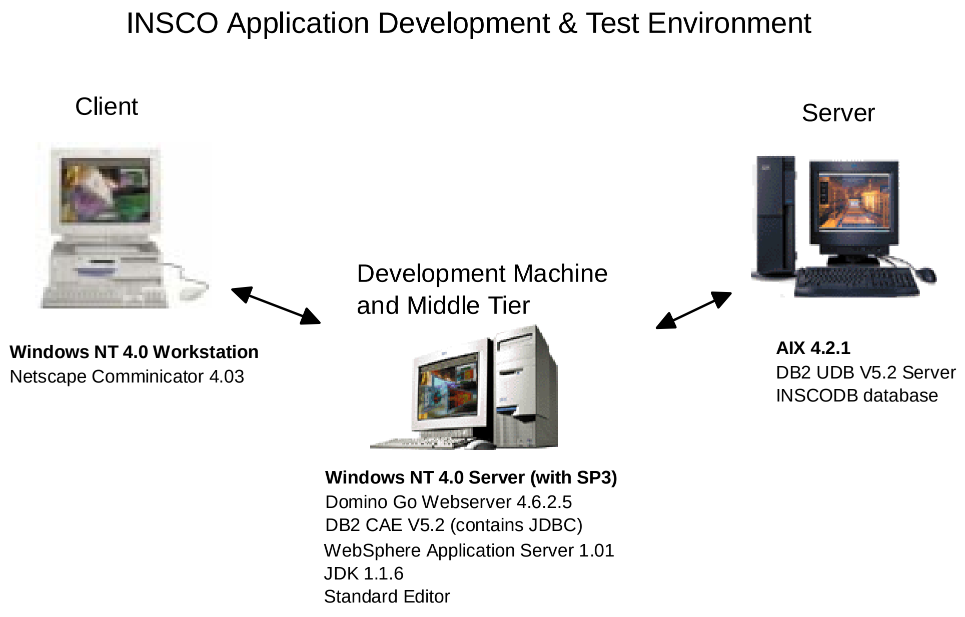
INSCO Application Development & Test Environment
Client
Server
Development Machine and Middle Tier
Windows NT 4.0 Workstation
Netscape Comminicator 4.03
AIX 4.2.1
DB2 UDB V5.2 Server
INSCODB database
Windows NT 4.0 Server (with SP3)
Domino Go Webserver 4.6.2.5
DB2 CAE V5.2 (contains JDBC)
WebSphere Application Server 1.01
JDK 1.1.6
Standard Editor

***Figure 4*: Application Development and Preliminary Test Environment**

Our development and test environment consisted of three machines: an IBM PC running Windows NT Workstation with a Netscape browser for the client, an RS/6000 running AIX with a DB2 database for the server, and an IBM PC running Windows NT Server for the middle tier. Our middle tier was our development machine as well as our test web server. Since the WebSphere Application Server was installed for compiling purposes, we just added the DB2 client in order to be able to connect to the database on the third tier. This allowed us to test that our application could successfully retrieve the DB2 data using JDBC.

[2] At the time we did our development, the WebSphere Application Studio toolset was not available. We will be using this toolset in future test phases.

WebSphere Performance Pack Overview

The IBM WebSphere Performance Pack (WSPP) helps corporate information technology specialists and Internet Service Providers (ISPs) reduce web server congestion, increase content availability, and improve web server performance. It brings together, in a single package, load balancing, caching, and web site replication. WSPP provides these capabilities by bundling the following three products:

- IBM Web Traffic Express
- IBM eNetwork Dispatcher
- Transarc Andrew File System (AFS)

The Web Traffic Express[3] is a caching proxy server that caches information, enabling subsequent requests to be retrieved locally, thereby improving access time. We did not need this component in our scenario. The two components we used were the eNetwork Dispatcher and the Andrew File System.

IBM eNetwork Dispatcher

IBM eNetwork Dispatcher is a load balancing server that allows heavily accessed Web sites to increase capacity by linking many web servers together as one logical server. The web site appears as a single IP address. eNetwork Dispatcher distributes incoming requests across the web servers based on a profile that you can customize. The "sticky" port option ensures that clients that need to preserve state are reconnected to the same server. A key benefit is that the application server returns the response directly to the client without passing back through the load balancing server. With the IBM eNetwork Dispatcher, you can add and remove servers without impacting your customers. You can switch to another machine (automatically or manually) running a standby eNetwork Dispatcher without shutting down your site or impacting your site's customers. The IBM eNetwork Dispatcher runs on AIX, Sun Solaris and Windows NT platforms. For more information go to http://www.software.ibm.com/enetwork/dispatcher/about/index.html.

Transarc Andrew File System (AFS)

Transarc Andrew File System (AFS) is an enterprise file system that provides automatic, non-disruptive, real-time replication of information across multiple AFS servers, which substantially reduces network and web server loads. Customers needing to share information on a global scale want assurance that the information is consistent across all platforms and locations, and that its access is appropriately protected no matter where they are located. The distributed file services in the WebSphere Performance Pack allow access to Web page files from any point

[3] To learn more about Web Traffic Express go to http://www.software.ibm.com/webservers/perfpack/resources.html

in the system – all with enhanced reliability, availability, functionality and security. By replicating information to a number of local and remote file servers, and then caching this information on multiple Web servers, high-volume Web sites can deliver enormous amounts of information to hundreds of thousands of users – with high performance and ensured information integrity. The combination of file replication and load balancing allows server maintenance to be done, non-disruptively, during normal working hours, providing more efficient operations and improved customer service.

Both the AFS client and web server functions always exist on the same machine. The web server interprets the HTTP or HTTPS request and asks the AFS client to handle the request. The AFS client retrieves the content from its cache or requests the content from the AFS server. When content is updated on the AFS server, the AFS server replicates the content across the all AFS file servers and notifies the AFS clients that the content has changed. WSPP ships AFS client and server support for AIX and Sun Solaris platforms, and client support only for Windows NT.
For additional information about AFS, access
http://www.transarc.com/dfs/public/www/htdocs/.hosts/external/Product/EFS/index.html.

A very useful source for information on the WebSphere Performance Pack is the redbook entitled *IBM WebSphere Performance Pack Usage and Administration*, available at http://www.redbooks.ibm.com.

WebSphere Application Server Overview

The WebSphere Application Server offers customers a solution to implement and manage e-business web sites. It lets you achieve the “write once, use everywhere” goal for servlet development. It consists of a Java-based servlet engine which is independent of your web server. This means that the Java servlets you develop on one system will run on any supported web server, on any platform. We created our application on a Windows NT 4.0 system, and deployed it on both Windows NT 4.0 and AIX 4.2.1, using both Lotus Domino Go Webserver and Microsoft Internet Information Server (IIS). There was no need to change the application for any of these environments.

The following web servers are supported by the WebSphere Application Server:
- Domino Go Webserver
- Netscape Enterprise Server
- Netscape FastTrack Server
- Apache Server
- Microsoft Internet Information Server

To learn more about the WebSphere Application Server, visit the following URL: http://www.software.ibm.com/webservers. To download a trial copy, go to: http://www.software.ibm.com/webservers/appserv/download.html

Servlets

Servlets are Java programs that use the Java Servlet Application Programming Interface (API) and the associated classes and methods. They are very similar to applets except they run on the web server, not on the browser. Because of this they have many advantages over applets. For example, you do not have to write your application for any particular browser. Since all the work is done on the server side you can return a pure HTML page to the client, and not have to require a particular level of browser. This makes servlets ideal for applications designed to run on the Internet, as you have little control over what browsers are used to access your server.

There are many other reasons to use servlets over applets and CGI programs when you are developing an application. Most are described in the WebSphere Application Server Guide, but in brief, here are a few things to consider:

- Applets can take considerably more time to download from the server to the browser. With servlets only the final HTML is transferred across the Internet.
- Servlets are written in Java, whereas most CGI programs are written in languages such as PERL, and C. Java is much more portable than either of these languages.
- Servlets can be loaded once, either at server initialization, or when they are first invoked. Thereafter they remain loaded, greatly improving performance.
- The Java Server Development Kit (JSDK) has been written expressly to facilitate the development of servlets, and is continually being enhanced with new classes and functions.

A servlet is simply a Java class file that is invoked by the web server. It is first loaded into memory, either when the web server is started, or when the servlet is first invoked by a client, depending on how you configure your web server. When a servlet is loaded, the server creates an instance of the servlet, then calls the servlet init() methods. This is the only time that the servlet's init() method is called. Any initialization for the servlet is done here.

If the servlet is invoked via a client's request, the server will create two objects specific to that request for the servlet to use: the Request Object and the Response Object. These objects are used by the servlet to exchange information with the client. All incoming information is contained in the Request Object. The servlet's response, which could include an HTML page, is contained in the Response Object. After these two objects are created the web server invokes the servlet's service() method. The service() method can, in turn, invoke other methods within the servlet, or even invoke other servlets.

The INSCO Application Details

For our application we created a servlet that was of the HTTPServlet class. This class has specialized methods for handling HTML forms, which allow for easy transference of data from the HTML form to a servlet for processing. Since our application was going to rely on HTML forms to receive input from the user, this was the logical choice. We named the servlet WASpolicyServlet, which stands for Websphere Application Server insurance policy servlet. It contains two main methods, init() and dopost().

The init() method is only invoked when the servlet is first initialized. In our case, this is when the first client request comes to the server, as we chose not to have it initialized when the web server was started. This was an arbitrary decision, but the logic was that the web server may be doing many things, and there may not be a request for an insurance policy review for some time. In this case, there was no need to load the servlet at startup time.

In our servlet there are two things that are done at initialization. First, the servlet reads a data file (*login.properties*) that contains information about the database it will be connecting to. By making this a separate file we were able to switch databases without having to alter our servlet. Along with the database properties you also need a userid and password to connect to the database. Had we wanted to use a single userid/password for the database connection, we could have stored this information in this file and had our servlet read them in at this time. Instead, we elected to have the user enter his userid/password when making the request. This accomplishes two purposes. First, it allows us to use the database server for authentication of the client, instead of needing to create our own authentication scheme, and secondly, it eliminates the need to have a database userid and password residing in a file on a hard drive. The second part of the init() method is establishing a connection to the database connection manager. This will be used throughout the life of the servlet to connect to the database.

An HTTPServlet normally has a dopost() and a doget() method. We decided to only have a dopost() method in our servlet due to security concerns. With a get method the data is sent from the browser to the web server as part of the URL and can easily be intercepted. With the post method it is encoded and sent as a separate message to the web server. In addition, the post method allows for much more data to be sent on a single request.

The dopost() method in the WASpolicyServlet has two main functions. The first is to create a session between the web server and the browser making the request. A session is a method of keeping track of who the requester is and what he has done. This is done in two parts, one at the browser, and one at the server. The server sends an identification file to the browser, known as a cookie. The browser then accepts the cookie and stores it for a predefined length of time. In all future communications with the web server, the web server examines the cookie to determine who the requesting browser is. On the web server side the server creates a session object for each session it establishes, and stores data about that session in the object. The session is created using the getSession() method. This method will check for the existence of a session with this client. If none exists, it will create one and send a cookie to the browser.

The second function of the dopost() method is to act as a traffic director for the servlet. Our application was designed so that all communication from the browser would be received as POST requests to the WASpolicyServlet. The dopost() method examines each request and routes it to the appropriate method within the servlet. The method will then handle that request and respond to the requester. The responses are in the form of JSPs (Java Server Pages). A JSP allows you to integrate regular HTML, JavaScript, servlets and even Java code (scriptlets) all within a single HTML page. In order to allow the servlet to build dynamic web pages, and to segregate the presentation design of the web pages from the business logic of the servlet, JSPs were used for most of the HTML pages. The following describes the flow of our application.

1. The insurance company's front page *(inscosec.html)* is loaded at the browser. This is simply an HTML page that anyone may access. It is served via the HTTP protocol and has a link via HTTPS to the logon page.

2. The insurance company's logon page (*inscopolicy.html*) is actually an HTML form. It is served via the HTTPS protocol (HTTP over SSL) and the user must accept the server's certificate in order to receive the page. On this page the user is asked to enter his userid and password, which will be sent via a POST to the WASpolicyServlet on the web server. Please note that whatever hostname you put in your HTTPS link will show up as the hostname in the certificate presented to the client's browser.

3. WASpolicyServlet is loaded at the web server, and the userid and password information is sent to the logon method. This information is passed on to the DB2 Server residing on the third tier. If DB2 is unable to validate the logon information, an exception is thrown and an error page is returned *(inscoerror.jsp).* If DB2 accepts the userid and password, a select statement is executed, and the customer's personal information is retrieved from the database. This information is stored in the session through cookies, and will be used by the JSPs. If the query is successful, the home page gets loaded *(inscohome.jsp).*

   In order to set up the DB2 connection, we used the Connection Manager[4] provided by the WebSphere Application Server. The Connection Manager maintains a pool of open data server connections to relational databases such as DB2. Our servlet uses the Connection Manager to request a connection from the pool. If there are no existing connections, the pool requests a new connection from DB2, then adds the connection to the pool and gives the connection to the servlet. The servlet then uses the connection to talk directly to DB2 through the JDBC driver[5]. When the servlet ends communications with DB2, it returns the connection to the pool for use by another user request.

---

[4] To learn more about the Connection Manager, please visit the following URL, http://www.developer.ibm.com/devcon/technews.htm, and look for the "IBM Developer Connection Magazine Release 2, Dec 1998" link.

[5] Please note that because we have an application, not an applet, we specified in our code to use the JDBC app driver *(COM.ibm.db2.jdbc.app.DB2Driver)*.

This same connection will be reused if future connection requests are identical to it, in other words, if the servlet is trying to access the same database using the same driver with the same userid and the same password. If another user logs on, submitting a different userid and password, a new connection is added to the pool. However, if the same user stays logged on and submits other requests that need access to DB2, the same connection is reused, thereby improving performance.

The following are the types of requests that reuse DB2 connections maintained in the pool in order to perform the queries:

- viewing a policy
- updating personal information
- undoing the changes made by the update.

These three options, as well as the logoff option, are all available from the home page. The home page is a JSP file -- at the top of the page there is a personalized welcome message. The first and last name of the customer were retrieved from the database upon logon, and placed into the session. We used a Java bean (*ABean.class*) to pass this data from the servlet to the JSP. The WASpolicyServlet instantiated this bean at the beginning of the session. The contents of this bean are used by the JSPs in the pages that are displayed.

4. When a user selects the "View Policy" button from the home page, the servlet is invoked again, and it obtains one of the existing DB2 connections from the pool. Once this connection is established, the following query is sent out to the database: return the customer's policy information based on his userid. The data is retrieved from the database, the policy details are displayed to the browser (*inscoview.jsp*), and the database connection is returned to the pool. This JSP is personalized in the same way as the insurance policy home page. Please note that any time a connection to the database has failed, an exception is thrown and the error page containing a brief description of where the error was encountered, is loaded

   Along with the policy details, the customer now has the choice of either updating his personal information, going back to the home page, or logging off. His choice is passed back to WASpolicyServlet via the post method.

5. When the user submits the request to update his personal information, the update page is displayed *(inscoupdate.jsp)*. This form contains all the fields he is allowed to update: his first name, last name, middle initial, home & e-mail addresses as well as phone and fax numbers. This information is stored in the Java bean, and used by the JSP to initialize the page. The page is an HTML form and the user can then change any fields on the form.

The user has the following options:

- make changes on the form and submit them
- make changes on the form and reset the form
- go back to the home page without making any changes
- logoff without having made any changes.

If the user chooses to reset the information, the browser just refreshes the fields with the information it had when it first loaded the update page. The next step covers what happens when he submits the changes.

6. Each of the fields from the update form has two corresponding session values: an *initial* value and a *current* value. The initial values are set during logon. These session values never change. The current value for each of the fields is updated when the user submits his changes. These updates are done first when our servlet is invoked through the "Submit Changes" button.

   Once these session values are updated, the servlet requests a DB2 connection from the pool. After obtaining this connection, a database update is performed for all of the update fields, using the *current* session values. Once the update is completed successfully, the servlet issues a DB2 commit, the database connection is released back to the pool, and the updated page is loaded *(inscoupdated.jsp)*. This page shows the same fields with the new values. It also allows for another update to be submitted, as well as an option to undo the changes.

7. *Undo Changes.* This option was included to allow the users to undo all the changes they made since they logged on, after they were submitted and committed. This method is similar to "Submit Changes". When the "Undo Changes" button is selected, the servlet is loaded and the current session values are set to the initial session values. The servlet then requests a DB2 connection from the pool, and uses this connection to issue another update to the database, as well as a "commit" statement. The values are restored to what they were when the user had initially logged on. The results are displayed as the servlet calls the *inscoupdated.jsp* page. Again, once the servlet is done with the database connection, it returns it back to the connection pool.

8. *Go Back Home*. This option allows the user to go back to the home page once he is done with the current page. Because the home page displays the user's first and last names, if he had made any changes to his name, he would be able to see these changes in the welcome message. When the "Go Back Home" button is submitted, the servlet is invoked, gets the current session values for the first name and last name fields, and loads the home page with this information.

9. *Logoff and terminate this session.* This option is available from all of the JSP pages because it is essential to security that the user logoff to end the session. The WASpolicyServlet is invoked and calls the logoff page *(inscologoff.jsp*), which displays a personalized good bye message to the user through the use of the ABean. The session is

then invalidated, which means the session's HttpSession object and its data values are removed from the system.

If the user fails to issue a logoff the session will remain active for 30 minutes from the last time the user sends a request to the web server. This is the default session length and we chose not to change it. This means that if the user does not log off, and leaves his browser unattended, anyone else using that browser within the 30 minutes will have access to all of the user's personal data. This puts the onus of security on the end user. We considered an alternative to shortening the life span of the session, but that had the drawback of potentially ending a session before the user was finished. If the session length is too short, say 3 minutes, then the user is forced to respond quickly and may make a mistake. If the session length is too long then the security exposure grows. We decided that 30 minutes was a reasonable amount of time for the user to respond, while keeping the security risk relatively low. Regardless of the session length, it is still the end user's responsibility to secure his workstation.

Discoveries and Recommendations

- You must have the WebSphere class libraries available on your development system, even if you do not plan to test you application there. This is because you will need them when you attempt to compile our application.

- Do not put your application servlets in the *servlets* directory. (i.e. *<drive>:\WebSphere\AppServer\servlets*). This is because the *servlets* directory receives special handling from WebSphere. All servlets in this directory are loaded when the server is started. For this reason, this directory is not contained in the default classpath when WebSphere is installed.

- The invalidate session method which we used will not invalidate the session immediately. The session is still valid while the servlet is running, in order to allow you to finish any processing that is needed. The session will be invalidated once you exit the servlet.

- If you create any JavaBeans, as we did, then you must put them in a directory that is in your java classpath.

- We found that if our servlet could not find a JSP that was invoked from our server via the callPage method, the message recorded in the web server error log was that the servlet was not found, not that the JSP was not found.

PIE Environment Details

Our Production Integration Environment (PIE) consists of several machines configured to represent the Internet, a demilitarized zone (DMZ) and an intranet. Firewalls are installed to keep intruders from accessing the corporate intranet. The firewall configuration for the INSCO application implements a variation of the screened subnet firewall architecture. Our firewall has a non-secure interface to the Internet, a secure interface to the DMZ, and a second secure interface to the Intranet. *Figure 5* shows a schematic of our firewall configuration.

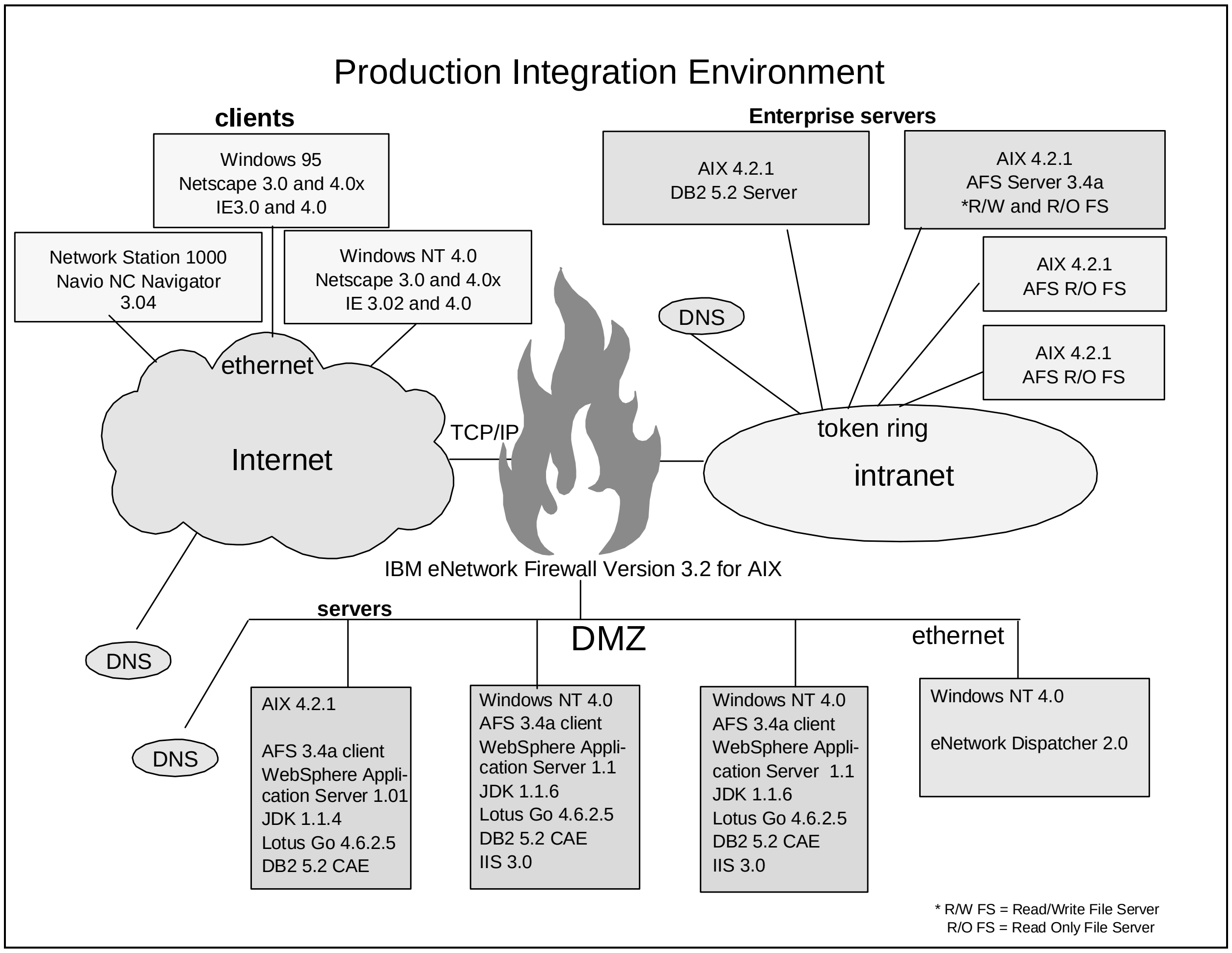

***Figure 5*: The Production Integration Environment**

The intranet's computers communicate to each other via a token ring local area network (LAN), whereas the Internet and the DMZ are run on ethernet LANs. The Internet contains the client machines. Each machine acting as a client has a web browser that supports HTTPS and cookies.

The DMZ is the middle tier. The DMZ is a network that is neither part of the Internet nor inside the private network. IP filters in the firewall between the Internet and the DMZ are designed to screen the DMZ. The IP filters allow inbound traffic, such as HTTP and HTTPS, to reach the web servers, yet still provide some protection for the computers in the DMZ. The firewall between the DMZ and the intranet provides further protection for the intranet. The DMZ consists of four machines: three of them are web servers, the fourth one does load balancing between the three web servers. The IBM eNetwork Dispatcher (eND) handles the requests as they come in, and sends them to the web servers in round-robin fashion. This way, each of the web servers have the loads distributed to them evenly (or unevenly if you so choose).

The domain name servers in both the DMZ and the Internet are updated with the IP addresses and domain name for our web servers. When clients on the Internet ask the browsers to connect to a URL, the clients usually resolve the URL to an IP address by using the DNS (Domain Name System). URLs should be registered with the Internet Service Provider (ISP) and/or registered with InterNIC so that clients anywhere around the world can resolve them and access pages on the web server.

When a request comes in through the Internet, the HTML pages are loaded on the web servers. However, these pages don't reside in the DMZ. They reside on an AFS file server on the third tier. Each web server has an AFS client, which communicates with the AFS file server to get these files. The servlet and the JSP pages are also stored on the AFS file server, but get loaded in the middle tier web server machines. The WebSphere Application Server and the JDK are installed on these machines to handle the servlets. These web servers also have DB2 clients installed. The DB2 Client comes with JDBC. The servlet communicates with the DB2 Client through this protocol, then the DB2 Client sends the database connection and query requests to the DB2 Server through TCP/IP.

The intranet is composed of the Enterprise Servers. This is where the AFS server, file servers, and replica read-only file servers reside. The DB2 Server is on its own machine and contains the insurance database. Access to this database is restricted to the users defined in the operating system. Since our database is on an AIX machine, an AIX user account has to exist for each customer.

The following sections describe the installation and configuration of these products on these three tiers.

Client Browsers

One of the goals of any Internet application is to put as few constraints as possible on those that will be using the application. This is because you have little or no control over what browsers they are using and you want to make it as easy as possible for them to access your site. This need for ease of use must be balanced with security issues. Because of our use of Java servlets and JSPs to produce pure HTML pages we were able to minimize the requirements that we put on our clients.

Our application put two minimal restrictions on the end users. The first was that their browser must support the SSL (Secure Sockets Layer) protocol. This is mandatory because of the type of data that will be flowing across the Internet. The user will be entering a userid and password, and will be requesting personal data that may be sensitive. SSL ensures that the data is encrypted and the data that is received is the data that was sent -- from the server to the browser, and from the browser to the server. Most current browsers are enabled for SSL so this is not a severe restriction. The requirement for data privacy and data integrity is a fact of life for anyone who wants to conduct any business over the Internet.

The second restriction is that the client's browser must accept cookies. A cookie is simply a file sent from the server to the browser, and stored on the browser for a specified length of time. In our case this length of time was simply the length of the session between the browser and the server. The cookie is passed back to the server each time the browser sends data. This allows the server to "remember" the client and store personal information about the client to be used for the length of the session. Most browsers have three settings concerning accepting cookies that the user can pick from. They can accept them without notifying the user, reject all cookies, or show the user the cookie and let the user decide whether or not to accept it. Because of the personalization within our application, if the user decided not to accept cookies we were forced to terminate the application.

In our testing, we used the following browsers. All were capable of enabling SSL, accepting server certificates and accepting cookies:

- Netscape 3.0, 4.01 and 4.05 on Windows 95
- Netscape 3.0, 4.03, and 4.06 on Windows NT
- Netscape 4.04 on AIX
- Internet Explorer 3.0 and 4.0 on Windows 95
- Internet Explorer 3.02, 4.0 on Window NT
- Navio NC Navigator 3.04

Discoveries and Recommendations

If you use self-signed certificates, be aware that some older browsers, such as Internet Explorer 3.0, will not allow the user to accept certificates if it does not recognize the Certificate Authority.

DB2 Setup and Configuration

We used DB2 version 5.2 for our application. The DB2 Universal Database provides reliable data management and allows you to access data on multiple platforms. Our environment consists of AIX and Windows NT machines on the middle tier, with our data residing on an AIX DB2 Server in the third tier. The middle tier machines each have the DB2 Client installed.

Both the DB2 Server and the DB2 Client have JDBC support integrated into the product. You don't need to install any other software for it. Let's start with the client installation and setup. The DB2 CAE (Client Application Enabler) is very easy to install. You can keep the default values as you go through the installation panels. There is an option you can select in order to remotely administer your DB2 Servers from your client machine. We left that button unchecked, because we did not wish to do so. Our middle tier machines are the web servers, and will be used for running our servlet and handling the browser requests. We don't want to be administering anything other than our web servers on those machines.

There is a little more work involved in setting up the server. The main steps are the following:

1. Install the DB2 Server
2. Create the database
3. Populate the database
4. Set up userids who will be accessing the database
5. Set up views to limit their access
6. Grant the proper authentication to the userids.

The following section details these steps for configuring the DB2 Server on an AIX machine.[6] The steps are similar if you are installing the DB2 Server on a different operating system — the only difference being the operating system specific installation process.

To install the DB2 UDB Server for AIX, you need to run *db2setup*. The installation is fairly straightforward. From the options, you need to select the DB2 Enterprise Edition[7] which contains both DB2 Server and DB2 Connect. We don't need to use DB2 Connect, as it is used for connecting to mainframe systems. During the installation, you will need to provide names for the DB2 instance. When naming users, groups or instances, make sure you comply with the DB2 naming conventions. These are all explained in the Quick Beginnings books. The main restriction to note is that the number of characters in any DB2 User Id cannot exceed eight (8). Your instance name will therefore be limited to eight characters, such as *db2inst1*. Please refer to the *DB2 Quick Beginnings* books for more information on how to install DB2.

[6] Please note that because we had an existing DB2 database with all of our customers' information, we did not have to do steps 2 and 3. Instead, we restored our database on the new server.

[7] If you are installing the DB2 Server on NT, selecting the Workgroup Edition is sufficient for the purposes of this type of application (it does not contain DB2 Connect).

Once you've completed the installation, you can log onto your instance by issuing the following command, "*su - db2inst1*", where *db2inst1* is your instance name. The very first thing you should do is type the following from the */sqllib* directory: ". *.db2profile*". This will set up the environment variables necessary to run DB2. Once this is done, you can invoke the Command Line Processor (CLP) by typing "db2" from the shell. From the DB2 CLP, type the following command to create a database called inscodb (for insurance company database): "*create db inscodb*". The creation of the database will take a few minutes. Once the operation is completed, you have an empty database and you need to populate it with tables and data.

Figure 6, depicts the INSCODB database schema. There are four tables that store the insurance company information: agent[8], clients, spolicy and policydb. The agent table contains the insurance agent information, the clients table contains the customers' personal data, the spolicy table contains the insurance policy details, and the policydb table keeps track of which agent handles which policy, and which customer that policy belongs to. Each table has a primary key, denoted by "PRI", that uniquely identifies a column of data. The foreign keys(fk1, fk2, fk3) are references to the data values in the other tables. They show the relationship between the tables.

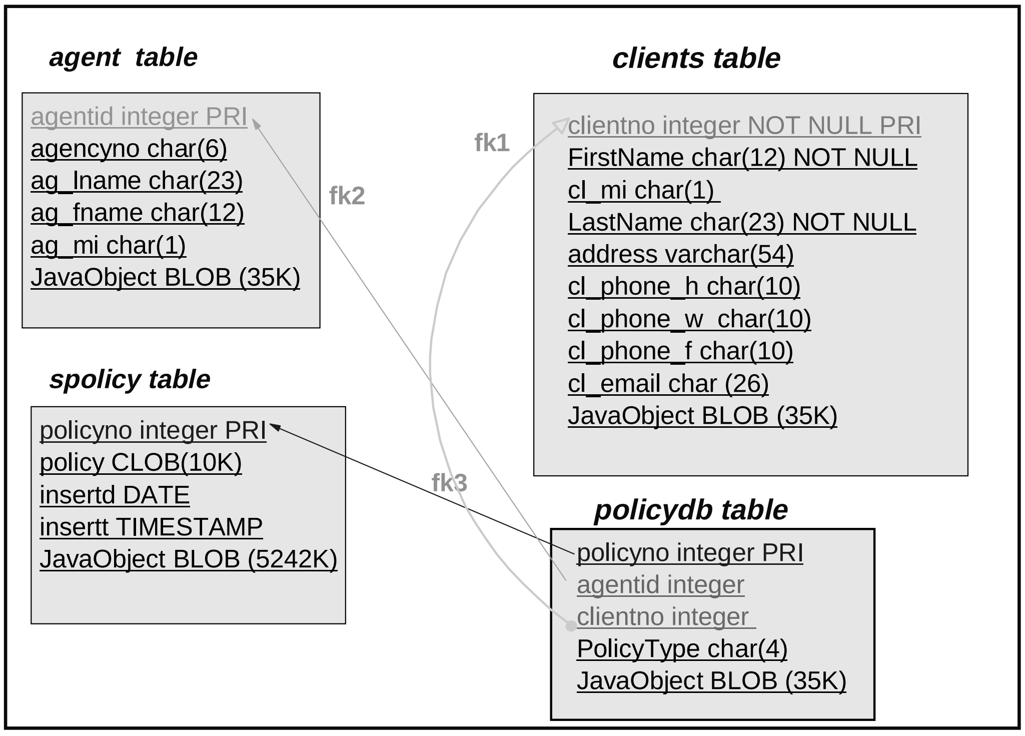

***Figure 6*: INSCODB Database Schema**

[8] Please note that the INSCODB database was created long before we designed this application. We are connecting to existing data, and did not make use of all the parts of the database. For instance, the agent table was used in previous applications, but not in the current one.

Our database contains entries for 9,518 customers. We will be giving each of those customers access to their own data when they log on through the Internet. We are using DB2's security for controlling access to this information. DB2 relies on the underlying operating system's security. Because of this, we will need to create userids for each of these customers on our server. To better protect the data residing in our DB2 database, we will be creating views for each of these userids.

We used the "SMIT" tool on AIX to create the local userids. We set the following restrictions in "SMIT" for each userid:

- user can log on locally
- user cannot log on remotely
- lock account

We wanted to create accounts that our application could access, but that users could not log onto. Since these accounts are locked, users cannot log onto them through telnet, ftp or even locally. However, because we specified that the user can log on locally, DB2 allows this userid to access the database.

Next, we created views for each userid and defined specific privileges for each userid. The use of views and privileges allow us to limit a customer to doing updates in the table containing his personal data, but not update the policy himself. You can create stricter views if you want to give your customers less privileges. The folowing commands were needed for each userid so that whenever they access the database, they can only view their own data. You can run these commands through the CLP:

```
create view insco.sel1 as select * from insco.clients where clientno=1
create view insco.pol1 as select * from insco.spolicy where policynumber in (select policynumber from
                                                   insco.policydb where clientno=1)
grant select on insco.sel1 to user """1"""
grant select on insco.pol1 to user """1"""
grant update on insco.sel1 to user """1"""
```

We based each person's userid on what their client number was in our database. So for client number one, we had a userid '1'. Please note that DB2 does not claim support for userids that begin with a number, so don't use them in your applications.[9] We were not aware of this restriction when we wrote our application. We were able to get this to work through some workarounds.

In order to create views for 9,518 customers, it was necessary to write a shell script which would run on our AIX Server. We named the script *inscoviews.sh*, and ran it from our db2instance. The contents of the script can be found in *Appendix G*.

[9] In the Appendix of the IBM DB2 UDB Quick Beginnings book, it is explained that all names must begin with one of the following characters: A through Z, @, # or $.

With all these steps completed, our database is ready to be accessed through our INSCO application.

Discoveries and Recommendations

If you have different levels of DB2 on the client and server, you need to get the latest updates in order to get the JDBC support in synch. For instance, if you have version 5.0 of the DB2 Server, and version 5.2 of the DB2 Client, you need to apply the latest fixpak onto your server. You can obtain the fixpaks from the following web page:
http://www.software.ibm.com/data/db2/db2tech/indexsvc.html

Installing and Configuring Lotus Domino Go Webserver

We installed Lotus Domino Go Webserver (known as Go Webserver) version 4.6.2.5 on two of our Windows NT 4.0 Servers, and on our AIX server. We elected not to install the Java Servlet component because we planned on using the WebSphere Application Server (WAS) for our Java support. WAS is a complete replacement for the Java Servlet component of Go Webserver. If you have already installed the Go Webserver with the Java Servlet component you must uninstall the Java Servlet component before installing WAS. We also elected not to install the Webserver Search Engine (also known as NetQuestion) that is included with Go. There would have been no harm in installing it, but since our web site was small, and we were not planning on introducing a search function, there was no need for it.

For the actual installation of Domino Go Webserver we ran the *Setup.exe* included with the product. This runs the InstallWizard that will guide you through the install. We elected to take all the defaults, except that we installed the product on the E: drive instead of the C: drive. We left the default *keyfile* parameter alone, as we were planning on changing it after the installation, and, as required, we chose a userid and password for web server administration. The installation of the Go Webserver was very straightforward and took less than ten minutes.

After the Go Webserver was installed we customized it for our application. The major change that we made was to change the PASS rules that control what pages are served by the server. There are two ways to do this, either via the Configuration and Administration forms, or by directly editing the *httpd.cnf* file. We chose to edit the *httpd.cnf* file directly. If you would rather use the Configuration and Administration forms you can find the link to them on the front page of the server. To access this page you simply need to go to a browser and enter the URL:

http://hostname

where hostname is the name by which your server is known. On this page there is a link to the Configuration and Administration form. This page is password protected, so you will need the userid and password that you supplied when you were installing the Go Webserver The "Request Routing" button under the "Request Processing" label will allow you to update the PASS rules for the server. Complete instructions can be found in the *Webmaster's Guide*, which is provided with the product.

The *httpd.cnf* file is found in the *WINNT* directory. This is the file that contains most of the configuration parameters for the Go Webserver. We made the following changes to the PASS directives within this file:

```
Original:     .
              .
              .
Pass /Admin/*.html e:\WWW\Admin\*.html
Pass   /* e:\WWW\HTML\*
```

Changed:

```
                .
                .
                .

Pass /Admin/*.html e:\WWW\Admin\*.html
Pass /base/*    e:\WWW\HTML\*
Pass   /*           x:\.pie\insco\*
```

As you can see, we changed the default page (the last one in the list of PASS statements) to no longer point to the e:\WWW\HTML directory, but instead, to point to the homepage of our application. Since our application has been stored on an AFS server, the actual files reside there, not on our server. The AFS client on each web server machine will link the AFS cell as the x: drive. This means that the user will only need to type the URL of **http://hostname** to retrieve the homepage. The filename of the homepage must match one of the filenames listed in the Welcome directive. In order to preserve the access to the default Go Webserver home page we added the Pass /base/* directive. This means that if we use the URL **http://hostname/base** we can access the home page, which contains the link to the Configuration and Administration Forms.

We made three other changes to the *httpd.cnf* file. First, we changed the value of the "logtogui" directive from off to on. This directive tells the web server whether it should write each log entry to the GUI, as well as logging it to the appropriate log file. For performance reasons you would want to leave this set to "off" during production, but while you are testing your server it is very convenient to be able to see each log entry as they occur. You can set this via the Configuration and Administration form via the Global Log File Configuration Settings link. The second change was to add our keyring via the *keyfile* directive. The keyring is required if you want to use Secure Sockets Layer (SSL) when transferring data from the server to a browser. Once again, this can be done from the Configuration and Administration form via the Security section. Finally, we added a "Welcome" directive for the file name **inscosec.html**. This means that this file will be served automatically when a user enters the URL **http://hostname**. Note that the Welcome directive is case sensitive. You must match the case of the file exactly.

Once we had made all of our changes we shut down our server and restarted it. The default installation of the Go Webserver will install the server as a service, and set it to automatically start at system startup time. This is normally what you would want, but during our installation and testing phase we wanted to be able to control when the web server would start. To accomplish this we went to Settings - Control Panel - Services and set Startup to Manual. When we were ready to run in production mode we changed this back to Automatic.

For the AIX server we also used Lotus Domino Go Webserver 4.6.2.5. We followed the default installation path, once again not installing the Java Servlet Component or the Webserver Search Engine. The same changes were made to the configuration file as with NT. Due to the different directory structure with AIX, as compared to NT, the changes in the PASS look like this:

```
Original:
                .
                .
                .
Pass        /*        /usr/lpp/internet/server_root/pub/*
```

```
Changed:
                .
                .
                .

Pass        /base/*        /usr/lpp/internet/server_root/pub/*
Pass        /*           /afs/pie/insco/*
```

It should be noted that the AFS directory pointed to by the AIX Go Webserver is the identical AFS directory pointed to by the Windows NT Go Webserver. The differences in the PASS directives are due to the differences in how the AFS client links to the AFS server.

As with Windows NT, the *keyfile* directive was changed to point to the keyring that was to be used (the same keyring that was used with the Windows NT servers) and the "Welcome" directive was changed to allow *inscosec.html* to be the default file served. There is no "logtogui" directive with the AIX Go Webserver.

Discoveries and Recommendations

- When you start the Domino Go Webserver on AIX, make sure to specify the db2instance name in the startsrc command. Our startsrc statement looked like this:

  startsrc -e "DB2INSTANCE=db2inst1" -s httpd

- It is possible to use just one configuration file (*httpd.cnf*) for all of your web servers, if they are all of the same type and on the same platform. We did this for our Go web servers on NT. We placed the configuration file in our AFS cell and then started our web servers using the -r option, which allows you to specify the location of the configuration file (the default is *httpd.cnf* in your *WINNT* directory). Since our web servers were configured identically, with all files, including the keyring and stash files, in identical directories on each machine, and because we were using eNetwork Dispatcher they all had the same hostname of **www.insco.dmz.ibm.com** only one configuration file was needed. This means that if you make any changes to you web server configuration file you only need to update one file.

Installing and configuring Microsoft's Internet Information Server

On our two Windows NT 4.0 servers we also installed Microsoft's Internet Information Server Version 3.0. Windows NT 4.0 only comes with IIS 2.0, but IIS 3.0 is contained in Windows NT 4.0 Service Pack 3, which we had applied to our systems. For the installation we took all the standard defaults and the product installed with no problems. We then configured our IIS server to function similarly to our Go web servers.

The first thing we did was to change the default so that the World Wide Web Publishing Service (the IIS component that acts as a web server) did not start automatically. In a production environment you would not want to do this, but for our test purposes we wanted to be able to control when the web server started. This was done via Settings-Control Panel-Services and setting Startup to Manual. Once you have done this you can still start and stop the WWW service from Microsoft Internet Manager which is located on the Start button

We designed our application so that we could switch our Internet servers between Go Webserver and IIS for testing purposes. In order to facilitate this we configured both web servers to serve our application from the same AFS directories. For IIS we made the following two directory changes:

1. Changed the default home directory from:

```
D:\InetPub\wwwroot
```

to:

```
X:\pie\insco
```

2. To continue to allow access to the Microsoft default page we added the following directory statement:

```
D:\InetPub\wwroot    /iisbase
```

WebSphere Application Server Details

The following details the steps we followed in setting up the WebSphere Application Server (WAS) on Windows NT 4.0. Make sure you already have one of the supported web servers installed on your machine before installing WAS. Note, if you plan to use WAS with more than one web server on the same machine you must have each web server you plan to use already installed on that system before you install WAS. This is because part of the WAS installation consists of modifying the web server configurations files for WAS and can only be installed one time per machine.

1. Begin the Installation of the WebSphere Application Server. If you downloaded WAS from the website you need to run *ibmwebas.exe* which will invoke the InstallShield wizard. If you are installing from the CD-ROM, then the autorun executable will immediately start up the InstallShield wizard for you. If you have autorun disabled on your system, simply run the Setup.exe from the CD-ROM. You will then be prompted to select which components of WAS you wish to install. We chose to install all the components of WAS. This is the default.

2. The installer detects which levels of the JDK you have installed, and will ask you to choose which one you want WAS to use. It is strongly recommended to install the JDK 1.1.6 for the WebSphere Application Server. There is an "install JDK 1.1.6" button you can choose for this, as JDK 1.1.6 is shipped with WAS. (Note: if you are installing on AIX you should use JDK 1.1.4, which is included with WAS).

   Select the plugins you will need. Plugins are components of WAS that are specifically developed for each web server supported. We chose "Lotus Go 4.6.1 or greater" and "IIS 2.0/3.0". Be aware that if you plan on running Websphere with multiple servers you must choose all plugins now. You must also have the web servers installed at this time. Select the directory in which to install the code. We chose the "E:\Websphere" directory. Reboot when the install process is complete.

3. When you have rebooted your machine, make sure your web server is running. You will need it so that you can configure WebSphere Application Server through the administration GUI. The browser you use can be on the same machine as your web server, or on a different machine. From your browser, you can load this GUI by going to your server's port 9090. For instance, if your server is www.insco.dmz.ibm.com, you will want to go to **http://www.insco.dmz.ibm.com:9090**. Make sure that your browser supports JDK 1.1.

4. When you load the administration page, you first get to a logon screen. As is noted on this page, you need to type "admin" for both your username and password. You can change this username and password when the Server Manager page is loaded, through the "Properties" button. We strongly recommend that you do this, as all copies of WAS are shipped with the same default userid and password. When you are done with the changes, select the "manage" button.

5. To configure the WebSphere Application Server to work with this application, choose "Setup — Basic". You are given a choice of using the System Classpath or having WAS use its own. The default is no, do not use the System Classpath, and we strongly recommend that you keep this default. It is much simpler and safer to control where WAS is looking for its class files if you do not use the System Classpath. The Java Classpath statement was set at installation and is shown on this panel. For our application we needed to allow WAS to be able to access the DB2 class files that we will be using in our application. We added *E:\sqllib\java\db2java.zip* to the Java Classpath for this purpose[10]. We also needed to give the web server access to our servlet and our JavaBean. Since they were both stored in the AFS directory, we also added that to the Java Classpath. It is recommended that you do not store your servlets in the WebSphere *servlets* directory. Each time a servlet in this directory is invoked, it is reloaded, which ends up affecting your performance.

6. In order to be able to provide session tracking in the servlet you must configure WAS to support this. In "Setup — Session Tracking", in the "Enable" tab, make sure "Enable Sessions" and "Enable Cookies" are set to yes. They are both set to this by default at installation. In the "Cookie" tab, pick a cookie name (we picked inscosession). Leave "Domain" blank. Also, leave "Maximum Age" blank This field is used to determine how long the cookie will reside in the clients browser. A value of null will ensure that the cookie only lives during a single browser session. We left the Path field blank, but you could use it to restrict your cookies to certain URLs. In the "Intervals" tab, leave the invalidate time to 1800000 (30 minutes). This increases security when a user walks away from an open browser session — the session will be invalidated after 30 minutes of inactivity. In the "Persistence" tab, we left "Persistence" set to "Yes" so that if the server goes down during a session, when it comes back up the session would still be active. The end user won't notice that the server had a momentary interruption.

   Make sure to save your changes by using the "SAVE" button before closing the Server Manager pop-up, and in between changing tabs to alter other configuration parameters.

7. One additional configuration change we made for testing and debugging purposes was to enable the WAS GUI. This can only be done by editing the *jvm.properties* file. This properties file can be found in the *\WebSphere\properties\server\servlet\servletservice* directory. We made the following changes to this file:

```
ncf.jvm.stdoutlog.enabled=true          # This enables/disables logging from WAS
ncf.jvm.stdoutlog.popup=2               # This enables the WAS GUI.  The only
                                        allowable value for this is 2
ncf.jvm.stdoutlog.file=false            # This enables or disables logging to a file
ncf.jvm.stdoutlog.filename=e:\WebSphere\AppServer\logs\ncf.log  # Filename of
                                                                 the WAS logfile
```

[10]For our AIX machine, to configure WebSphere to work with DB2, we needed to add two things on this panel: 1) the Java Classpath *:/usr/lpp/db2_05_00/java/db2java.zip* and 2) the Java Libpath, *:/usr/lpp/db2_05_00/lib*.

The end result is that all output from your servlets will be written to the WebSphere GUI, but not to a log. This is very useful when you are writing and debugging an application. You can also perform monitoring and tracing from this GUI. Once you are ready to put your application in production mode you should comment out the *ncf.jvm.stdoutlog.popup* directive and change the *ncf.jvm.stdoutlog.file* directive to true for performance reasons. Our production directives were:

```
ncf.jvm.stdoutlog.enabled=true
#ncf.jvm.stdoutlog.popup=2
ncf.jvm.stdoutlog.file=true
ncf.jvm.stdoutlog.filename=e:\WebSphere\AppServer\logs\ncf.log
```

At this point you have finished setting up the WebSphere Application Server. Some sample applications are included with WAS for you to try. The simplest one to run is the HelloWorldServlet. Running this servlet is a quick and easy way to test if your web server is configured properly to handle servlets. Simply go to a browser and enter the following URL:
http://hostname/servlet/HelloWorldServlet
where hostname is the hostname of your web server. When entering a servlet name on a URL, pay attention to the uppercase letters because the servlet name is case sensitive. To try some of the other sample servlets that are included with WAS, follow the instructions in the "Application Server Guide" included with WebSphere.

Discoveries and Recommendations

- If you are using Netscape 4.03 or 4.04 to administer WAS then you must install the JDK patch to your browser. This can be found at: http://developer.netscape.com/software/jdk/download.html.

- If you have two different levels of WAS installed on different servers, you may have a problem with WAS administration. The problem only occurs if you try to administer a WAS 1.01 server from a browser that resides on a machine that has WAS 1.0 installed. The *jst.jar* file for 1.0 conflicts with the Java applet that runs the WAS administration for 1.01. You could temporarily remove the *jst.jar* file from your classpath while you are administering the WAS 1.01 server, but the best solution is to not attempt to administer WAS from a server that has a different level of WAS installed.

- In order for servlet support to function the directory that contains the WAS plugins must have execute authority. The default directory on Windows NT is *<drive>:\WebSphere\AppServer\plugins\nt*. When WAS is installed this directory is created with execute authority, so unless you change it, or it already existed, this should not be a problem.

- You should be careful if you start your web server from an MS-DOS command window. This can affect the classpath, as the current directory when you start the server becomes part of the classpath. This can sometimes be useful during development, but can lead to unexpected results if you are not aware of what directory you are in when you start the web server.

Migrating WebSphere Application Server to 1.1

During our test, WebSphere Application Server version 1.1 became available. We decided it would be a good test of our application, and of WebSphere's migration abilities, to try and migrate our systems from WAS 1.01 to WAS 1.1. To test compatibility between WAS 1.01 and WAS 1.1 we left our AIX system running WAS 1.01 and migrated our two Windows NT systems to WAS 1.1.

The migration process for WebSphere Application Server involved uninstalling the previous release, and then installing the new release. Before uninstalling you should copy any files in the WebSphere directory that you have modified. The most likely examples are the *jvm.properties* file, which is in the *properties* subdirectory, and any files you may have placed in the *classes* subdirectory. Since we had moved all of our application files to the AFS server, the *jvm.properties* file was the only file we needed to back up. The uninstall of WAS 1.01 appeared to go smoothly, however, after it had completed we realized that there had been a problem WAS had not detected. As we mentioned earlier, we placed our Domino Go Webserver's configuration file (*httpd.cnf*) on the AFS server. This allowed us to have one configuration file for all our Windows NT Domino Go web servers. This worked fine for most things. However, since this directory was a read only directory, the uninstall of WAS 1.01 could not make changes to this file, and therefore could not properly perform the delete function. We forgot to move the configuration file over to each individual server before performing the uninstall of WAS 1.01. We then had to copy the *httpd.cnf* file we had saved from before we originally installed WebSphere into the *WINNT* directory. This was necessary because the installation of WAS 1.1 would be making updates to this file.

The installation of WAS 1.1 went very smoothly, with no problems. After the installation we moved the *httpd.cnf* file over to the AFS server, updated the new *jvm.properties* files to match the ones we had saved before uninstalling WAS 1.01, and restarted our application. There was only one small problem, which is explained in the Discoveries and Recommendations section below.

Discoveries and Recommendations

Configuration File

WebSphere currently assumes that the configuration file for your Domino Go Webserver is in the default directory (on NT, it is the *WINNT* directory). During an uninstall, if it cannot find the directory it will not give an error message, but will create an empty *httpd.cnf* file in the *WINNT* directory. Also, at installation time WAS makes the same assumption, that you are using the default configuration file and it is located in the *WINNT* directory. If you are using a differently named configuration file, or you have placed it in a different directory, you should make a copy of it, name it *httpd.cnf* and place it in your *WINNT* directory for the installation of WAS. After the installation is complete you may move and rename this directory. Do not forget to update the *jvm.properties* file with the new name of the Domino Go Webserver configuration file.

JSP meta tags
We ran into one problem with our JSP files when we migrated to WAS 1.1. On Netscape 4.x browsers, the first time we loaded a JSP, the Netscape browser gave the message that the cached copy had expired. After some investigation we found that the meta tags we had placed in our header file were causing the problem. We originally had:

```
<META HTTP-EQUIV="pragma" CONTENT="no-cache">
```

and this worked fine with WAS 1.01. However we needed to change to

```
<%
response.setHeader("Pragma","No-Cache");
response.setDateHeader("Expires",0);
response.setHeader("Cache-Control","no-Cache");
%>
```

in order to get the application to run properly with WAS 1.1. We tested this Java code in our JSPs with WAS 1.01, and it worked correctly there, also. We recommend that for all JSPs you use the above Java code in between your <HEAD> and </HEAD> tags to avoid any problems.

Putting the Application on AFS

During the development stage we did not use AFS as a repository for our application. The servlet, bean, HTML and JSP files were stored and served from the same machine as the web server. This was done for two reasons. The first was that it eliminated one factor during debugging, the second being that it made making updates to the application files much simpler. In addition, during development of the application we only used one web server, so there were no benefits to using AFS. Once we were satisfied with our application it was time to deploy it across multiple web servers, and to keep them in synch we used our AFS server. This meant we only needed one copy of each file and we were guaranteed that each web server would access and deliver identical content.

The details on how to install and configure the AFS server and the AFS client are explained in the section entitled "Transarc Andrew File System (AFS) - File Sharing System". The current section will just cover the steps necessary to place our application on AFS, and to make it available to our web servers. It assumes that you have installed the AFS client on each machine that will serve as a web server.

Our AFS server was configured with a cell named "pie", to contain our application's files. In the pie cell we created an *insco* directory, and added a *servlet* and a *gocnf* directory off of the *insco* directory. We placed our HTML and JSP files in the *insco* directory, our servlet and bean files in the *servlet* directory and our common *httpd.cnf* file in the *gocnf* directory. Because AFS uses Kerberos security, we needed a two step process to place our files there. The first step was to ftp to the AFS server machine using a userid and password known to the AIX machine where the AFS server was installed. Using the "put" command of ftp we placed our files in a temporary directory we had created. The next step was to telnet to the AFS server machine, log on once again with a known userid/password, then log onto the AFS server with the "admin" ID and the proper password. From there we were able to copy all of our application's files from the temporary directory where we had stored them to their proper directories in the pie cell. Once they were copied we executed the following two commands to make them available to the AFS clients:

vos release root.cell
vos release root.afs

Discoveries and Recommendations

- If one of the AFS clients loses its connection to the AFS server, eNetwork Dispatcher has no way of discovering this. It will think that the web server is still up and functional, so it will pass client requests to that server. Since the web server responds directly to the client, eND will not know that the file could not be found. The client will get the "file not found" message at his browser. However, if the client then repeats the request, eND will route the request to another web server and it will be fulfilled. The more web servers you have running with eND the less likely it is that a customer will get repeated "file not found" failures, as each request will most likely be routed to a different web server, assuming that the weights of the web servers are evenly distributed. Currently we know of no good way to monitor each AFS client to detect this problem.

- We found no problems with putting our configuration file for the Go Webserver on the AFS server. However, when we placed the keyring and stash file there we were not able to get SSL to function. We recommend you leave the keyring and stash files on the same machine as the web server.

- Because Microsoft Internet Information Server does not have an externalized configuration file we were not able to use AFS to keep our IIS web server configurations in synch. We were able to store our application files on AFS.

- You must be sure to add the directory containing your servlets and Java beans to the WAS javaclasspath. In our case we added *x:\pie\insco\servlets* to our javaclasspath using the WAS Administration GUI.

Establishing a Secure Sockets Layer for Encrypted Transactions

In the customer logon process of the INSCO application, the userid and password entered over the Internet are sensitive data that need to be protected. Neither the customer nor the insurance company want this data accessible to any other party. For data such as this, an additional layer of security is needed. Secure Sockets Layer (SSL) provides this additional security. It is used by applications to ensure that transferred data is encrypted to provide privacy and checked for integrity — ensuring that the data received is in fact the data sent.

SSL requires that the server in the client (browser)/server interaction has a secure certificate. Optionally, the client may also be required to have a secure certificate. In our scenario we required only the server to have a secure certificate. A secure certificate, also called a digital certificate or digital ID, identifies you to someone who needs proof of your identity. It is the digital equivalent of a driver's license, passport, or employee badge. Secure certificates can be issued to a server, a client, or both. Over a network, a secure certificate establishes the identity of a client or server machine. To do business over the Internet, a secure server certificate may be obtained from a Certificate Authority such as VeriSign. To do business within a company, a self-certified certificate may be used or the company can create its own internal certificate authority which issues secure certificates to be used within the company

In a cluster of web servers, the physical servers in the cluster are acting as one logical server, sharing the same certificate. Therefore, when the network dispatcher balances the workload, the security system behaves the same way and the interface is transparent to the web client — regardless of the distribution of work among the servers. See *Figure 7*: Setting Up eND and Web Server Clustering.

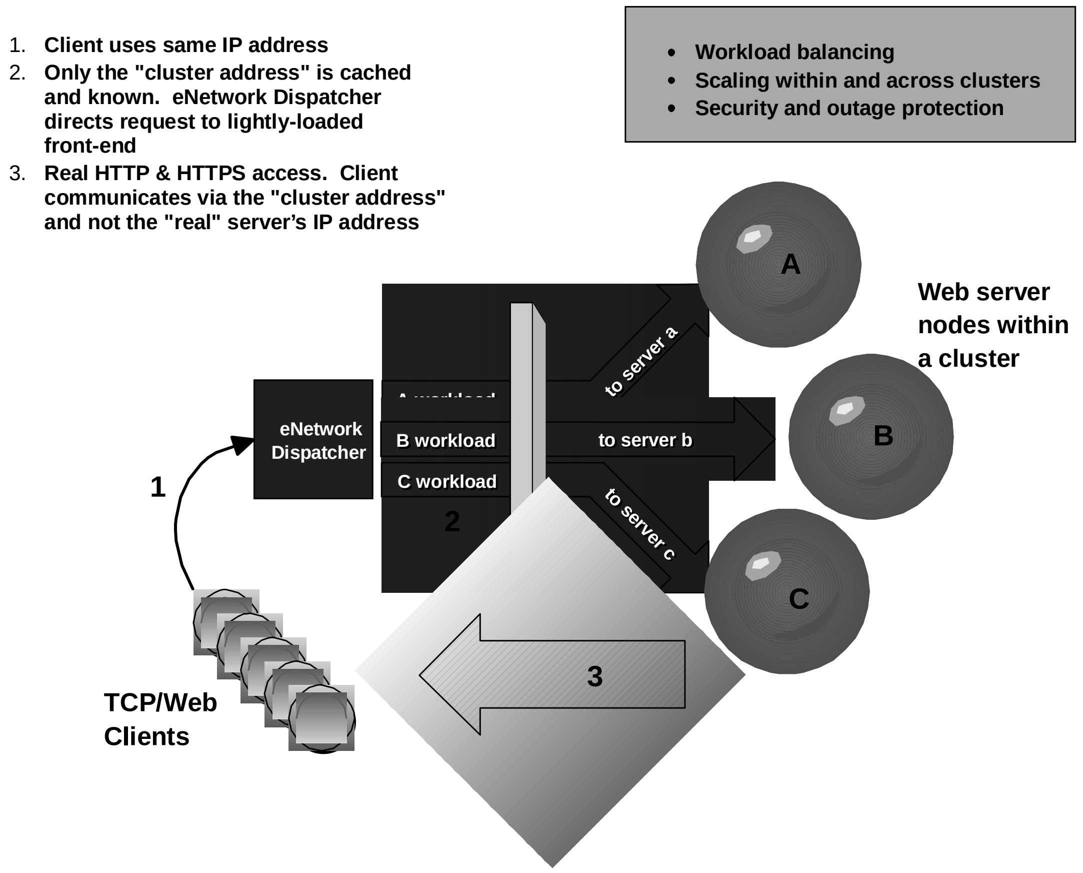
1. Client uses same IP address
2. Only the "cluster address" is cached and known. eNetwork Dispatcher directs request to lightly-loaded front-end
3. Real HTTP & HTTPS access. Client communicates via the "cluster address" and not the "real" server's IP address
• Workload balancing
• Scaling within and across clusters
• Security and outage protection
A
B
C
Web server nodes within a cluster
eNetwork Dispatcher
A workload
B workload
C workload
to server a
to server b
to server c
1
2
3
TCP/Web Clients

***Figure 7*: Setting Up eND and Web Server Clustering**

We tested with a cluster of Lotus Domino Go Web servers 4.6.2.5 and then with a cluster of Microsoft Internet Information Servers (IIS) 3.0. The Domino Go Web servers were installed on two Windows NT 4.0 servers and on one AIX server running AIX V4.2.1. The IIS web servers were installed on two Windows NT 4.0 servers. In both cases the web servers were configured to be one logical server sharing the same key pair and secure server certificate.

SSL requires a secure server certificate (a.k.a., secure server id or digital certificate) for each web server (e.g., Lotus Domino Go, Microsoft Internet Explorer, Apache). If the web server runs on more than one platform, then the secure server id will work on both platforms. For example, the Lotus Domino Go Webserver runs on both AIX and Windows NT platforms. The identical Secure Server ID installed on the Lotus Domino Go web servers on both the Windows NT and AIX platforms will work in an SSL session with web browsers.

The steps we followed to create a key pair and secure server certificate for Domino Go web servers were:

1. Install the web server as a service, causing the web server to run in the background, which uses fewer system resources. Running the web server in the background enhances performance because there is no GUI to be constantly updated. In most of the Domino Go Webserver installations, the web server is installed as a service.

2. Open a web browser. Disable caching and proxy services on the browser while the browser is used for the web server configuration.

3. Use the browser to access the front page of the Domino Go Webserver by entering **http://hostname** .

4. Click on Configuration and Administration forms to configure security for the Domino Go Webserver. You will be prompted to enter the username and password of the web administrator.

5. Scroll down to the Security heading and click on Create Keys.

6. On the Create Key and Request Certificate screen, decide if you want to request a certificate from VeriSign or another Certificate Authority (CA). We chose another CA, so we clicked on Other, then clicked the Apply button. On the Other Certificate screen, specify a key name and key ring name. Also specify a key size. The larger the key size the more difficult it would be for a hacker to compute your key. However, the larger the key size the longer the time it will take to use that key in the encryption and decryption processing. We chose a key size of 1024 for our scenario testing. Specify a key ring password. Check Automatic Login on the screen. This saves the password in a stash file. In our scenario, the Go Webserver is installed to run as a service. In order to automatically start the secure web server, the password must be stashed. Scroll the screen down to the Request Certificate heading.

   The Request Certificate portion of the screen asks for information required to obtain a secure certificate for the web server, or in our case the three web servers grouped to represent one logical web server. For this certificate, the 'Server name' is the URL of the web site, **www.insco.dmz.ibm.com**. The 'Server name' resolves to the 'cluster address' of the IBM eNetwork Dispatcher. **www.insco.dmz.ibm.com** should be specified as the server name during the installation of the Go Web servers. It is important to use the exact name. If the server name in the certificate does not match the server's name, some versions of the Netscape and Microsoft browsers will display a screen warning the user that the certificate presented by the server does not contain the correct site name. While the user may choose to continue with the connection, it is disconcerting to unnecessarily see this warning screen.

   Fill in the e-mail address field. Based on the requirements of the CA that you choose to sign your server certificate, choose whether or not the certificate should be mailed. Fill in the Save Copy field. We used a unique file name *inscokey.txt*. Click the Apply button. Upon completion of the Other Certificate screen, the file that you just specified will contain the certificate signing request (CSR). The CSR may be delivered to a certificate authority to generate a digital certificate that uniquely identifies the web server. The CA will determine that the party making the request has a legitimate claim to that request, then create and send the digital certificate to the requester.

Optionally, you may use a self-signed certificate. To do this, don't send the CSR to a CA and skip step 7 below.

7. When the digital certificate is received from the CA, copy it to the sub-directory where you created the CSR.

8. From the Configuration and Administration forms, scroll down to the Security heading and select Security Configuration. Choose the key ring that you want to work with and click on 'Set selected key ring as current key ring'.

9. From the Configuration and Administration forms, scroll down to the Security heading and select Receive Certificate. Use the Receive Certificate screen to receive the digital certificate from the CA into the web server's key ring. If you choose a self-signed certificate, you would specify the name and location of the CSR that you created in step 6 above. Enter the web server's key ring password, which you specified in step 6 of these instructions. Click Apply.

10. From the Configuration and Administration forms, under Security, select the Key Management form, then select the Manage Keys option. Click Apply.

11. On the Manage Keys screen, under Keys, click on the key name specified in step 6 of these instructions. Click on the Set as Default button. Click on Apply.

12. From the Configuration and Administration forms, under Security, select Security Configuration.

13. On the Security Configuration screen, make sure that the following are selected: *Allow HTTP connections*, and *Allow SSL connections using port 443*. We did **not** select *Enable SSL client authentication*, for this phase of our scenario testing.

14. Backup the key ring file (*inscokey.kyr*)**,** the stash file **(***inscokey.sth*) and the digital certificate file that you received from the CA or the CSR *(inscokey.txt*) if you chose to use a self-certified certificate.

15. If you disabled your browser's caching and proxy services in step 2, re-enable caching and proxy services.

16. Stop and start the web server to enable the security changes. A restart will not invoke the

security configuration changes.

17. There are two files that must be copied to each server in the cluster: (1) the main key ring file, *inscokey.kyr*, and (2) the "stash" file, *inscokey.sth*. (The stash file provides the password so that you don't have to manually enter it at server startup.)

18. The first time a client accesses the database, the browser presents the server certificate to the client, if the certificate was signed by a CA unknown to the browser. If the client trusts the certificate, the client should accept the server certificate for **www.insco.dmz.ibm.com**. If the client chooses the browser option to 'accept this certificate forever (until it expires)', the browser will store the certificate on the client's browser key ring. By accepting the certificate forever (until it expires), the client will not be asked to accept the server certificate each time the client starts a session to that server.

To create a key pair and secure server certificate for the Microsoft IIS web servers:

Click on the Windows Start button, select Programs, select Microsoft Internet Server (Common), select Internet Service Manager. From the Microsoft Internet Service Manager screen, select Help, select 'IIS Topics', which displays the *Microsoft IIS Installation and Administration Guide.* Select *Chapter 5 - Securing Your Site Against Intruders*. Go to the section entitled *Securing Data Transmissions with Secure Sockets Layer (SSL).* As the documentation states, enabling SSL security on a web server requires four steps. We followed the directions for those four steps to enable SSL on our IIS. Under each of the four steps in the Microsoft documentation, we learned additional information that was not documented in the section entitled *Securing Data Transmissions with Secure Sockets Layer (SSL).* We have documented the additional information to those four steps to enable SSL on an IIS below:

- Step 1. "Generate a key pair and a request file"

  In the 'Generating a Key Pair' section, we accepted the Key Manager default of a key pair that is 1024 bits long.

- Step 2. "Request a certificate from a certification authority"

  In the 'Acquiring a Certificate' section, we acquired our secure server certificate (a.k.a. a secure server ID) from VeriSign, Inc. We followed the steps outlined at http://digitalid.verisign.com/server/enrollIntro.htm.

- Step 3. "Install the certificate on your server"

  In the 'To install a certificate' section, follow steps 1 through 5. At this point additional information needs to be provided to the reader. That information is provided here:

  Step 5a. On the 'Key Manager' screen, the message 'You must now choose a server connection for this key to become fully activated on the target machine.' is displayed. Click OK.

  Step 5b. On the 'Server Connection' screen, click on 'Default IP' and click OK.

  Continue with steps 6 and 7 as documented.

- Step 4. “Activate SSL security on a WWW service folder”

No additional information was necessary for step 4.

Discoveries and Recommendations

- In the ‘To install a certificate’ section described above, when prompted by the ‘Server Connection‘ panel, clicking on ‘DefaultIP’ allows SSL connections to both the real IP address of the Windows NT server as well as the ‘Loopback’ address which is used by the IBM eNetwork Dispatcher. If you specify the IP address of the Windows NT server that the IIS resides on, SSL connections will only occur to the IIS at that IP address. If you specify the ‘Loopback’ address on the Windows NT server (a.k.a. the eND’s IP cluster address), then SSL connections will only occur through eND. Specifying ‘default IP address’ allows SSL connections to both IP addresses.
- For more in-depth information about SSL (digital certificates, Public Key/Private Key mechanisms, symmetric-key and Public-Key Cryptography, etc.) see the documentation on the following web site: http://home.netscape.com/newsref/ref/128bit.html
- We needed information from ‘Microsoft Online Support’ in order to enable our IIS for SSL. You will need a userid and password to access the information. Once you are logged in, you can access the URLs that we found helpful:
  - ❖ Require Secure SSL Channel Option is Not Available
    http://support.microsoft.com/support/kb/articles/q170/8/59.asp
  - ❖ IIS: Certificate Security Affected By Schannel.dll
    http://support.microsoft.com/support/kb/articles/q184/0/55.asp

- When establishing an HTTPS link in your HTML document, the server name that you enter in the link will become the name that is presented to the client browser as the certificate owner. For example, if you have the link:

  <a href=“https://www.insco.dmz.ibm.com/inscopolicy.html”> click here </a>

  then when the certificate is presented to the browser it will say it is from **www.insco.dmz.ibm.com**. However, if your link appears as:

  <a href=“https://9.67.142.69/inscopolicy.html”> click here </a>

  then the certificate is presented to the browser from the host 9.67.142.69

  For this reason, you should make sure that you use your site name in all HTTPS links.

Load Balancing with eNetwork Dispatcher

IBM's eNetwork Dispatcher (eND) is a software product that performs load-balancing and IP traffic management across multiple web servers over either the Internet or corporate intranet. It enables multiple web servers to efficiently function as a single system, greatly increasing system availability and responsiveness. eND addresses the scalability and availability challenges faced when bringing business applications to the Internet. The INSCO application needed the requests from the Internet to be balanced between the DMZ web servers. We used eND to distribute the requests initiated from the Internet client pool to the DMZ web servers.

Installing and Configuring eND on Windows NT

This section explains how to install eNetwork Dispatcher on Windows NT using the product CD. If you are downloading an evaluation copy of the product from the Web site, use the installation instructions on the Web site: http://www.software.ibm.com/enetwork/dispatcher/downloads/. Please note that the Windows NT version of eNetwork Dispatcher will not run on Windows 95 or any other version of Windows other than Windows NT, Workstation or Server, Version 4.0. If you have a version of eNetwork Dispatcher already installed on your sytem, please refer to the eNetwork Dispatcher Version 2.0 User's Guide for instructions on what to do before installing the latest version.

To install eNetwork Dispatcher:

1. Insert the eNetwork Dispatcher CD-ROM into your CD-ROM drive.

2. Using your mouse, click the left mouse button to perform these tasks:

   - Click on **Start**.
   - Select **Run**.
   - Specify the CD-ROM disk drive, followed by setup.exe, for example: *e:\setup*
   - Select **OK**.
   - Select **Component**.

3. From the Welcome window, click **Next**.

4. A license agreement will be displayed Read the agreement, and if you accept the provisions, click **YES**.

5. Select the type of installation.

   Note: The relevant README file or files are automatically installed.

   - **Typical**—Installs the English versions of Dispatcher and the Interactive Session Support

- **Custom—**Allows you to select and install the following functions in various languages:
  - Dispatcher
  - Interactive Session Support

6. Click Next.

   Note: The remainder of this procedure assumes a **Typical** installation.

7. From the Choose Target Directory window, select a target directory for the function you are installing:

   - The default target directory for Dispatcher is *C:\Program Files\eND\dispatcher*
   - The default target directory for ISS is *C:\Program Files\eND\ISS\*
   - The default target directory for this *User's Guide* is *C:\Program Files\eND\documentation*

8. If you want to change the drive or directory destination, click **Browse**.

9. Click **Next**.

10. After ISS has been installed, a message appears asking if you want to open the README file for that function. This README file contains up-to-date information regarding Interactive Session Support. You should familiarize yourself with it at this time.

    Note: The README files will explain how to start the eNetwork Dispatcher ISS functions in NT Services. You can run a function individually by selecting **NT Services** from the **Windows Control Panel**.

11. After installation is completed, a message will tell you to reboot your system before using eNetwork Dispatcher.

12. Click **OK**.

Once the system was rebooted, the following steps were performed to configure eND. A configuration script that contains these steps was written to instruct eND to forward the requests from the Internet to the DMZ web servers.

1. *Start the eND Service.* The eND Service is configured through the **Control Panel->Services** to be automatically started when the NT Server is rebooted.

2. *Start the eND executor.* This will load kernel extensions to the NT Server. From a command prompt window, issue the following command:

```
ndcontrol executor start
```

3. *Alias the eND's network adapter to the IP address of the eND cluster address.* The eND cluster address is the address we want the clients in the Internet to use to connect to the DMZ web servers. The eND cluster address for the INSCO application is 9.67.142.69 (www.insco.dmz.ibm.com). The eND's network adapter needs to have an alias defined to match the cluster address so that eND can receive packets destined for the cluster address and forward them on to the web servers. To define an alias, issue the ndconfig command:

   ```
   ndconfig en0 alias 9.67.142.69 netmask 255.255.255.240
   ```

4. *Define the non-forwarding address.* The non-forwarding address is the IP address of the eND machine itself. For the INSCO application, this is 9.67.142.77. The non-forwarding address is defined so eND won't forward the IP traffic destined for the eND machine to the web servers. To define the non-forwarding address, issue the following command:

   ```
   ndcontrol executor set nfa 9.67.142.77
   ```

5. A cluster address needs to be defined such that eND can accept traffic destined for 9.67.142.69 (www.insco.dmz.ibm.com). To define a cluster address, issue the following command:

   ```
   ndcontrol cluster add 9.67.142.69
   ```

6. Since the INSCO application needs to handle HTTP and HTTPS traffic, two ports were added to distinguish between the two types of traffic. Port 80 is used to handle the HTTP traffic, and port 443 is used to handle the HTTPS traffic. To add ports, issue the following command:

   ```
   ndcontrol port add 9.67.142.69:80
   ndcontrol port add 9.67.142.69:443
   ```

7. Since one of the ports is defined to handle HTTPS traffic, the sticky time setting needs to be set for this port. The sticky time setting allows eND to remember what server it assigned to a request. When the secure session is established, eND will forward a request from the client to the web server for the established connection. The client connects to a web server (through eND) that establishes a session key that will be used by the server and the client in the encrypted communication. The next time the client needs to communicate with the server it will use this session key and it needs to connect to the same server that issued the session key. For example, the client establishes a session to submit a change to data. The client will then have to enter his/her name and password. This information is passed back to the web server for authentication. If the authentication is successful the client would see the information and continue to submit changes. The sticky port ensures that establishing the session key, authenticating, and the remainder of the transactions are continued with the same server. The sticky time setting of the eND

port allows eND to remember what server eND assigned the client for the amount of time specified in seconds. To define a sticky time for a port, issue the following command:

```
ndcontrol port set 9.67.142.69:443 stickytime 180
```

8. A connection is considered stale when there has been no activity on that connection for the number of seconds specified by the staletimeout setting for that port. Once the stale timeout has elapsed, the connection is removed. To define a stale timeout for a port issue the following command:

```
ndcontrol port set 9.67.142.69:443 staletimeout 300
```

9. *Define within eND the web servers and addresses that are in the Cluster*. For the INSCO application, there are three web servers that are used to handle requests. To define web server addresses for the Cluster, issue the following commands:

```
ndcontrol server add 9.67.142.69:80:9.67.142.65
ndcontrol server add 9.67.142.69:80:9.67.142.67
ndcontrol server add 9.67.142.69:80:9.67.142.74

ndcontrol server add 9.67.142.69:443:9.67.142.65
ndcontrol server add 9.67.142.69:443:9.67.142.67
ndcontrol server add 9.67.142.69:443:9.67.142.74
```

10. *Start the manager component of eND.* The manager component is an optional component of eND and is not needed for eND to function. However, the manager component is an excellent way to optimize the load-balancing provided by eND. The manager uses internal counters in the executor, feedback from the servers provided by the advisors, and feedback from a system-monitoring program (such as Interactive Session Support) to decide which web server should be sent incoming requests. The manager component of eND is started by issuing the following command:

```
ndcontrol manager start
```

11. *Start the advisor component of eND.* The advisor component is used in conjunction with the eND manager component. The advisor component will provide information to the manager component about the web servers that are in the cluster. This information is used by the manager component to decide which web server should be sent the incoming request. The advisor gathers information on a per port, per server basis. The advisor is started by issuing the following commands:

```
ndcontrol advisor start HTTP 80
ndcontrol advisor start SSL 443
```

12. *Assign proportions to eND.* Assigning proportions instructs eND how to weigh decisions in determining to which web server the incoming request should be sent. There are four fields that can be weighed for the decision making. The first is the number of active connections on each web server. The second is the number of new connections on each web server. The third is the input from the advisors, and the fourth is the input from the system monitoring tools such as Interactive Session Support (ISS). The manager and advisor were sufficient for the INSCO application, so we did not use ISS. The proportions used for the INSCO application are 45-45-10-0. To configure the manager component to use proportions, issue the following command:

```
ndcontrol manager proportions 45 45 10 0
```

Configuring the Web Servers

The next set of steps are to be performed on the web servers that will be in the eND Cluster. These steps are used to alias the loopback adapter on the web server machine to that of the eND Cluster Address. The steps are different depending on the operating system.

On the AIX Server, issue the following command:

```
ifconfig lo0 alias 9.67.142.69 netmask 255.255.255.240
```

On the Windows NT server, the MS Loopback Adapter must be installed. To do this, perform the following:

1. Click **Start**, then click **Settings**.

2. Click **Control Panel**, then double click **Network**.

3. If you have not done so already, add the MS Loopback Adapter Driver.

   - In the Network window, click **Adapters**.
   - Select MS Loopback Adapter, then click **OK**.
   - When prompted, insert your installation CD or disks.
   - In the Network window, click **Protocols**.
   - Select TCP/IP Protocol, then click **Properties**.
   - Select MS Loopback Adapter, then click **OK**.

4. Now in this window set the loopback address to the cluster address and define the netmask. The loopback adapter is set to 9.67.142.69 and the netmask is defined as 255.255.255.240. Note: You may have to exit and reenter Network Settings before the MS Loopback Driver shows up under TCP/IP Configuration.

5. An extra route is added into the routing table that needs to be deleted. After installing and configuring the MS Loopback Adapter, you will be asked to reboot the server. After rebooting, you need to delete a route from the routing table. For more information on this see the eND User's Guide. The following command was used to remove this route:

```
route delete 9.67.142.64 9.67.142.69
```

Discoveries and Recommendations

Sticky Time & Stale Timeout
In our implementation, we began by testing that eND would dispatch requests in a round-robin fashion without using the manager and the advisor components. However, we didn't take into account the interactions of the sticky time and stale timeout of the ports.

The sticky time is used to set the affinity of a client to a particular web server. This is especially important for applications which uses SSL and HTTPS. A client uses SSL to connect to a web server for secure communication. Once the session key is agreed upon between the client and web server, the client should continue to go to the same web server for the rest of the encrypted session. When the sticky time, in seconds, is set for a port and a client connects to that port, eND dispatches the request to a server based on whatever selection criteria it is using. The client will continue to go to the same web server until the connection is terminated AND the sticky time has elapsed.

A connection is considered stale when there has been no activity on that connection for the number of seconds specified by the staletimeout setting for that port. Once the stale timeout has elapsed, the connection is removed.

Note that the sticky time doesn't take effect until the stale timeout has elapsed for open connections. So, for example, a client connects to a web server and for some reason, the connections remains open meaning it's not in the FINned state. Then, the sticky time will not take effect until the stale timeout has elapsed and the connection is closed. Once the connection is closed, then the sticky time will take effect Once the sticky time has elapsed, the client will no longer have an affinity for that web server and the next time the client wants to connect to the web address, eND will dispatch the request to the next available web server based on the load balancing criteria.

Reconfiguration after reboot
When the eND machine is rebooted, the configuration steps defined above are lost and the commands have to be re-entered on the eND machine. You can, however, create a configuration command file to initiate steps 1 to 11, described above in "Configuring eND." A sample configuration command file is shipped with eND and is located in the *\eND\ND\BIN* directory. A sample of the command file for our environment is located in "Appendix F eND Configuration File," in this report.

Advisor on SSL Port

Handshake failed error messages will show up in the web server's logs because of the way the advisor probes the web servers for information. You will see these messages for ports defined to the advisor as SSL ports. The advisor will probe each web server according to the interval, in seconds, defined by the update interval parameter and you will see an error message in each web server's log corresponding to each probe. This is a known problem and can be ignored. If you find these error messages confusing, hindering debug efforts, or filling up the web server's log, it is recommended that you stop the advisor for the SSL ports.

Resetting the Loopback Adapter on the Web Server

If a web server is rebooted, the Loopback Adapter aliasing needs to be redone:

- For AIX, add the ifconfig statement in the */etc/rc.net* file.
- For Windows NT, delete the route using the route delete command shown above in Step 5.

Logging

For performance reasons, eND does not keep a log of each request it receives and forwards. The logging is left to each individual web server. Because of this, if you wish to keep a log of all activity at your web site you will need to combine the logs of each individual web server

Transarc Andrew File System (AFS) - File Sharing System

AFS was initially developed at Carnegie Mellon University (CMU) in the early 1980s to solve the problem researchers had of efficiently sharing information. Network File System (NFS) was initially studied and CMU found many inefficiencies with the architecture. CMU needed a file system which could alleviate the problems of duplicate data, file server outages, decentralized & fragmented storage located at different physical sites, having consistent access to the same data by different clients, and minimizing system administration & effort. The result was the development of AFS.

AFS offers a variety of features which make it easy to use, scaleable, and preferable to other existing file systems. AFS provides a simple filenamespace architecture. The actual files and directories are related to physical data on servers, however, the names are visible in a single, abstract, enterprise-wide, and consistent directory structure. This provides a seamless file namespace visible by all desktops and all users.

AFS clients cache data locally to reduce network traffic and AFS servers provide tracking and invalidation processing of data between clients and servers. In AFS, files are stored in logical containers called volumes. Volumes can exist on one or more servers within the enterprise. These volumes can be replicated for reliability, higher availability, load balancing, and failure recovery.

To facilitate the management of the file system, AFS uses a centralized database on one or more database servers. The database servers, file servers, and clients make up what is called an AFS cell. AFS uses Kerberos along with access controls for security management of the AFS cell.

One of the problems imposed by the INSCO scenario was how to keep all the data consistent between all the web servers in the DMZ while making it easy to update this data and make this data available to all the web servers at the same time. The answer was the Transarc Andrew File System (AFS) Version 3.4a which is packaged in the IBM WebSphere Performance Pack Version 1.0.

We began by asking the question, "How do we keep data between all the web servers consistent while making it easy to update the data?" In addition, we considered having the exact directory structure so that web servers could use the same configuration files. We started with three web servers in the DMZ, but needed a scaleable solution with the ability to add more web servers as needed. Because we used eND, the data needed to be consistent across the web servers. The solution needed to be secure; in case the DMZ was compromised, there would still be levels of security. AFS was the perfect solution for our environment.

Our design was straightforward. One AFS server and two AFS read-only replica file servers to make up the AFS cell, which we named "pie". If the AFS server crashed or was not available on the network, the other two file servers could still make the data available to the web servers in the DMZ. The three web servers in the DMZ (2 NT & 1 AIX) run AFS client software and connect to the "pie" cell. The following section describes the installation and configuration of these AFS machines.

Installing the AFS Server on AIX

1. Installing the First AFS Server

Install the first AFS server using WSPP. This creates the AFS cell containing only one server and also makes this server an AFS client in the AFS cell.

- Insert CD into CD-ROM drive
- mkdir /cdrom
- mount -rv cdrfs /dev/cd0 /cdrom
- cd /cdrom/AIX
- java setup
- The Java installshield welcome window appears. Click "Next>"
- The Software License Agreement appears. Read & click on the Accept all terms of the license box. Click "Next>"
- Read the README. Click "Next>"
- Ensure you have met the Disk Space, Memory, and Operating System requirements.
- Select Language. Click "Next>"
- Enter destination for WebSphere, /public/WebSphere is the default; Click "Next>"
- When asked, "Would you like the location created?", answer "Yes"
- Select "File Sharing Server"; Click "Next>"
- Do you want to replace the current version of any product that has already been installed? Yes, click "Next>"
- Click "Install"; When done, click "Finish"
- Follow the instructions in the *IBM WebSphere Performance Pack for Multiplatforms, Getting Started***,** Chapter 3. "Configuring Performance Pack Components" document.

2. Configuring the AFS Server on AIX

Use the instructions found in "Configuring the AFS Server on AIX" section to configure the server portion of AFS on the first server machine. Please note that the directory where *afsConfigureServer.ksh* is found should be *WebSphere/AFS* and **not** *WebSphere/afs*. Also, make sure to use the full path name for commands issued in this section, for example, */usr/afs/bin/bos*, */usr/afs/bin/kas*, etc. If you have DCE installed on the machine, there will be DCE commands in */usr/bin* with the same name as the AFS commands in this section. Using full path names will ensure you are issuing the correct commands for AFS.

3. Configuring the AFS Client on AIX

Although this section is titled "Configuring the AFS Client on AIX", it is important to note that it works only if the server portion of AFS is already installed on the machine. This section is actually used to install the client portion of AFS on the first server machine. Note the following changes to the instructions.

1. In step 1 of the *IBM WebSphere Performance Pack for Multiplatforms, Getting Started***,** Chapter 3, section entitled "Configuring the AFS Client on AIX", you are asked to copy *ThisCell* and *CellServDB* from the AFS server directory (*/usr/afs/etc*) to the AFS client directory (*/usr/vice/etc*). These files already exists in the client directory as symbolic links to the files in the server directory. You must remove the symbolic links before

copying the files. Issue the following commands to remove the symbolic links and copy the files to the client directory.

```
# cd /usr/vice/etc
# rm ThisCell
# rm CellServDB
# cp /usr/afs/etc/ThisCell /usr/vice/etc/ThisCell
# cp /usr/afs/etc/CellServDB /usr/vice/etc/CellServDB
```

2. In step 11 of the *IBM WebSphere Performance Pack for Multiplatforms, Getting Started*, Chapter 3, section entitled "Configuring the AFS Client on AIX", the Java command in the last sentence should include the CELL_NAME as follows:

   ```
   java afsConfigureClient CELL_NAME MACHINE_NAME PARTITION_NAME
   ```

4. Installing & Configuring Additional File Servers
To install additional file server machines, follow the steps outlined in the *AFS Installation Guide Version 3.4* by Transarc, "Chapter 3. Installing Additional Servers". Read all the steps very carefully and follow them very closely. Note the following change to the instructions.

In *step 1*, section 3.1.1 "Loading Files to the Local Disk", add the following command just before creating the */usr/vice/etc* directory:

```
# mkdir /usr/vice
```

Installing the AFS Client

1. Installing the AFS Client on AIX
To install an AFS client on AIX, you must first install the first AFS server machine. Then, follow the steps outlined in the *AFS Installation Guide Version 3.4* by Transarc, "Chapter 4. Installing Additional Client Machines". Read all the steps very carefully and follow them very closely. Note the following change to the instructions.

In *step 3*, section 4.2.1 "Using the Kernel Extension Facility on AIX Systems", the last instruction should read:

afsd (you will be instructed to add this command in Section 4.4)

2. Installing the AFS Client on Windows NT
The steps to install the AFS Client on a Windows NT Server are:

1. Verify that you have Java Development Kit(JDK) 1.1.6 installed and check the associated environment variables: click the Windows Start button, click Settings, click Control Panel, click System, click Environment. Under the System Variables, the

‘classpath’ variable should contain “*<drive>:\jdk1.1.6\lib\classes.zip;.* ” and the ‘path’ variable should contain “*<drive>:\jdk1.1.6\bin;* ” (without the “ ” and <drive> is the harddrive location of the file).

2. Insert the WebSphere Performance Pack CD Version 1.0, change to the NT directory and double click on setup.exe .
3. On the ‘InstallShield Java Edition’ screen, a list of the Java Virtual Machines found is displayed. We chose JDK1.1.6 since our scenario includes WebSphere Application Server, which requires JDK1.1.6. Click OK.
4. On the ‘Welcome’ screen, click Next.
5. On the ‘Software License Agreement’ screen, check the box to indicate that you ‘Accept all terms of the license’.
6. On the ‘README Information’ screen, read the Readme file and click Next.
7. On the ‘InstallShield Java Edition’ screen, select English and click Next.
8. On the ‘Choose Destination Location’ screen, you can accept the default or specify another location. Click Next.
9. On the ‘InstallShield Java Edition’ screen, to install the AFS client, click ‘File Sharing Client’ and click Next.
10. On the ‘InstallShield Java Edition’ screen, when asked “Do you want to replace the current version of any product that has already been installed?’ Click Yes. Click Install.
11. On the ‘Setup Complete’ screen, when prompted to read the AFS readme file, select ‘Yes I want to review the Readme file’. When prompted to configure the AFS select ‘Yes I want to Configure AFS client now’.
12. On the ‘AFS Client Configuration’ screen, enter the AFS cell name. The other values on the screen are okay.
13. On the ‘AFS Client Configuration’ screen, click the ‘Cell Hosts’ tab, click add. On the ‘Add Cell’ screen enter the cell name. For each of your AFS servers, enter the TCP/IP name and the TCP/IP address and click add. When you have added all of your AFS servers, click OK.
14. On the ‘Cell Hosts’ screen, delete the AFS cells that you will not be using. Click OK.
15. Click the Windows Start button, click Shut Down, click ‘restart the computer?’, click Yes.
16. After the computer restarts, click the Windows Start button, click Settings, click Control Panel, select Services, select Transarc AFS Daemon, click Stop to stop the service. When you are asked ‘Are you sure you want to stop the Transarc AFS Daemon Service?’ click Yes. Click Startup button, under Startup Type select Manual, click OK, click Close.
17. As directed by the WSPP readme.txt file, you must get the latest AFS 3.4a Client for Windows patch file from the Transarc web site:

http://www.transarc.com/dfs/public/www/htdocs/.hosts/external/Support/afs/index.html
We tested with patch 8 (afs34p8.exe).

18. Execute the AFS 3.4a Client patch file.
19. On the 'Information' screen, both 'Backup Enabled' and 'Installation Logging Enabled' should be checked. Click Next.
20. On the 'Question' screen click Yes to read the Patch Readme file now.
21. On the 'Setup Complete' screen, select 'No, I will restart my computer later.' Click Finish.
22. Click Windows Start button, click Settings, click Control Panel, click Services, select Transarc AFS Daemon, click Startup button, under Startup Type select Automatic, click OK, and click Close.
23. Click the Windows Start button, click Shut Down, click 'restart the computer?', click Yes..
24. On your Windows NT Server desktop, right click on the Network Neighborhood icon. Select Map Network Drive, select an unused drive, in the path box type \\nnnnnnnn-afs\all (where *nnnnnnnn* is the Windows NT server's name.). Check 'Reconnect at Logon' and click OK.
25. Your AFS client on Windows NT server is ready to use.

Discoveries and Recommendations

References
WSPP Java installation works very well to get started. It installs the first server in the AFS cell as both an AFS server & client. If you want to install other clients or additional file servers in the AFS cell, then you will need to use the *AFS Installation Guide Version 3.4*, follow the steps, and use the corrections indicated above.

AFS can become quite complex especially for very specialized requirements. We found the following to be very good references:

Managing the Andrew File System, by Richard Campbell, published by Prentice Hall.

AFS Installation Guide Version 3.4, published by Transarc.

Refer to the following web sites for updates, fixes, and help:
- The WebSphere Performance Pack home page at
  `http://www.software.ibm.com/webservers/web/`
- The Andrew File System home page at
  `http://www.transarc.com/`

Resource Control File - */etc/rc.afs* (for AIX only)

The following is the contents of */etc/rc.afs* to be used on AIX clients and servers. Once the appropriate entry is added to */etc/inittab*, *rc.afs* will start AFS automatically upon system reboot.

```
#
# Starts the AFS server and client processes
#
echo 'Starting AFS...' > /dev/console

echo '   Adding AFS Kernel Extensions...'>/dev/console
/usr/vice/etc/dkload/cfgexport -a /usr/vice/etc/dkload/export.ext
/usr/vice/etc/dkload/cfgafs    -a /usr/vice/etc/dkload/afs.ext

if [ -f /usr/afs/bin/bosserver ]
then
        echo '   Starting AFS bosserver...' > /dev/console
        /usr/afs/bin/bosserver &
fi

echo '   Starting the AFS daemon...'>/dev/console
/usr/vice/etc/dkload/rc.afsd.large
```

Entry to Add to */etc/inittab* (for AIX only)

Add the following line to the */etc/inittab* file just before the command that starts the NFS daemons but just after the line that starts the TCP/IP daemons:

```
rcafs:2:wait:/etc/rc.afs > /dev/console 2>&1 # Start AFS Daemons
```

Setting up the Firewalls

The firewall is a functional unit that protects and controls the connection of one network to other networks. Firewalls prevent unwanted or unauthorized communication traffic from entering the protected network and allow only selected communication traffic to leave the protected network. One of the most commonly used components of firewall architecture is IP packet filtering, in which all packets passing through a firewall are screened to either prevent or allow access to hosts and to ports on those hosts. IP packet filtering screens packets for the following information:

- Source IP address
- Destination IP address
- Protocol (TCP, UDP, ICMP)
- TCP or UDP source port
- TCP or UDP destination port

In addition, the IP packet filtering component knows on which firewall interface the packet arrives, and on which firewall interface the packet exits the firewall. Based on this packet information a list of packet filter rules (configured on the firewall) is scanned to determine if the packet will be forwarded or not. IP packet filtering is the basis for other firewall architectures, such as the *screened subnet architecture,* which we chose to implement.

The *screened subnet architecture* usually has two firewalls. The first firewall sits between the Internet and the DMZ. Clients on the Internet use web browsers to access the web servers. The web servers (or second tier servers) are located in the DMZ. The second firewall sits between the DMZ and the intranet. The Enterprise Servers are located in the intranet. The most sensitive servers are located in the intranet. To break into the intranet, an intruder would have to get by both firewalls. See *Figure 2*: Screened Subnet Architecture, shown earlier in this report, for an example of a screened subnet architecture implemented with two firewalls.

The *screened subnet architecture* may also be implemented using one firewall. The decision to use only one firewall may be based on availability of a second firewall or on the need to save money by purchasing and installing only one firewall. We chose to implement the *screened subnet architecture* with only one firewall. Our firewall has a non-secure interface to the Internet, a secure interface to the DMZ, and a secure interface to the intranet. See *Figure 5*: *The Production Integration Environment,* shown earlier in this report.

We installed and configured the IBM eNetwork Firewall Version 3.2 for AIX. The following sections explain how this firewall was configured.

Implementation of IBM eNetwork Firewall for AIX

We used the *IBM eNetwork Firewall for AIX User's Guide Version 3.2*, Chapter 2, "Planning" and the "Network Configuration Planning Worksheet" to plan our network configuration before installing and configuring our firewall. Once we completed planning for our firewall, we were ready to install the IBM eNetwork Firewall for AIX. To install the firewall, we referred to the *Setup and Installation of the IBM eNetwork Firewall* instruction booklet that is included with the *IBM eNetwork Firewall Version 3.2* product package. Then we read the *IBM eNetwork Firewall for AIX User's Guide Version 3.2,* Chapter 3, "Setting Up the Configuration Server and the Configuration Client," and Chapter 4, "Using the Configuration Client" to set up and logon to the configuration client.

Configuring the firewall consists of seven main tasks:

1. Designate the network interfaces
2. Set up your general security policy
3. Create Network Objects
4. Configure the domain name system (DNS)
5. Create additional firewall services as needed
6. Build connections on the firewall
7. Enable connections on the firewall

The following sections describe in detail these seven tasks. For each task we referred to specific chapters and sections of the *IBM eNetwork Firewall for AIX User's Guide Version 3.2*. We will identify these chapters in the descriptions of each task.

Designate the Network Interfaces

In Chapter 5, "Getting Started on the IBM Firewall," the section, "Designating your Network Interface" explains how to specify which of your network interfaces are secure and which are not secure. We had one non-secure interface and two secure interfaces. The non-secure interface connected to the Internet. One of the secure interfaces connected to the DMZ and the other secure interface connected to the intranet. The steps are:

- Select System Administration from the configuration client navigation tree
- Select Interfaces, which displays the addresses of the firewall's network interfaces
- Select an interface and click Change
- Specify if the interface is secure or non-secure and click Close to complete the change
- Repeat for the other interfaces

Set Up the General Security Policy

In Chapter 5, the section "Using the Configuration Client to Define a Security Policy" explains how to define the firewall's general security policy. The Security Policy provides a quick and easy way for administrators to set blanket policies for the firewall. The steps are:

- Select System Administration from the configuration client navigation tree
- Select Security Policy which displays the Security Policy window
- Select the following options and then click OK
    1. Permit DNS queries, which allow Domain Name Service resolution requests and replies.
    2. Deny broadcast messages to the non-secure interface, which prevents broadcast messages from being received at the non-secure adapter.
    3. Deny socks to non-secure adapters, in order to disallow socks traffic to enter the firewall from the non-secure network.

We did not select either 'Enable Telnet' or 'Enable FTP'. We did not need to use Telnet or FTP for our scenario and to minimize the security exposure we did not allow Telnet and FTP through the firewall. In order for these options to take effect, the changes must be activated through the Connection Activation panel. You can wait until step 7 of these configuration steps, "Enable Connections on the Firewall" to activate the changes.

Create Network Objects

In Chapter 5, the sections, "Network Objects" and "Using the Configuration Client to Define Network Objects" explain how single network objects are created to represent single objects (a host, a firewall, a router, etc.) in your network, and group network objects are created to represent a group of objects, such as a cluster of web servers.

We created a single network object of type host for:

- the IBM eNetwork Dispatcher's (eND) cluster address in the DMZ (see the "Load Balancing with eNetwork Dispatcher" section of this report for more information)
- each web server in the DMZ
- each AFS Server in the intranet
- DB2 Server in the intranet

We created a group object for:

- all the web servers in the DMZ.
- all the AFS servers in the intranet

We used the predefined network object called "The World" to represent all the clients in the Internet portion of our test environment.

Note: These network objects will be used as the Source object and/or the Destination object in the "Build Connections on the Firewall" section below.

Configure the Domain Name System (DNS)

In Chapter 5, the section "Configuring DNS Using the Configuration Client" explains how to configure DNS on the firewall. To configure DNS on the firewall, the steps were:

1. Select System Administration from the configuration client navigation tree
2. Select Domain Name Service
3. Enter the information. We entered the information as follows:

    - Secure Domain Name - raleigh.ibm.com
    - Secure Domain Name Server - 9.67.148.246
    - Non-Secure Domain Name Server - 9.67.148.246 9.67.142.70 9.67.142.60

9.67.142.70 is the IP address of the DNS in the DMZ. 9.67.142.60 is the IP address of the DNS in the Internet.

Create Additional Firewall Services as Needed

Firewall services permit or deny various protocols to flow through the firewall. The firewall has a set of predefined services. The firewall user may also create services for their specific environment and needs. The firewall has predefined HTTP and HTTPS services from the intranet to the Internet. Our scenario required HTTP and HTTPS services from the Internet to the DMZ, so we defined new services to permit that traffic through the firewall. To add these and other services and their rules, we used the configuration client navigation tree.

- Select Traffic Control
- Select Connection Templates
- Select Rules to add the new rules.
- Under Connection Templates, select Services to create a service that will contain these new rules. The new services and the rules in these services were defined as shown in the following tables. For each firewall rule, the rule template is:

action	protocol	operation at source	port # at source	operation at destination	port # at destination	interface	routing	direction

For additional information about the rule template fields, see the *IBM eNetwork Firewall for AIX User's Guide Version 3.2,* Chapter 8: "Controlling Traffic Through the Firewall." We defined the following services and their rules, which were needed by the INSCO application.

Note: We named the services and the rules in these services to correspond with the naming conventions used for the firewall's predefined services and rules. These services and their rules generate a list of IP filter rules that determine whether or not each IP packet that arrives at the firewall will be permitted or denied access to and through the firewall. For example, the new service "HTTP Direct In" contains four rules. The first rule "HTTP 1/2 Inbound from Non-secure" will generate the IP filter rule to permit an HTTP packet from the Internet, inbound to the firewall, over the non-secure adapter, destined for the DMZ. The second rule "HTTP 2/2 Outbound to Secure" will generate the IP filter rule to permit an HTTP packet from the Internet, outbound from the firewall, over the firewall adapter connected to the DMZ. The third rule "HTTP Ack 1/2 Inbound from Secure" will generate the IP filter rule to permit an HTTP Ack from the DMZ, over the firewall adapter connected to the DMZ inbound to the firewall. The fourth rule "HTTP Ack 2/2 Outbound to Non-secure" will generate an IP filter rule to permit an HTTP Ack outbound from the firewall, over the firewall adapter connected to the Internet, to the Internet.

<u>HTTP</u>

Service name: HTTP Direct In

Rule name: HTTP 1/2 Inbound from Non-secure

permit	tcp	gt	1,023	eq	80	non-secure	route	inbound

Rule name: HTTP 2/2 Outbound to Secure

permit	tcp	gt	1,023	eq	80	secure adapter to DMZ	route	out bound

Rule name: HTTP Ack 1/2 Inbound from Secure

permit	tcp/ack	eq	80	gt	1,023	secure adapter from DMZ	route	inbound

Rule name: HTTP Ack 2/2 Outbound to Non-secure

permit	tcp/ack	eq	80	gt	1,023	non secure	route	out bound

Note: The most secure approach is to specify "secure adapter to/from DMZ" for the interface field to ensure that the HTTP traffic will only flow over the secure adapter connecting to the DMZ and NOT flow over the secure adapter connecting to the intranet.

HTTPS

Service name: HTTPS Direct In

Rule name: HTTPS 1/2 Inbound from Non-secure

permit	tcp	gt	1,023	eq	443	non-secure	route	inbound

Rule name: HTTPS 2/2 Outbound to Secure

permit	tcp	gt	1,023	eq	443	secure adapter to DMZ	route	out bound

Rule name: HTTPS Ack 1/2 Inbound from Secure

permit	tcp/ack	eq	443	gt	1,023	secure adapter from DMZ	route	inbound

Rule name: HTTPS Ack 2/2 Outbound to Non-secure

permit	tcp/ack	eq	443	gt	1,023	non secure	route	out bound

Note: The most secure approach is to specify "secure adapter to/from DMZ" for the interface field to ensure that the HTTPS traffic will only flow over the secure adapter connecting to the DMZ and NOT flow over the secure adapter connecting to the intranet.

DB2

Service name: DB2 Client/Server Access

Rule name: DB2 to Port 50000 from the DMZ to the intranet

permit	tcp	gt	1,023	eq	50,000	secure	route	both

Rule name: DB2 to Port 50001 from the DMZ to the intranet

permit	tcp	gt	1,023	eq	50,001	secure	route	both

Rule name: DB2/Ack from Port 50000 from the intranet to the DMZ

permit	tcp/ack	eq	50,000	gt	1,023	secure	route	both

Rule name: DB2/Ack from Port 50001 from the intranet to the DMZ

permit	tcp/ack	eq	50,001	gt	1,023	secure	route	both

AFS

Service name: AFS Client/Server Access

Rule name: AFS Client Port 7001 from the DMZ to AFS Server Port 7000 in the intranet

permit	udp	eq	7,001	eq	7,000	secure	route	both

Rule name: AFS Client Port 7001 from the DMZ to AFS Server Port 7003 in the intranet

permit	udp	eq	7,001	eq	7,003	secure	route	both

Rule name: AFS Server Port 7000 in the intranet to AFS Client Port 7001 in the DMZ

permit	udp	eq	7,000	eq	7,001	secure	route	both

Rule name: AFS Server Port 7003 in the intranet to AFS Client Port 7001 in the DMZ

permit	udp	eq	7,003	eq	7,001	secure	route	both

Note: AFS uses the ports shown in the rules above as follows:

7000 - fileserver

7001 - cache manager callback service

7003 - vlserver (vldb) - the client accesses this port to look for the location of new information

We did not include port 7004 in our rules since our scenario did not require the use of Kerberos authentication for security. For more information about AFS ports access: http://www.angelfire.com/hi/plutonic/afs-faq.html#sub3.17

Debugging Service

Service name: Deny All Traffic

Include the firewall's predefined rule called 'All-deny any' in this service.

Build Connections on the Firewall

Once you have defined any additional services and rules that you will need for your firewall configuration, you must define the connections that will use the firewall's predefined services as well as the additional services that you defined. Refer to the *IBM eNetwork Firewall for AIX User's Guide Version 3.2*, Chapter 8, "Controlling Traffic Through the Firewall," for step-by-step instructions on building connections using predefined service and the services that you defined. We built the following connections and services for the firewall in this scenario. The Source objects and Destination objects in the table below were created in the "Create Network Objects**"** section above**.**

Source object	**Destination Object**	**Services**
The World	Single object for the IBM eNetwork Dispatcher in the DMZ	HTTP direct in HTTPS direct in
Group object for the web servers in the DMZ	Single object for the DB2 server in the intranet	DB2 Client/Server Access
Group object for the web servers in the DMZ	Group object for the AFS Server in the intranet	AFS Client/Server Access

Enable Connections on the Firewall
Once the connections are defined,

- Select Connection Activation from the configuration client navigation tree.
- Click on Regenerate Connection Rules and Activate, then click the Execute button
- Once that command has completed, we clicked on List Current Connection Rules to see the list of IP filter rules generated for the firewall. Each time an IP packet comes into the firewall, the packet is compared against this list of rules until a match is found. If no match is found the firewall default is to deny that IP packet.
- We also clicked on Enable Connection Rules Logging, and clicked on the Execute button. The firewall will log selected traffic to the *local4* log. See the *IBM eNetwork Firewall for AIX User's Guide Version 3.2*, Chapter 19, "Managing Log and Archive Files" for more information about the firewall's logging facilities.

Discoveries and Recommendations

Network Objects. When defining a single network object for a single server, be sure to specify a subnet mask of 255.255.255.255. The network object will be part of a network connection with firewall services specified by the firewall administrator. When the connections are activated, the list of IP filter rules will be generated on the firewall. A single network object with an IP address and a subnet mask of 255.255.255.255 ensures that only the host with that exact IP address will be allowed the access specified in the IP filter rules, in which that network object was included.

NetBIOS Configuration information.
The AFS client uses NetBIOS over TCP/IP to access data on the AFS server. Windows NT NetBIOS requires a specific value for the Lana number of the LAN adapter carrying the NetBIOS traffic over TCP/IP. To ensure the correct configuration: at the Windows NT server, click the Windows Start button, click Control Panel, click Network, click Services, click NetBIOS interface. On the NetBIOS Configuration panel, the Lana number must be 000 for the adapter which will carry the NetBIOS traffic. If the Lana number is not 000, NetBIOS will not work.

A VERY GOOD Debugging Aid. If you are having a problem with traffic not going through your firewall as expected, add the following connection at the bottom of the list of connections.

Source object	**Destination Object**	**Services**
The World	The World	Deny all traffic

Once this connection is activated, if an IP packet does not match any of the previous IP filter rules (activated in the “Enable Connections on the Firewall” section), it will match this one and also be logged. By studying the log, you can determine if there is traffic that should be going through the firewall and is being denied. If it is appropriate, you can add connections with services that will permit that traffic.

Caching on the Server and the Browser

When designing any Internet application you must take into consideration what data will be cached, and where it will be cached. In even the simplest Internet application caching can occur at both the client's browser and at your web server. In addition, there may be intermediate proxy servers that could also be caching your data. If you are using a file system such as AFS, as we did, you will introduce another level of caching. Of these, the only ones you have complete control over are those in your environment. For our application these were the web server and AFS. Fortunately, there are things you can do to avoid problems where the caching is done by processes you cannot control.

In general, caching is done to improve performance. For example, there is no need to take the time to send an HTML page across the Internet if you have a perfectly good copy of it already saved in your browser. For similar reasons, many sites employ caching web servers, whose sole function is to keep "commonly used pages" and delver them on request. This allows the load on the regular web servers to be reduced, and can greatly improve the performance of a web site. However, when you are building dynamic HTML, as we do in our application, these pages may be out of date, and serving older, cached pages defeats the purpose of building your HTML dynamically. For sites using a distributed file system, such as AFS, having an AFS client cache pages from the server reduces the time it will take for the web server to find and serve a page.

The following is a discussion of each type of caching and what you can do to control and benefit from them when enabling your application.

To avoid the problem of intermediate proxy servers caching your HTML you just need to ensure that the following three fields are set as shown in the header of the HTML page you return. If you are generating your HTML from a servlet you can use the res.setHeader method to do this.

FIELD	SETTING
Pragma	No-cache
Cache-Control	No-cache
Expires	0

These settings will tell any server that may be forwarding your page that it is not to cache it. The reason for multiple no-cache fields is that not all servers recognize all header elements.

In our application, all but our first two pages were built dynamically by our servlet. Since the request for the page is made via a POST request to the servlet, as opposed to a GET request for an HTML page, and the page is built by the servlet, current server design will always recreate these pages. However, putting the no-cache element into the page does no harm, and will in fact improve the overall performance, as it tells any caching server not to waste time and cache space with this page.

Caching at the browser is another thing that must be considered. As an application developer you have no control over the client's browser settings. It is possible for the client user to eliminate all browser caching when using Netscape, but Microsoft's Internet Explorer does not allow this. For these reasons you must develop your application to handle any browser caching. As a general rule, browsers treat meta-data for caching in the same manner as proxy servers. This means that the lines we added to our HTML headers to prevent proxy servers from caching our pages will also keep browsers from doing so. Someone thing we learned, and documented in the Migrating Websphere Application Server to 1.1 section, is that for JSPs the META tags listed above may not always work with all browsers. The solution to this was to replace the META tags with inline Java code that will accomplish the same thing. The Java code is listed in the Migration section of this report and can also be found in the sample JSP in Appendix D.

Caching is also an important part of AFS. To reduce network traffic and server load AFS clients cache files retrieved from the file server. When a file is accessed it is automatically cached at the client. The next time the client needs to access the file it will be able to use the cached version, which will increaseperformance. When a new version of the file is installed on the file server the file server will call back the clients that cached the file and give them notification that the file has been updated. On the next access of the file the client will realize that the local copy is out of date and retrieve the newer version from the file server. The size of the AFS cache is set by each AFS client when it is installed and can be changed at any time. On Windows NT virtual memory is used for the AFS cache, while on AIX you have the choice of designating that the cache be either in memory or on hard disk. In either case you should balance the size of your AFS cache with other applications needs for these resources. In our application we chose a cache size of 20 MB for each AFS client. This was much larger than we needed, as the total size of the files we had stored on the AFS server was less than this, however it was small enough that it did not constrain any other applications on our clients and allowed us to be able to easily add to our application without having to worry about increasing the size of the AFS client cache.

Running the Application with WAS 2.0

Just before we had completed our testing, the 2.0 version of Websphere Application Server became available to us. Even though we had recently migrated to WAS 1.1 we decided to upgrade our environment to WAS 2.0 for our final long run testing. The migration to WAS 2.0 was very similar to that done for 1.1, which is documented earlier in this report. There are some differences between WAS 2.0 and previous releases, such as directory structure and naming of log files. These will not be discussed in this report

In our test we had three client machines that simulated Internet users of our application. They had been automated to continually logon to the INSCO application as random users, and make various requests to view their policies, review their personal information and to update this information. The three clients were set to run continuously for three days and were checked periodically. No problems were found during this time, and at the end of the three days the clients were still running the application and accessing the database successfully.

Appendix A: Products and Versions Used

The list below is the specific products and version used for our solution. In general, for our tests we use the latest version of the software available, including product betas. Unless explicitly stated, earlier versions of software may work, but were not part of this test scenario.

Client
Network Station 1000 running Navio NC Navigator 3.04
Windows 95 Netscape Navigator 3.0, 4.01, and 4.05
Windows 95 Internet Explorer 4.0, and 3.0
Windows NT Netscape Navigator 3.0, 4.03, and 4.06
Windows NT Internet Explorer 3.02, 4.0

Note: if you are using Internet Explorer Version 3.x levels, you must use certificates from a known 3rd party. Domino Go generated certificates cannot be used.

Firewall
IBM eNetwork Firewall 3.2 for AIX

Middle Tier
Windows NT 4.0 with Service Pack 3
AIX 4.2.1
Sun JDK 1.1.6 for Windows NT
IBM JDK 1.1.4 for AIX
IBM WebSphere Performance Pack 1.0 Components:
  eNetwork Dispatcher 2.0 for Windows NT
  AFS client 3.4a Windows NT and AIX
IBM WebSphere Application Server 1.01 for AIX
IBM WebSphere Application Server 1.01, 1.1 and 2.0 for Windows NT
Lotus Domino Go 4.6.2.5 for Windows NT and AIX
  Note: Domino Go 4.6.2.2 will not work with Internet Explorer 4.0
Microsoft IIS 3.0 for Windows NT
DB2 5.2 Client Application Enabler for Windows NT and AIX

Note: you cannot use DB2 UDB 5.0 Server with a DB2 5.2 Client without applying at least the first fix pack on the server

Third Tier
AIX 4.2.1
IBM WebSphere Performance Pack 1.0 Components:
  AFS server 3.4a AIX
DB2 UDB Server 5.2 for AIX

Note: you cannot use DB2 UDB 5.0 Client with a DB2 5.2 Server without applying at least the first fix pack on the client

Application Development
IBM WebSphere Application Server 1.01 for WindowsNT
Sun JDK 1.1.6 for Windows NT

Standard Editor

Appendix B: Source code for WASpolicyServlet

```
// *****************************************************************
// * WASpolicyServlet.java                                         *
// *****************************************************************
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;
import com.ibm.servlet.connmgr.*;
import com.sun.server.http.*;
import java.beans.Beans;
import ABean;

public class WASpolicyServlet extends HttpServlet
{

  // Use to communicate with connection manager.
  static IBMConnMgr connMgr = null;

  // Use later in setConnection() to create JDBC connection specification.
  static IBMConnSpec spec   = null;      // the spec
  static String DbName      = null;      // database name
  static String Db          = "db2";     // JDBC subprotocol for DB2
  static String poolName    ="JdbcDb2"; // from Webmaster
  static String jdbcDriver = "COM.ibm.db2.jdbc.app.DB2Driver";
  static String url         = null;      // constructed later
  static String owner       = null;      // table owner

  // Name of property file used to complete the DbName, and table
  // owner information at runtime.  ".properties" extension assumed.

  static final String CONFIG_BUNDLE_NAME = "login";


  // **************************************************************
  // * Initialize servlet when it is first loaded                 *
  // **************************************************************

public void init(ServletConfig config) throws ServletException
 {
    super.init(config);
    try
    {
      // Get information at runtime (from an external property file
      // identified by CONFIG_BUNDLE_NAME) about the database name
      // and the associated database user and password.  This
```

```
        // information could be provided in other ways.  It could be
        // hardcoded within this application, for example.

        PropertyResourceBundle configBundle =
        (PropertyResourceBundle)PropertyResourceBundle.
        getBundle(CONFIG_BUNDLE_NAME);
        DbName   = configBundle.getString("JDBCServlet.dbName");
        url      = "jdbc:"+ Db +":"+ DbName;
        owner    = configBundle.getString("JDBCServlet.dbOwner");
      }
      catch(Exception e)
      {
        System.out.println("read properties file: "+ e.getMessage());
      }

        // **********
        // * STEP 2  *
        // **********

      // Get a reference to the connection manager.
      try
      {
        connMgr = IBMConnMgrUtil.getIBMConnMgr();
      }
      catch(Exception e)
      {
        System.out.println("set connection spec, get connection manager: "+
                     e.getMessage());
      }
    } // End of Init


    // ***************************************************************
    // * Respond to user Post request                                *
    // ***************************************************************

  public void doPost(HttpServletRequest req, HttpServletResponse res)
   {
    try
    {
          System.out.println("Made it to post!" + req.getParameter("homebutton"));
       ABean ab;

          System.out.println("before instantiating the session");
       ab = (ABean) Beans.instantiate(null, "ABean");
       ((com.sun.server.http.HttpServiceRequest)req).setAttribute("ab", ab);
          HttpSession session = req.getSession(true);
    if (req.getParameter("homebutton").equals("Logon"))
    {
      logon(session,req, res, ab);
    }
    else if (req.getParameter("homebutton").equals("View Policy"))
    {
      viewpolicy(session,req, res, ab);
    }

    else if (req.getParameter("homebutton").equals("Back to Home Page"))
    {
       gobackhome(session,req, res, ab);
    }

    else if (req.getParameter("homebutton").equals("Update Personal Information"))
    {
      updatepolicy(session,req, res, ab);
    }

    else if (req.getParameter("homebutton").equals("Submit Changes"))
    {
      submitchanges(session,req, res, ab);
    }

    else if (req.getParameter("homebutton").equals("Undo Changes"))
    {
      undochanges(session,req, res, ab);
   }

        else if (req.getParameter("homebutton").equals("Logoff"))
    {
      logoff(session,req, res, ab);
     }

        else
   {
          ab.seterrortype("An invalid button was pushed.");
      errorhandle(req, res, ab);
   }
  } //end of try

  catch(Exception e)
   {
      System.out.println("Error is:"+ e.getMessage() +" and"+ e.toString());
   }


} // doPost()
```

```
/***********************************************************************/
/* setConnection method                                                */
/***********************************************************************/

public Connection setConnection(HttpSession asession)
 {
    IBMJdbcConn cmConn   = null;
    Connection dataConn = null;

    try
    {
      String user = (String) asession.getValue("WASpolicyServlet.user");
      String password = (String) asession.getValue("WASpolicyServlet.password");

      // **********
      // * STEP 1  *
      // **********
      // Create JDBC connection specification.

      spec = new IBMJdbcConnSpec
            (poolName,    // pool name from Webmaster
             true,        //  waitRetry
             jdbcDriver,
             url,
             user,
             password);
 System.err.println("Setting up spec using the following: "+ poolName +"(poolname), "+
            jdbcDriver +" (jdbcDriver), "+ url +" (url),"+ user +"(userid) "+
                   password +" (password).");
      // **********
      // * STEP 3  *
      // **********
      // Get an IBMJdbcConn object (cmConn) meeting JDBC
      // connection specs, from the connection manager pool.

      cmConn = (IBMJdbcConn)connMgr.getIBMConnection(spec);
         asession.putValue("WASpolicyServlet.cmConn",cmConn);

      // **********
      // * STEP 4  *
      // **********
      // Get a Connection object (dataConn).  This is
      // an object from the java.sql package and it is used
      // for JDBC access.

      dataConn = cmConn.getJdbcConnection();


     }
     catch (Exception ex)
```

```
        {
          System.out.println("Can't setConnection "+ ex.getMessage());
        }
System.out.println("endof setconnection");
     return(dataConn);

    } // end of setConnection


/************************************************************************/
/* Logon method                                                         */
/************************************************************************/

public void logon(HttpSession asession, HttpServletRequest areq, HttpServletResponse ares,
                    ABean abean)
 {
        Connection adataConn = null;
        IBMJdbcConn acmConn = null;
             String firstname = null;
             String cl_mi = null;
        String lastname = null;
             String address = null;
        String cl_phone_h = null;
             String cl_phone_w = null;
             String cl_phone_f = null;
        String cl_email = null;
     try
     {
            String user =areq.getParameter("userid");
        asession.putValue("WASpolicyServlet.user",user);
        String password =areq.getParameter("password");
            asession.putValue("WASpolicyServlet.password",password);
        adataConn = setConnection(asession);
            acmConn = (IBMJdbcConn) asession.getValue("WASpolicyServlet.cmConn");

        // **********
        // *  STEP 5  *
        // **********
        // Run DB query

        Statement stmt = adataConn.createStatement();
        String query = "Select firstname, cl_mi, lastname, address,"+
                      "cl_phone_h, cl_phone_w, cl_phone_f, cl_email "+
                        "from"+ owner +".SEL"+ user +
                       "  where clientno ="+ user;
        ResultSet rs   = stmt.executeQuery(query);

             if(rs.next())
        {
           firstname = rs.getString(1);
        cl_mi = rs.getString(2);
     lastname = rs.getString(3);
        address = rs.getString(4);
     cl_phone_h = rs.getString(5);
        cl_phone_w = rs.getString(6);
        cl_phone_f = rs.getString(7);
     cl_email = rs.getString(8);

        asession.putValue("WASpolicyServlet.firstname",firstname);
        asession.putValue("WASpolicyServlet.cl_mi",cl_mi);
        asession.putValue("WASpolicyServlet.lastname",lastname);
        asession.putValue("WASpolicyServlet.address",address);
        asession.putValue("WASpolicyServlet.cl_phone_h",cl_phone_h);
        asession.putValue("WASpolicyServlet.cl_phone_w",cl_phone_w);
        asession.putValue("WASpolicyServlet.cl_phone_f",cl_phone_f);
        asession.putValue("WASpolicyServlet.cl_email",cl_email);

        asession.putValue("WASpolicyServlet.origfirstname",firstname);
        asession.putValue("WASpolicyServlet.origcl_mi",cl_mi);
        asession.putValue("WASpolicyServlet.origlastname",lastname);
        asession.putValue("WASpolicyServlet.origaddress",address);
        asession.putValue("WASpolicyServlet.origcl_phone_h",cl_phone_h);
        asession.putValue("WASpolicyServlet.origcl_phone_w",cl_phone_w);
        asession.putValue("WASpolicyServlet.origcl_phone_f",cl_phone_f);
        asession.putValue("WASpolicyServlet.origcl_email",cl_email);
   }

   else
   {
        abean.seterrortype("There has been a logon failure.");
        errorhandle(areq, ares, abean);
   }

   // Invoke close() on stmt, which also closes rs, freeing
   // resources and completing the interaction.  You must
   // not, however, close the dataConn object.  It must
   // remain open and under the control of connection manager
   // for possible use by other requests to this servlet or
   // to other servlets.
   stmt.close();

// **********
// * STEP 6  *
// **********
// Release the connection back to the pool.
   acmConn.releaseIBMConnection();


  System.err.println("Logon completed successfully :)");
  abean.setbeanfirstname(firstname);
```

```
       abean.setbeanlastname(lastname);

      ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscohome.jsp",areq);

    }
    catch (Exception ex)
    {
       System.out.println("Error in logon"+ ex.getMessage());
       System.out.println(ex);
       ex.printStackTrace(System.out);
       abean.seterrortype("There has been a logon failure.  Please verify your userid and/or password and
try again.");
          errorhandle(areq, ares, abean);
    }

  } // End of logon()


/**********************************************************************/
/* updatepolicy method                                                */
/**********************************************************************/

public void updatepolicy(HttpSession asession, HttpServletRequest areq, HttpServletResponse ares,
                         ABean abean)
  {
System.out.println("start of updatepolicy");
    try
    {
       String firstname = (String) asession.getValue("WASpolicyServlet.firstname");
          String cl_mi      = (String) asession.getValue("WASpolicyServlet.cl_mi");
          String lastname   = (String) asession.getValue("WASpolicyServlet.lastname");
          String address    = (String) asession.getValue("WASpolicyServlet.address");
          String cl_phone_h = (String) asession.getValue("WASpolicyServlet.cl_phone_h");
          String cl_phone_w = (String) asession.getValue("WASpolicyServlet.cl_phone_w");
          String cl_phone_f = (String) asession.getValue("WASpolicyServlet.cl_phone_f");
          String cl_email   = (String) asession.getValue("WASpolicyServlet.cl_email");

       abean.setbeanfirstname(firstname);
       abean.setbeancl_mi(cl_mi);
       abean.setbeanlastname(lastname);
       abean.setbeanaddress(address);
       abean.setbeancl_phone_h(cl_phone_h);
       abean.setbeancl_phone_w(cl_phone_w);
       abean.setbeancl_phone_f(cl_phone_f);
       abean.setbeancl_email(cl_email);
   System.out.println("callpage of updatepolicy");
      ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscoupdate.jsp",areq);
System.out.println("after callpage of updatepolicy");
    }
    catch (Exception ex)
    {
      System.out.println("Error in updatepolicy" + ex.getMessage());
      System.out.println(ex);
      ex.printStackTrace(System.out);
      abean.seterrortype("There has been an update failure.");
      errorhandle(areq, ares, abean);
    }

  } // End of updatepolicy()


/**************************************************************************/
/* viewpolicy method                                                      */
/**************************************************************************/

public void viewpolicy(HttpSession asession, HttpServletRequest areq, HttpServletResponse ares,
                       ABean abean)
 {
   Connection adataConn;
   IBMJdbcConn acmConn;

   try
   {
     String user = (String) asession.getValue("WASpolicyServlet.user");
     String firstname = (String) asession.getValue("WASpolicyServlet.firstname");
     String lastname = (String) asession.getValue("WASpolicyServlet.lastname");

     abean.setbeanfirstname(firstname);
     abean.setbeanlastname(lastname);
     System.err.println("The policy is being viewed by " +firstname + lastname);
     adataConn = setConnection(asession);
     acmConn = (IBMJdbcConn) asession.getValue("WASpolicyServlet.cmConn");

    // Run DB query
     Statement stmt = adataConn.createStatement();
     String query = "Select policy from"+ owner +".POL"+ user;

     ResultSet rs   = stmt.executeQuery(query);
     if(rs == null)
     {
       abean.seterrortype("No policy currently exists for"+ firstname +" "+ lastname +".");
       errorhandle(areq, ares, abean);
     }
     else
     {
       abean.setbeanresult(rs);
       ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscoview.jsp",areq);
     }
     stmt.close();
    acmConn.releaseIBMConnection();

   }
   catch (Exception ex)
   {
      System.out.println(ex);
      ex.printStackTrace(System.out);
      abean.seterrortype("The system is unable to view the policy at this time");
      errorhandle(areq, ares, abean);
   }

  } // End of viewpolicy()


/**************************************************************************/
/* gobackhome method                                                      */
/**************************************************************************/

public void gobackhome(HttpSession asession, HttpServletRequest areq, HttpServletResponse ares,
                 ABean abean)
 {
  try
  {
    String firstname = (String) asession.getValue("WASpolicyServlet.firstname");
    String lastname = (String) asession.getValue("WASpolicyServlet.lastname");

    abean.setbeanfirstname(firstname);
    abean.setbeanlastname(lastname);

    ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscohome.jsp",areq);
   }
   catch (Exception ex)
   {
     System.out.println(ex);
     ex.printStackTrace(System.out);
     abean.seterrortype("The home page is not available at this time.");
     errorhandle(areq, ares, abean);
    }

 } // End of gobackhome

/**************************************************************************/
/* submitchanges method                                                   */
/**************************************************************************/

public void submitchanges(HttpSession asession, HttpServletRequest areq, HttpServletResponse ares,
                 ABean abean)
 {
    Connection adataConn;
    IBMJdbcConn acmConn;
```

```
try
 {
     String user = (String) asession.getValue("WASpolicyServlet.user");
          String firstname =areq.getParameter("formfirstname");
          String cl_mi =areq.getParameter("formcl_mi");
          String lastname =areq.getParameter("formlastname");
          String address =areq.getParameter("formaddress");
          String cl_phone_h =areq.getParameter("formcl_phone_h");
          String cl_phone_w =areq.getParameter("formcl_phone_w");
          String cl_phone_f =areq.getParameter("formcl_phone_f");
          String cl_email =areq.getParameter("formcl_email");

     asession.putValue("WASpolicyServlet.firstname",firstname);
     asession.putValue("WASpolicyServlet.cl_mi",cl_mi);
     asession.putValue("WASpolicyServlet.lastname",lastname);
     asession.putValue("WASpolicyServlet.address",address);
     asession.putValue("WASpolicyServlet.cl_phone_h",cl_phone_h);
     asession.putValue("WASpolicyServlet.cl_phone_w",cl_phone_w);
     asession.putValue("WASpolicyServlet.cl_phone_f",cl_phone_f);
     asession.putValue("WASpolicyServlet.cl_email",cl_email);

     adataConn = setConnection(asession);
     acmConn = (IBMJdbcConn) asession.getValue("WASpolicyServlet.cmConn");

     Statement stmt = adataConn.createStatement();

     String query = "update" + owner +".SEL"+ user +
                              "set firstname = '"+ firstname +
                              "', cl_mi = '"+ cl_mi +
                              "', lastname = '"+ lastname +
                              "', address= '"+ address +
                              "', cl_phone_h = '"+ cl_phone_h +
                              "', cl_phone_w = '"+ cl_phone_w +
                              "', cl_phone_f = '"+ cl_phone_f +
                              "', cl_email = '"+ cl_email +
                    " ' where clientno =" + user;

   stmt.executeQuery(query);
   stmt.executeQuery("commit");

   abean.setbeanfirstname(firstname);
   abean.setbeancl_mi(cl_mi);
   abean.setbeanlastname(lastname);
   abean.setbeanaddress(address);
   abean.setbeancl_phone_h(cl_phone_h);
   abean.setbeancl_phone_w(cl_phone_w);
   abean.setbeancl_phone_f(cl_phone_f);
   abean.setbeancl_email(cl_email);
   ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscoupdated.jsp",areq);
```

```
     stmt.close();
     acmConn.releaseIBMConnection();
   }
   catch (Exception ex)
   {
     System.out.println(ex);
     ex.printStackTrace(System.out);
     abean.seterrortype("The system is unable to submit changes at this time.");
     errorhandle(areq, ares, abean);
    }

  } // End of submitchanges()

/*************************************************************************/
/* undochanges method                                                     */
/*************************************************************************/

public void undochanges(HttpSession asession, HttpServletRequest areq, HttpServletResponse ares,
                    ABean abean)
 {
    Connection adataConn = null;
    IBMJdbcConn acmConn;

   try
   {
      String user = (String) asession.getValue("WASpolicyServlet.user");
      String firstname = (String) asession.getValue("WASpolicyServlet.origfirstname");
          String cl_mi      = (String) asession.getValue("WASpolicyServlet.origcl_mi");
          String lastname   = (String) asession.getValue("WASpolicyServlet.origlastname");
          String address    = (String) asession.getValue("WASpolicyServlet.origaddress");
          String cl_phone_h = (String) asession.getValue("WASpolicyServlet.origcl_phone_h");
          String cl_phone_w = (String) asession.getValue("WASpolicyServlet.origcl_phone_w");
          String cl_phone_f = (String) asession.getValue("WASpolicyServlet.origcl_phone_f");
          String cl_email   = (String) asession.getValue("WASpolicyServlet.origcl_email");

      asession.putValue("WASpolicyServlet.firstname",firstname);
      asession.putValue("WASpolicyServlet.cl_mi",cl_mi);
      asession.putValue("WASpolicyServlet.lastname",lastname);
      asession.putValue("WASpolicyServlet.address",address);
      asession.putValue("WASpolicyServlet.cl_phone_h",cl_phone_h);
      asession.putValue("WASpolicyServlet.cl_phone_w",cl_phone_w);
      asession.putValue("WASpolicyServlet.cl_phone_f",cl_phone_f);
      asession.putValue("WASpolicyServlet.cl_email",cl_email);


      adataConn = setConnection(asession);
          acmConn = (IBMJdbcConn) asession.getValue("WASpolicyServlet.cmConn");

      Statement stmt = adataConn.createStatement();
```

```
        String query = "update" + owner +".SEL"+ user +
                                    "set firstname = '"+ firstname +
                                    "', cl_mi = '"+ cl_mi +
                                    "', lastname = '"+ lastname +
                                    "', address= '"+ address +
                                    "', cl_phone_h= '"+ cl_phone_h +
                                    "', cl_phone_w= '"+ cl_phone_w +
                                    "', cl_phone_f= '"+ cl_phone_f +
                                    "', cl_email = '" + cl_email +
                          " ' where clientno =" + user;

        stmt.executeQuery(query);
        stmt.executeQuery("commit");

        abean.setbeanfirstname(firstname);
        abean.setbeancl_mi(cl_mi);
        abean.setbeanlastname(lastname);
        abean.setbeanaddress(address);
        abean.setbeancl_phone_h(cl_phone_h);
        abean.setbeancl_phone_w(cl_phone_w);
        abean.setbeancl_phone_f(cl_phone_f);
        abean.setbeancl_email(cl_email);
      ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscoupdated.jsp",areq);

        stmt.close();
        acmConn.releaseIBMConnection();
      }
      catch (Exception ex)
      {
        System.out.println(ex);
        ex.printStackTrace(System.out);
        abean.seterrortype("The undochanges function is not available at this time.");
        errorhandle(areq, ares, abean);
       }

  } // End of undochanges()



/***************************************************************************/
/* logoff method                                                           */
/***************************************************************************/

public void logoff(HttpSession asession, HttpServletRequest areq, HttpServletResponse ares,
                    ABean abean)
 {

   try
    {
      String firstname = (String) asession.getValue("WASpolicyServlet.firstname");
```

```
            String lastname   = (String) asession.getValue("WASpolicyServlet.lastname");
            IBMJdbcConn acmConn = (IBMJdbcConn) asession.getValue("WASpolicyServlet.cmConn");

         abean.setbeanfirstname(firstname);
         abean.setbeanlastname(lastname);
      ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscologoff.jsp",areq);

         asession.invalidate();
      }
      catch (Exception ex)
      {
         System.out.println(ex);
         ex.printStackTrace(System.out);
         abean.seterrortype("There has been a logoff failure.  Please close all instances of your browser so
that the session gets properly invalidated.");
         errorhandle(areq, ares, abean);
      }

  } // End of logoff()


/*************************************************************************/
/* errorhandle method                                                    */
/*************************************************************************/

public void errorhandle(HttpServletRequest areq, HttpServletResponse ares, ABean abean)
 {

   try
    {
     ((com.sun.server.http.HttpServiceResponse)ares).callPage("/jsp/inscoerror.jsp", areq);
    }
    catch (Exception ex)
     {
       System.out.println("error in errorhandle"+ ex.getMessage());
       System.out.println(ex);
       ex.printStackTrace(System.out);
     }
  } // End of errorhandle()

} // End of WaspolicyServlet
```

Appendix C: Source code for ABean

```
import java.sql.*;

public class ABean {

  private ResultSet beanresult;
  private String beanfirstname;
  private String beancl_mi;
  private String beanlastname;
  private String beanaddress;
  private String beancl_phone_h;
  private String beancl_phone_w;
  private String beancl_phone_f;
  private String beancl_email;
  private String beanerrortype;

  public void setbeanresult(ResultSet rs) { beanresult = rs; }
  public ResultSet getbeanresult() { return beanresult; }

  public void setbeanfirstname(String s) { beanfirstname = s; }
  public String getbeanfirstname() { return beanfirstname; }

  public void setbeancl_mi(String s) { beancl_mi = s; }
  public String getbeancl_mi() { return beancl_mi; }

  public void setbeanlastname(String s) { beanlastname = s; }
  public String getbeanlastname() { return beanlastname; }

  public void setbeanaddress(String s) { beanaddress = s; }
  public String getbeanaddress() { return beanaddress; }

  public void setbeancl_phone_h(String s) { beancl_phone_h = s; }
  public String getbeancl_phone_h() { return beancl_phone_h; }

  public void setbeancl_phone_w(String s) { beancl_phone_w = s; }
  public String getbeancl_phone_w() { return beancl_phone_w; }

  public void setbeancl_phone_f(String s) { beancl_phone_f = s; }
  public String getbeancl_phone_f() { return beancl_phone_f; }

  public void setbeancl_email(String s) { beancl_email = s; }
  public String getbeancl_email() { return beancl_email; }

  public void seterrortype(String s) { beanerrortype = s; }
  public String geterrortype() { return beanerrortype; }

}
```

Appendix D: JSP Source for INSCOHome.jsp

Here is one of the JSP files we used in our application: theinscohome.jsp.

```
<html>
<head>
<% response.setHeader("Pragma","No-Cache");
response.setDateHeader("Expires",0);
response.setHeader("Cache-Control","no-Cache"); %>
<title>INSCO Home Page</title>
</head>
<%@ import="ABean" %>
<bean name="ab" type="ABean" introspect="no" create="no" scope="request">
</bean>
<!--------------------------------------------------------------->
<body BGCOLOR="lightgreen">
<CENTER>
<h2>INSCO Home Page</h2>
<form action="/servlet/WASpolicyServlet"method=post>
<H2>Welcome <%= ab.getbeanfirstname() %>  <%= ab.getbeanlastname() %>! </H2>
<HR>
<TABLE>
<TR>
<TD>&nbsp</TD><TD>&nbsp</TD>
<TR><TD><H4>To view your insurance policy..................................................</H4></TD>
<TD><input name="homebutton"value="View Policy" type="submit"></TD>

<TR><TD><H4>To make name, address, phone number andemail changes...</H4></TD>
<TD><input name="homebutton"value="Update Personal Information" type="submit"></TD>

<TR><TD><H4>To undo changes you made during thissession.......................</H4></TD>
<TD><input name="homebutton"value="Undo Changes" type="submit"></TD>

<TR><TD><H4>To log you off from this session...............................................</H4></TD>
<TD><input name="homebutton"value="Logoff" type="submit"></TD></TR>
</TABLE>
</CENTER>
<HR>
</form>
<H4> To contact a customer service representative, please call your local
office or send us email! <A HREF="mailto:custservice@insco.com"><img
src="/mailto.gif"></a></H4>
</body>
</html>
```

Appendix E: login.properties file

Here are the contents of our login.properties file. This file should be placed in the same directory as the servlet.

```
# @(#)login.properties

JDBCServlet.dbOwner=insco
JDBCServlet.dbName=inscodb
```

Appendix F: eND Configuration File

The following is the contents of our eND configuration file:

```
rem
rem pie1.cmd - eND config file for INSCO Scenario
rem

CALL ndcontrol executor start
CALL ndconfig en0 alias 9.67.142.69 netmask 255.255.255.240
CALL ndcontrol executor set nfa 9.67.142.77
CALL ndcontrol cluster add 9.67.142.69

CALL ndcontrol port add 9.67.142.69:80
CALL ndcontrol port add 9.67.142.69:443

CALL ndcontrol port set 9.67.142.69:443 stickytime 180
CALL ndcontrol port set 9.67.142.69:443 staletimeout 300

CALL ndcontrol server add 9.67.142.69:80:9.67.142.65
CALL ndcontrol server add 9.67.142.69:80:9.67.142.67
CALL ndcontrol server add 9.67.142.69:80:9.67.142.74

CALL ndcontrol server add 9.67.142.69:443:9.67.142.65
CALL ndcontrol server add 9.67.142.69:443:9.67.142.67
CALL ndcontrol server add 9.67.142.69:443:9.67.142.74

CALL ndcontrol manager start
CALL ndcontrol manager proportions 45 45 10 0
CALL ndcontrol advisor start HTTP 80
CALL ndcontrol advisor start SSL 443
```

Appendix G: Shell Script for Creating DB2 views

Here are the contents of *inscoviews.sh*:

```
#!/bin/ksh

custid=1
db2 connect to inscodb user db2inst1 using db2inst1
while [ $custid -lt 9518 ]
do
 db2 "create view insco.sel$custid as select * from insco.clients where clientno=$custid"
 db2 "create view insco.pol$custid as select * from insco.spolicy where policynumber in (select
policynumber from insco.policydb where clientno=$custid)"
 db2 grant select on insco.sel$custid to user \"$custid\"
 db2 grant select on insco.pol$custid to user \"$custid\"
 db2 grant update on insco.sel$custid to user \"$custid\"
 custid=‘expr $custid + 1‘
done
```

Appendix H: Notices and Trademarks