Web Based Client/Server Computing

by

Hoony C. Youn

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering in Electrical Engineering and Computer Science

and

Bachelor of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

October 1995

Signature of Author...... ....
nd Computer Science
October 3, 1995

Certified by..
Richard Y. Wang
Associate Professor of Information Technology
Thesis Supervisor

Accepted ....
Frederic R. Morgenthaler
Chairman, Department Committee on Graduate Theses

DEDICATION

I dedicate my thesis to my parents. Mom and Dad, I thank you for all of your support and encouragement through my years at MIT.

ACKNOWLEDGMENTS

I would like to thank Jesse Jacobson for guiding me through this thesis process. Thanks for all of your time and support. Last, but not least, I would also like to thank my thesis advisor, Professor Rich Wang, for your time and encouragement. Thanks for pushing me hard and always keeping me focused on the task at hand.

Web Based Client/Server Computing

by

Hoony C. Youn

Submitted to the Department of Electrical Engineering and Computer Science on
October 3, 1995, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Electrical Engineering and Computer Science
and
Bachelor of Science in Electrical Engineering and Computer Science

Abstract

The fastest growing Internet application today is the World Wide Web. A user can get around the World Wide Web through a Web browser. With a Web browser a user has access to a wealth of information provided either free or for a fee. Much of the information that can be obtained from the Web can greatly help businesses as well as individuals with their research. The Web browser, however has one disadvantage. It is not capable of doing any processing of the data (thus failing to utilize the client side resources). Once the data is received on the server sided, it is just displayed. If there is any processing being done, it is done on the server side. By creating a custom application that talks to a Web server, the information can be processed with the client side resources.

The first part of this thesis was to create a WWW application that would allow a user to access a back-end database from their Web browser. The scenario used for this thesis was a fictional home banking system that would allow users to check their balances, recent account activity, and do other simple bank tasks. The second part of this thesis was to create a customized front-end that would communicate with a Web server and process the returned information locally (versus making another request to the server to further process the information). The rationale behind this idea is to remove the processing load off of the remote Web server and bring it to the client side which is in most cases almost as powerful as the Web servers out there, thus making a more distributed computing environment.

Thesis Supervisor: Richard Y. Wang
Title: Associate Professor of Information Technology

TABLE OF CONTENTS

Web Based Client/Server Computing

1. Introduction

The Internet, a word that has been getting a lot of hype recently, has been around for over 30 years. Until recently, the users of the Internet were primarily academics and the science community. In 1993, the World Wide Web (WWW) was introduced. This was the beginning of a revolution.

The WWW allows users for almost all computer platforms to access information on the Internet with a graphical user interface (GUI). Because it was built on top of TCP/IP, it can be accessed from anywhere on the Internet. The WWW consists of countless servers that are connected to the Internet. These servers can be accessed through browsers (clients such as Netscape or Mosaic) which run on all of the common platforms (PC, Macintosh, UNIX).

Before the introduction of WWW, to access the Internet, users were forced to use terminal type emulators which were text based. This was not very user friendly for the majority of the public. With the GUI, the door was open for the majority of computer users/nonusers to access the Internet. With the GUI, there is little or no learning curve. If one can learn how to point and click a mouse, then he/she is well on her way on becoming an expert "Net Surfer."

Now there is a very easy way of connecting to the Internet: the wonderful Web browser (like Netscape and Mosaic). But in reality, all the Web browser does is **display** formatted text and certain formatted images. The keyword is **display,** nothing more. The way the information is presented is somewhat graphical, but it is very limited. Therefore, a Web browser is nothing more than a bona-fide dumb terminal. The main drawback of a Web browser is that it fails to take advantage of over 15 years of rapid PC application development. When using a Web browser from a PC, the only resources that the Web browser uses are the hard drive, RAM for caching and communications software. By creating a customized intelligent front-end application that communicates with a Web Server, it can be made to exploit the PC's resources as well as the application's functionality.

1.1 The Goal

The first part of this thesis was to create a WWW application that would allow a user to access a back-end database from their Web browser. The scenario used for this thesis was a fictional home banking system that would allow users to check their balances, recent account activity, and do other simple bank tasks. The second part of this thesis was to create a customized front-end that would communicate with a Web server and process the returned information locally (versus making another request to the server to further process the information). The rationale behind this idea is to remove the processing load off of the remote Web server and bring it to the client side which is in most cases almost as powerful as the Web servers out there, thus making a more distributed computing environment.

The home banking system, which was named Web Banking, has three major components: the front-end (Web browser or PowerBuilder GUI), the Web Server, and the RDBMS. (See Figure 1-1.) All of the components in the system will be explained in much detail throughout this thesis.

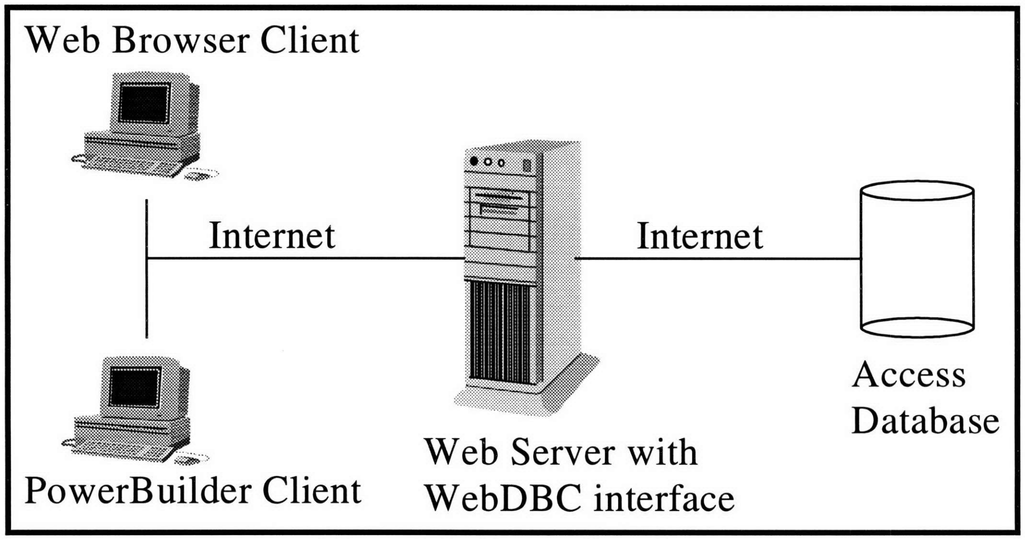


Figure 1-1: System Overview

Here is also a brief table that gives a summary of advantages and disadvantages of a Web browser client versus a customized PowerBuilder client.

Front End	Advantages	Disadvantages
Web Browser Client	Can access all Web sites	Does not allow processing to be done on client-side
PowerBuilder Client	Can do data processing and manipulation	Can only access few Web sites

Table 1-1: Web Browser vs. PowerBuilder Client

1.2 Overview

Chapter 2 will discuss the background of the field involving this thesis. Chapter 3 will discuss the components used in this thesis. Chapter 4 will discuss the implementation process of the system. Chapter 5 will discuss the final conclusions and further thoughts.

2. Background

The World Wide Web uses the **HyperText Transfer Protocol** also known as HTTP. This protocol is a higher level protocol that runs on top of TCP/IP. The browsers view documents (or files) that are typically in **HyperText Markup Language** (HTML) format. These HTML files are specially formatted text with references to specific image formats. When viewing an HTML document there are highlighted text (HyperText) that will reroute the user to another section on the same document or another document that can either be residing on the same server or a different server. As mentioned before, the WWW servers can be anywhere where there is a connection to the Internet. Therefore, it is possible for a user to jump from a server in Berkeley, California, to a server in London, England, and then to another server in Singapore. The HyperText that the user clicks on to jump to another location triggers the browser to jump to another address which is known as the **Universal Resource Locator** (URL).

When viewing an HTML document, the document physically resides on the client's machine. The browser, however, is not just limited to communicating to WWW servers. It can communicate with ftp and gopher servers which greatly expands the amount of information that can be accessed from one single GUI. It can also display graphic images such as Graphics Interchange Format (GIF) and Joint Photograph Experts Group (JPEG) images within HTML files or by themselves. Both of these graphic formats offer the same picture quality, differing only in compression methods. JPEG is known to have better compression than GIF at the cost of image loading time.

Because of recent rapid growth in the number of Internet users, companies already on the Internet can reach more people with no additional effort. Other companies who are not on the Internet realize this as well, and as a result, are in the process of getting connected the Internet. The World Wide Web is not only a place where to acquire information, it is also a turning into a marketplace, where companies can sell services on the Internet.

Because of the potential that the Internet has of being a huge marketplace, companies are always looking for ways to provide new services on the Internet. Since a lot of these services involve providing specific information that has already been formatted and organized in to a relational database, it would be too much work to try to replicate this information into HTML documents. To solve this problem, tools have been developed that allow a WWW server to communicate with back-end databases like Oracle, Sybase, and Informix.

This thesis employs one of the tools (database access tool) which allows a user to access a back-end database through a WWW server. With the ability to access back-end databases through the WWW servers, the Web browsers have even more added value. Not only can users access data from servers, but they can directly input data to the same database servers (with specific privileges) through HTML forms, which can greatly reduce tedious manual data entry hours. These are only two of many applications that can be made as a result of integrating a Web browser client with a Web Server that can communicate with a back-end database. Now, there can be more services available on the Internet.

The scenario used to drive the development of the application for this thesis is a fictional Web banking system. As mentioned before, the Web browser is basically a dummy front-end GUI (meaning that it does not do any processing or computations on the client side). It is only capable of displaying text and special images. Because a good banking application involves a lot of data transfer, a good presentation method, and the ability to manipulate data, there is a need for an alternative front-end. A front-end that could access the same information from the Web Server that would be customized to fit bank customers' needs. Therefore, two system prototypes were developed for this thesis.

The first prototype was developed using a browser as the front-end to the Web Server. This application is geared for the casual or inexperienced user. The second prototype was developed using PowerBuilder as the front-end GUI. Because PowerBuilder was used in the second prototype, some of the features of PowerBuilder were implemented into the prototype which made the PowerBuilder front-end far superior to the Web Browser front-end. One such feature of PowerBuilder that was implemented was the graphing feature. After the user has viewed the information in the table, he/she may wish to see the information in a chart format. With the PowerBuilder interface, this can be done instantly because the processing (creation of the chart) is done locally. Graphing is possible with a Web browser front-end, but the graph has to be created remotely (on the Server) and then is converted from its native format to GIF format so that it can be displayed by a Web browser. Although the server is a high output machine, this is a processor-intensive task. GIF files, however, tend to be large in size and therefore will take more time to send it back to the user requesting for a chart. The end result is the performance of the overall system (Web browser application) will be much slower than the PowerBuilder application. Although the PowerBuilder front-end is far superior to the Web browser front-end, it does have one limitation. It is not generic, (meaning that it cannot browse the Web and view other home pages) but customized which is precisely the reason it is superior for this particular application. PowerBuilder could be used to make a generic GUI like Netscape, but that was not the purpose of this second prototype.

There were two parts to this thesis. The first part of this thesis was to build an application on the Web that utilizes tools that will allow a WWW server to communicate with a back-end database. The second part of this thesis was to offer an alternative approach to accessing data on the web (other than through a Web browser). This more advanced front-end (PowerBuilder) is intelligent because it uses local (client-side) resources to display objects and manipulate/process information as opposed to the browser which only displays information.

3. Description of Components

3.1 Distributed Computing Environment

A distributed computing environment exists when server pieces of a particular application run on different machines in a network. A typical network might include PC's, MACs, UNIX workstations, NT server, or OS/2 servers. (See Figure 3-1.) Because there are many different machines with different operating systems connected to the same network, there must be a common language that every computer speaks so that they can exchange data back and forth. This common language is implemented through software that each computer has and this software is referred to as the transport layer. Without this transport layer, the network cannot function. A network assumes a transport layer which will allow file transfer (from one machine to another) as well as accessing data from one machine to another.

Because the scope of this thesis dealt with the Internet, the transport layer adopted for this thesis was TCP/IP. Another reason why TCP/IP might be a better choice for a transport layer is that all OS platforms have TCP/IP software, whereas other transport layer software are only available on some of the platforms.

There are several advantages that distributed computing has over single-system computing. Probably the most important advantage is accessing distributed data. In many companies data exists in all kinds of formats on many different machines. This encompasses utilizing the current computing resources to meet their information needs. In other words, a company would not have to throw away certain computers because of compatibility issues.

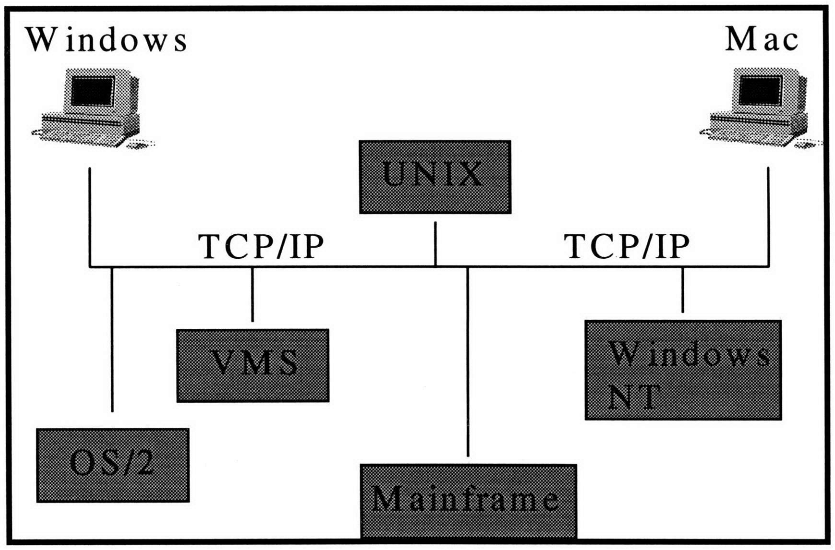
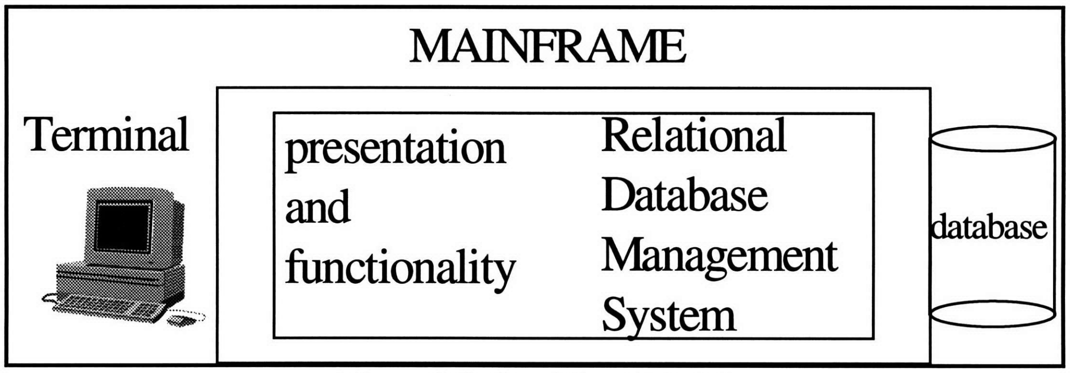


Figure 3-1: A Distributed Computing Network

There are three types of client/server models that illustrate the benefits of a distributed computing environment.

3.1.1 One-Tiered Model

A one-tiered model is the classic mainframe/terminal model where one single machine satisfies the users. As can be seen from the Figure 3-2, all of the functionality, presentation, and the DBMS's reside on the mainframe. This provides centralized control for the administrators as well as a secure environment. The client, however is a dumb terminal, meaning that it does not do any presentation or processing. Everything is done on the mainframe and the information is just fed back to the terminal.


MAINFRAME
Terminal
presentation and functionality
Relational Database Management System
database
Figure 3-2: One-Tiered Model

The advantages of this system is that it is much easier to maintain from a systems administrator point of view because it is centralized. However, there are certain disadvantages to the system for today's business environment. One of the most significant drawbacks to this architecture is that the applications do not scale.

Since the host machine is limited to the number of processes it can run at one given time, there is only a finite number of users that can access the system at one time. Moreover, once the maximum capacity is reached, another machine must be purchased.

Another, perhaps more significant, disadvantage to this architecture is that one-tiered computing leads to hardware lock-in. This means that the system is running proprietary software usually made by the hardware vendor. Once that version is installed, the system administrator is tied to the particular vendor for support, upgrades and help.

3.1.2 Two-Tiered Model

The main difference between the one-tiered model and the two-tiered model is that the application runs on two separate processes. (See Figure 3-3). The client, the first process, will make a request to the server, another process. Many times the server is usually an RDBMS. The client would be something like a PowerBuilder GUI which requests data from an Oracle Server and upon retrieving it, displays it in a graphical manner (tables and charts). In this example, the processing is done at both ends.

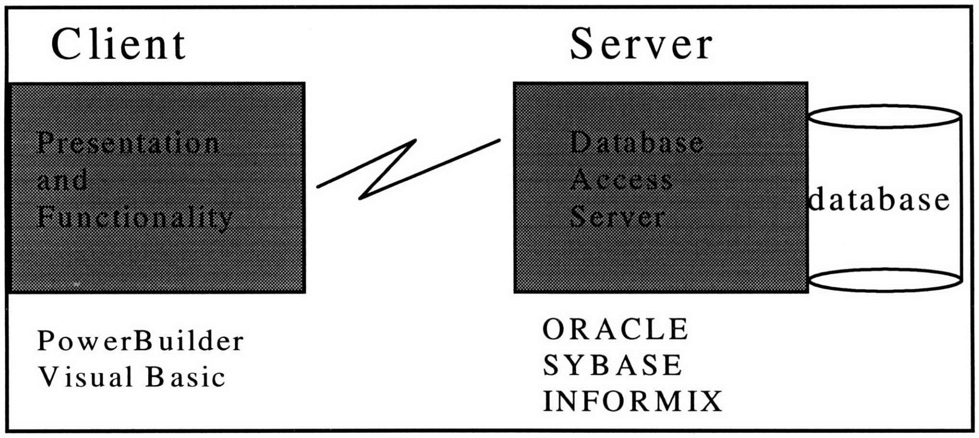


Figure 3-3: Two-Tiered Model

“Drawbacks to two-tiered computing are that applications tend to rely heavily on hard-coded SQL in the form of proprietary stored procedures, and the applications only scale to a point. With too many users, the applications run very slowly.” (Open Environment Corporation, p9)

The main drawback, like the one-tiered model, deals with lock-in, but in this case it is software lock-in. Once a company commits to one software package, they are locked-in to the software for the life of it.

3.1.3 Three(Multi)-Tiered Model

A multi-tiered computing environment breaks up the applications into three (or more) levels (see Figure 3-4). The front layer handles the presentation, the functionality part has moved to the middle layer, and the back end is the databases and legacy systems. One of the biggest advantages of this strategy is the increased interoperability among operating systems, client programs, server languages, RDBMS's, and legacy systems. Also, this architecture is modular and scaleable.

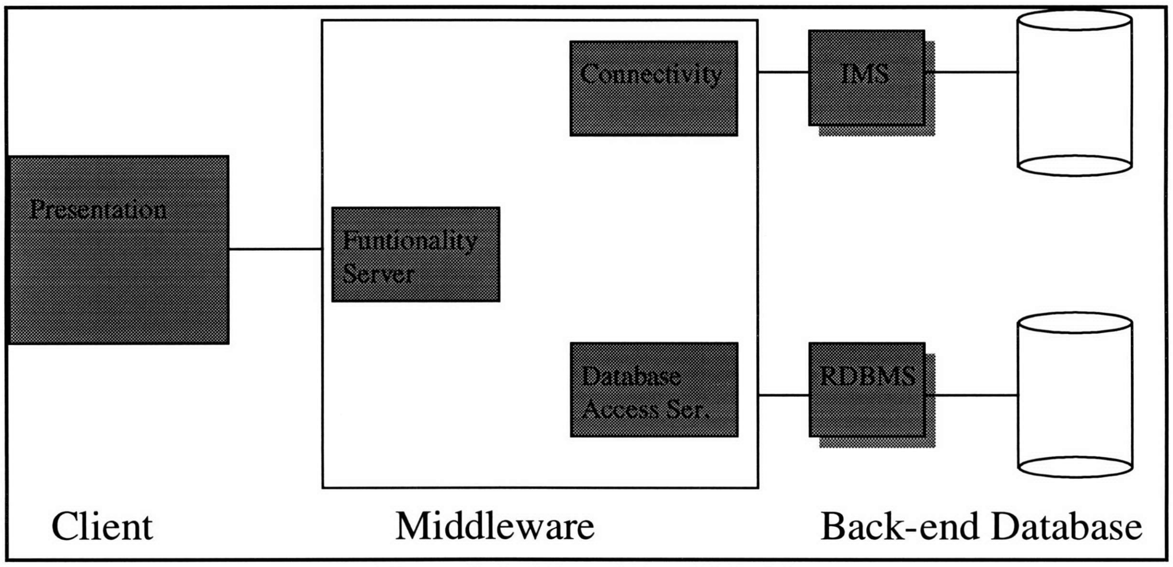


Figure 3-4: Three-Tiered Model

The functionality servers can be written in a variety of languages and can reside on a variety of operating systems. Probably the most significant advantage of this architecture is that "it solves the major drawbacks of both the one-tiered environments (hardware lock-in) and two-tiered environments (software lock-in) by allowing any machine to talk to any other machine without requiring a specific software product for this communication. (Open Environment Corporation, p10)

3.2 Server Software

3.2.1 The World Wide Web Server

The WWW server used in this thesis was the HTTP Server Version 0.96 freeware developed by the European Microsoft Windows NT Academic Center (EMWAC). This software was installed on a Windows NT Server (Version 3.5) machine which runs on a 60 Megahertz Intel Pentium Processor. The WWW Server uses port 80 which is the standard World Wide Web port number.

3.2.1.1 Simple Requests

A WWW server listens on port 80 for requests. Once a request is received then the system analyzes it. A sample request by a Web client could be the following:

```
http://myserver.com/path/filename.htm
```

The Web browser then does a lookup on the **myserver.com** address to figure out where to send it. After connecting to the machine, the **HTTP** in the request tells it to send the message to port 80 of the **myserver.com** machine. Then the browser sends the path with the filename (/path/filename.htm). Along with the message it has to send a method, which in most cases is a GET. This is explained in further detail in the next section.

3.2.1.2 HyperText Transfer Protocol

HyperText Transfer Protocol is a higher level protocol that the WWW uses to communicate between the Web browsers and the WWW Servers. Because HTTP works over the Internet, it assumes a TCP/IP connection. (TCP/IP is the mainly used over the Internet) This does not, however, mean that it cannot run over other protocols over the Internet. Here is a definition of HTTP according to the W3 Consortium.

"The HyperText Transfer Protocol (HTTP) has been in use by the WWW global information initiative since 1990. HTTP is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hyper media information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. HTTP is also used for communication between user agents and various gateways, allowing hypermedia access to existing Internet protocols like Simple Mail Transfer Protocol (SMTP), Network News Transfer Protocol (NNTP), FTP, Gopher, and WAIS ." (World Wide Web Consortium, http://www.w3.org/pub/WWW/Protocols/Overview.html)

HTTP can be thought of as a set of methods or commands that the client will use to communicate to the WWW server. Table 3-1 below list some of the methods according to HTTP version 1.0.

Method	Description
GET	Retrieve the specified document
HEAD	Retrieve only the HTTP headers for the specified document
SHOW METHOD	Return a description for a given method
PUT	Store the specified data in the specified URL
DELETE	Request that the server delete the item in the specified URL
POST	Create a new object linked to the specified object
LINK	Link an existing object to the specified object.
UNLINK	Remove link information from an object

Table 3-1: A description of the methods in HTTP version 1.0

Of all of the methods described in Table 3-1, the **GET** method is most frequently used. When using a Web Browser like Netscape, clicking on a HyperText will trigger an **GET** with the URL associated to the HyperText. In other words in most cases users are downloading HTML documents which are obtained through the **GET** method. Likewise both of the prototypes implemented only used the **GET** method.

3.2.2 Database Server

A relational database management system (RDBMS) consists of data organized into structured tables (rows and columns). The columns of the table are called *fields* or *attributes* much like Table 3-1. Table 3-1 has 2 fields or attributes. There are many software vendors who make RDBMS's. Some of the most common RDBMS are Oracle, Sybase, DB2 (IBM) and Informix. These RDBMS's are very large programs that usually reside on powerful UNIX workstation servers NT or OS/2^{TM} servers. The reason why these RDBMS's are not on the standard desktop is because of the lack of processing power or because of the operating system. Within a large distributed computing environment, a database server will be taking requests from many users as well as doing other tasks or processes. Therefore, it needs to be able to perform multiple tasks at the same time instead of queuing up the tasks which would make it unpleasant for the clients requesting data and other services. Each of the operating systems are multi-tasking, multi-threading operating systems and in many cases these servers even have multiple processors.

All of the RDBMS's mentioned above were designed to optimize the multi-threading ability of the operating system.

3.2.2.1 Structured Query Language

All of the RDBMS's mentioned above work in a language called **Structured Query Language**, more commonly referred to as **SQL**. SQL was developed by IBM in the 1960's and is now the industry standard on RDMBS's. With SQL a user can create, update, modify and delete tables. As well as administering the data, the user can query the database with a given criteria. For example, if the user wanted to find out all of the HTTP methods that began with the letter 'p' then the user would type:

Select method from Table_3-1

 where method like 'p%'

This is an elementary query for an elementary table. As the table become more and more complex, so does the query. Although SQL is a very powerful language for accessing and administering a database, it is not the easiest language to use, especially for non-technical users. To help the users many software vendors like Oracle and Microsoft have come out with their database browsers also know as Query By Example software. This allows the user to go through a database from a PC and point and click with their mouse to select and view tables. Although this is very powerful, in many cases, it is too general or generic. What the end users need are very customized front-ends so that they can access just the data that they are interested in.

To meet the demand for customized front-end (like the marketing division of a company), there is presentation software like Visual Basic, PowerBuilder, and SQL Gupta just to name a few. These allow easy access to viewing data as well as updating data through an easy to use graphic user interface (GUI). Each of these presentation software (or toolkits) can communicate with the back-end RDBMS's through their own Dynamically Linked Libraries (DLL's) or through a standard defined and developed by Microsoft called Open Database Connectivity (ODBC).

3.2.2.2 Open Database Connectivity

ODBC is a standard defined by Microsoft which provides a standard mechanism and language for applications to communicate to SQL relational databases. Although SQL is it's own language, there are different flavors which differs from vendor to vendor.

ODBC allows applications to talk to a variety of back-end RDBMS's with one single dialect. It also allows any application that is ODBC compliant to communicate with any back-end database. One of the disadvantages in using ODBC is that the application is limited to the SQL commands that ODBC supports. In most cases, this usually is not a problem, but in some cases there might be special (SQL) commands that are specific to the RDBMS (each RDBMS has its own flavor of SQL) that user might wish to execute which would not be possible through ODBC.

The second disadvantage to using ODBC is performance. An application will run faster if it communicates directly with the database versus talking to ODBC and have ODBC communicating to the database (see Figure 3-5). In applications where transaction speed is the bottleneck, connecting to a database via ODBC might not be the optimal choice, but more often then not (in large companies with LANs and WANs) the network throughput is the bottleneck. (Belford, p12)

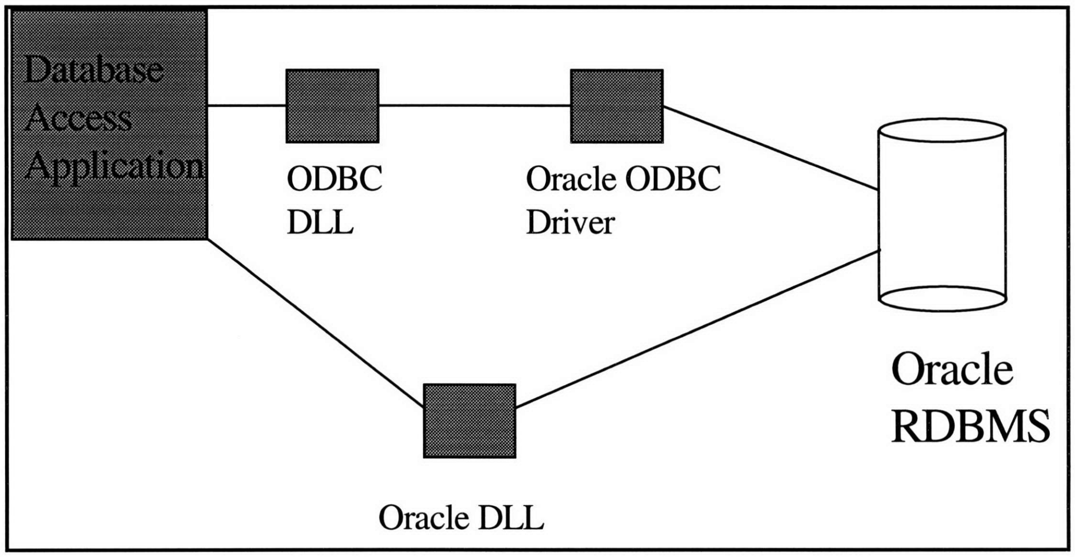


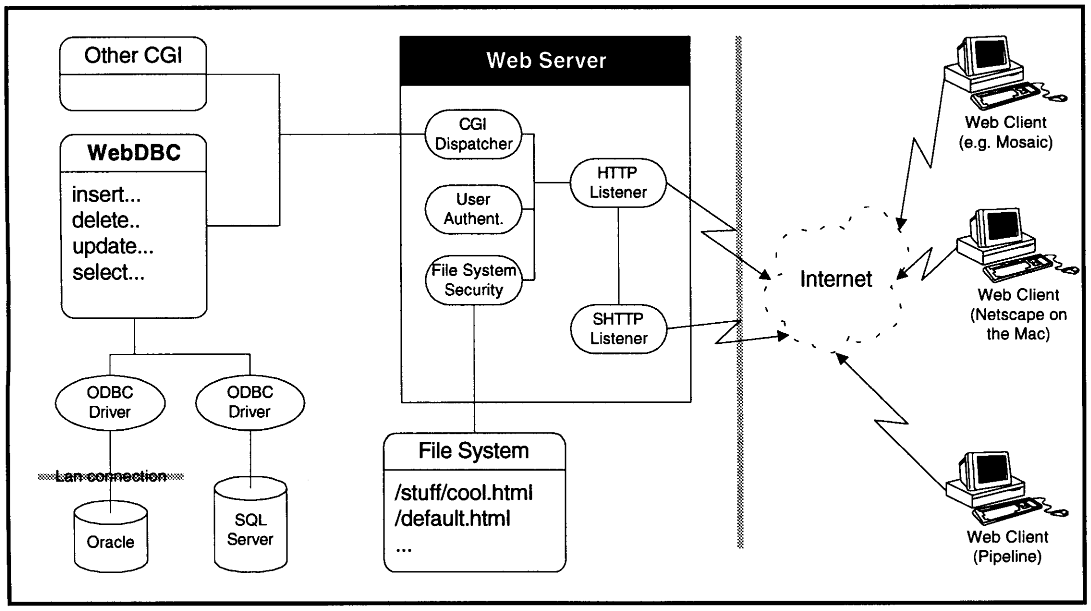
Figure 3-5: Database Access: ODBC vs. Direct DLL

3.2.3 WebDBC

WebDBC is the first SQL database connectivity product for the World Wide Web. It is a CGI application, but unlike other database connectivity tools, it does not require intense programming in C or Perl. Because there was such a high demand for services that wanted to be offered on the Internet that involved connectivity to databases like point-of-sale systems, user-registration, keyword searches, and catalog shopping (just to name a few), Nomad created WebDBC.

WebDBC is a command line application that helps Web Servers carry out SQL encoded requests. It takes a request (from the Web Server) and converts it into a SQL query, submits the query to the database, formats the results of the query, creates a Web page dynamically and then returns the page to the Web Server so the Web server can return the page to the client (usually a Web browser).

When a Web server receives a request from a client it tries to find the document (if the request was for a document) and then return it to the client. It can also, however, run an application on the server as opposed to retrieving a document. The application is invoked through the URL and the rest of the URL is usually the arguments that are to be passed to the application being invoked. Then, the application will process the request and return a Web page to the user. For any Windows, Windows NT or UNIX machine to be invoked by a Web server, the application must be a command line application which WebDBC is (See Figure 3-6).



©Nomad Corporation 1995

Figure 3-6: WebDBC Architecture

As mentioned before, once WebDBC is invoked by the Web Server it examines the request (the URL passed from the client) more closely and converts it into an SQL statement. This statement (select or update) is then passed to the database that is specified in the request. The connection to the database is made through ODBC, the Open Database Connectivity API defined by Microsoft. This is good because ODBC can communicate with the majority of RDBMS's out there. The only problem is that sometimes these 32 bit drivers which are required for Windows NT are sometimes scarce and almost non-existent for UNIX. (There is only one company that makes ODBC drivers for UNIX). With the release of Windows 95, a full 32 bit operating system, 32 bit ODBC drivers will become more available as Windows 95 becomes the PC desktop standard operating system. (Belford, p5)

When the results are ready (from the database) WebDBC merges the data into a Web Page which has special formatting markers. This process is very similar to a mail merge process in common word processor programs. This output Web page with the special format markings can is very much like a template specifying where the data should be placed. WebDBC does not determine which output page to use. It is already determined from the request which comes from the client.

These output pages or files are called HTX files because they have a .HTX extension instead of a .HTM or .HTML extension. In addition to the special format markings, the .HTX file also has generic HTML markings. This is good because it allows the data to be incorporated into an elaborate Web page.

3.2.3.1 Querying the Database: The WebDBC URL

As mentioned before, the request from the client is submitted via a URL. Figure 3-7 shows a sample request.

```
http://persimmon.mit.edu/bin/webdbc.exe/bank/q_combined_trans/select/&/demo
/access/1/welcome1.htx?&d_AccountNumber=45678
```

Figure 3-7

Note: The URL is always one continuous line. There is never a carriage return.

Most Web users do not just type in random URL's to get connected although they do have to do it at least once. They usually start of at one place and start clicking their way around. In other words URL's are always predefined within the Web pages and that is how the database is queried from the Web browser. (The URL is hard coded into the Web page.) Therefore, the user is not required and should not type in the WebDBC URL (as in Figure 3-7). However, this does not mean that the query is always hard coded and the users cannot specify parameters. Because WebDBC supports forms, and can pass variables, queries can be created in a more dynamic fashion.

When the URL in Figure 3-7 is submitted, it is mapped by the Web server into the following request-specific environment variables.

Variable	**Value**	**Comments**
SCRIPT_NAME	bin/webdbc.exe	The webdbc.exe is in the bin directory of the Web root directory defined by the server
PATH_INFO	/bank/q_combined_trans/select/&/demo/access/1/welcome1.htx?	These are the arguments that are importance to webdbc, the CGI application
QUERY_STRING	&d_AccountNumber=45678	Everything after the question mark
FORM_CONTENTS	<empty>	This is not an environment variable. It is read in from stdin.

Table 3-1: Web Server Environment Variables

After the server invokes WebDBC, the PATH_INFO is further broken down (by WebDBC) into the following variables:

Variable	**Value**	**Comments**
DATABASE	bank	The database to connect to for this request. Due to ODBC, this is usual case sensitive
TABLE	q_combined_trans	The table to use in the SQL statement.
OPERATION	select	The type of operation to perform.
RESULT_FILE	/demo/access/1/welcome1.htx	The results of the query will be merged into this file.

Table 3-2: WebDBC Variables

The resulting SQL statement from Figure 3-7 would be

```
select COLUMNS from q_combined_trans where AccountNumber=45678
```

COLUMNS are the fields in the table q_combined_trans that are specified in the .HTX file. In other words, the return fields are not specified in the URL request, but rather in the return file (.HTX file).

3.3 Client Software

3.3.1 Winsock and TCP/IP

Creating TCP/IP applications under Windows 3.1 was never easy, but with Winsock it is much easier than before. In the beginning (of Windows 3.1) there were a number of companies that developed Windows 3.1 TCP/IP networking products, each with its own proprietary programming interface. Although these networking products worked well within a Windows based TCP/IP network they was not universal. Software developers had to program to several conflicting API's for their software products to capture the whole Windows TCP/IP market. Windows 3.1 lacked a standard interface for Windows TCP/IP program until the introduction of Winsock.

Winsock is based on the Berkeley Sockets, a known proven standard for TCP/IP (programming in UNIX). With this standard, developers who worked with a Winsock-compliant TCP/IP networking package can begin developing their own Winsock applications without buying a separate TCP/IP development library. Also, it became much easier to port code over to Windows from UNIX TCP/IP applications.

Despite this new development, Winsock did not really become popular all of a sudden. Because companies had invested so much of their time, energy and money into development of their TCP/IP applications, they were very reluctant to scrap their whole project and move over, their current TCP/IP applications did everything they needed to be done.

Then came the Internet. Internet TCP/IP applications were being developed by people around the world which made the lack of standard no longer acceptable. Therefore, Winsock became the standard for Internet application programming for Windows as well as Internet TCP/IP networking applications. Companies who wanted to connect there Windows computers to the net had no choice but to switch to Winsock or not be connected. (Coombs, Chapter 15)

3.3.1.1 Web Browsers

Any Web browser can be used to operate the Web Banking system. Web browsers that are designed for Windows (which was the client platform in this thesis) use the Winsock DLL to communicate with the sockets.

3.3.1.2 PowerBuilder 4.0

PowerBuilder 4.0 is a very powerful graphical PC-based client/server application development environment. PowerBuilder is specially tuned for developing Windows applications which access database sources of all types. PowerBuilder is an object oriented, event-driven programming environment. This is ideal for creating Windows applications since most Windows applications take advantage of a GUI.

Although PowerBuilder's most powerful feature is its ability to easily connect to RDBMS's either through their proprietary drivers or through ODBC and return the data in many different formats, it can be used to create other Windows applications because of its easy-to-use developing environment. A window with buttons can be created literally in seconds. What once took weeks or months to develop, can now be developed in hours or days. This is the power of PowerBuilder.

Although it seems like PowerBuilder is a very simple program it is very powerful. Almost every object in PowerBuilder can be customized and manipulated to do whatever the developer wishes. An example of an object is a window or a button which can be either clicked or double clicked or dragged. Clicking, dragging, and dropping are events. Scripts can be written for events such as those to execute functions for calculations, displaying a graph, or doing whatever the developer wants it to do.

PowerBuilder was designed in a very modular way so that libraries and other modules can be added on to it. This allows software vendors to develop add-ons to PowerBuilder or additional toolkits that will enable PowerBuilder developers access into certain gateways. One such application of this is the PowerSocket Library developed by Jason Coombs.

3.3.1.2.1 PowerSocket Library v2

The PowerSocket library is a dynamic library called *WINSOCK.PBD*. It enables a PowerBuilder developer to do any type of socket programming for any type of TCP/IP application. Perhaps the best way of illustrating how PowerSocket programming works is through a sample example. (Coombs, Chapter 16)

3.3.1.2.1.1 Example: Retrieving files from a Web Server

Retrieving files from a WWW Server is a fairly simple task. There are four basic things that need to happen to retrieve a file:

1) Create a new socket user object

2) Connect to the WWW server on port 80 (the default port for WWW servers).

3) Send a GET with the specific file and path.

4) Retrieve the file using asynchronous messaging

Here is the command for issuing the get after connecting to a specific computer on port 80.

//This code was originally written by Jason Coombs, slightly modified by Hoony Youn

```
ulong ulAddr            // the unsigned long version of the host's IP address
string sSend, sPath
int a

// the urlgetfilename() function strips the filename off of the URL
// NOTE: This is not an intelligent application. It assumes that the
// c:\temp\ directory exists and will convert the filename entered
// in the URL to DOS form: no more than 8 characters with a
// possible 3 character extension. A file with
// the extension .html is converted to .htm so that
// you can specify the URL for a UNIX-based HTML document
// without causing this program to choke. All other > 3 character
// extensions are suspect, however.
//
// sFileName is an instance variable of the window. Sloppy, but
// simple. The custom event triggered when data arrives on the socket
// needs to know this filename, and making it an instance variable
// was an easy solution.
//sFile = "c:\temp\" + urlgetfilename(URL)

sFile = "c:\temp\file.txt"

// sSocket is an instance variable of the window because it
// needs to continue to exist after this script is executed
sSocket = create socketstream         // create a new stream socket

// The URLGetAddress function accepts a URL and performs gethost
// DNS lookups to obtain the unsigned long address. No error
// checking is performed -- if the name or address resolution
// fails, you're out of luck.
```

```
ulAddr = URLGetAddress(URL)

// The URLGetPath function accepts a URL
sPath = URLGetPath(URL)

// perform a wsconnect() on the stream socket. Connect to port 80
// of the host specified in the URL. Port 80 is the standard http
// port.
a = sSocket.wsconnect(ulAddr,80)
if a = ws.SOCKET_ERROR then
        messagebox("Error","No connect")
end if

// Enable asynchronous messaging for this socket.
// the ws.wsor(ws.FD_READ,ws.FD_CLOSE) function returns the OR'ed
// value, which contains the appropriate bit-mask to tell the
// wsaasyncselect() function which events to monitor on the socket.
// handle(w_main) returns the window's handle.
// 1024 is the code for PowerBuilder's custom01 event: pbm_custom01
// this is the event that is executed every time either of the
// FD_READ or FD_CLOSE events occur on this socket.
sSocket.wsaasyncselect(handle(w_main),1024,ws.wsor(ws.FD_READ,ws.FD_CLOSE))

// this is the command sent to the HTTP server to retrieve the
// file identified in the URL. the ~r~n~r~n means carriage return (~r)
// linefeed (~n) carriage return (~r) linefeed (~n)
sSend = "get " + sPath + "~r~n~r~n"

// this sends the get command to the socket (which was connected to
// port 80 (the http server) of the specified host) -- again, there
// is no error checking or other intelligence here. If the connect
// failed for some reason, this is going to blow up.
sSocket.send(blob(sSend),len(sSend),0)

// check to see if the file that we want to write to exists.
// if it does, delete it just to be on the safe side. (not safe
// for the file in question, of course.)
if fileexists(sFile) = TRUE then
        filedelete(sFile)
end if

// open the output file
iFile = fileopen(sFile,StreamMode!,write!)

return
```

The following script is placed in the pbm_custom01 event of one of the windows (it doesn't matter which window, just as long as it is opened and the script references its handle). This script does the actual reception and storage of the data which in this case, is written into *file.txt*.

```
int iLen, iWhichEvent
blob blobtest                  // this is blob. Blob is your friend. Blob is a
                                                // distant relative of Microsoft's Bob(tm).
int fp
string text

// this is the birth of Blob. The baptism comes next.
// the blob() function is a PowerBuilder function. It might help to
// read about it.
blobtest = blob(space(1024))

// the WSAGetSelectEvent() function returns the event code that
// occurred on the socket, causing the Winsock DLL to trigger this
// event. In this application, there are only two possible values
// for iWhichEvent: ws.FD_READ or ws.FD_CLOSE since the http()
// function performed a WSAAsyncSelect() on the socket requesting
// notification for these two events.
iWhichEvent = ws.WSAGetSelectEvent(message.longparm)

// Check to see if this event was triggered due to a ws.FD_READ
// condition on the socket -- indicating that there is now data ready
// to receive.
if isvalid(sSocket) = TRUE and iWhichEvent = ws.FD_READ then

// iLen will contain the number of bytes received. This is important
// because your entire blob buffer might not be filled. You're
// only interested in the data copied by the recv() function, so
// capture its return value and only use the portion of the blob
// buffer that was actually populated with data by recv()
        iLen = sSocket.recv(blobtest,1024,0)

        // now write the data of interest to the file.
        // the blobmid() function is useful for doing this, as you can
        // specify the length of the data you want returned, thus
        // limiting what you write to the file to just the data
        // populated by the recv() function. (see note above)
        if iLen > 0 then
                filewrite(ifile,blobmid(blobtest,1,iLen))
        elseif iLen = ws.SOCKET_ERROR then  // oops, an error occurred.
                messagebox("Socket Error",string(ws.wsagetlasterror()))
        end if

end if // end of isvalid() if

// Now check to see if the ws.FD_CLOSE event triggered this script.
// If so, then the code within the previous if statement will have
// been skipped.

if iWhichEvent = ws.FD_CLOSE then
        fileclose(iFile)            // close the file since we're done writing
        sSocket.closesocket()       // close the socket
        destroy sSocket             // destroy the socket object
        if status = 1 then open(w_menu)
        if status = 2 then open(w_account_info)
        if status = 3 then open(w_checking_trans)
        if status = 4 then open(w_all_trans)
end if
```

3.3.2 Connecting to Internet

Connecting to the Internet can be done in a variety of ways. The MIT net is physically connected to the Internet. For people who are not in academia, connecting to the net could involve getting an ISDN connection from a provider, leasing a T1 line, or for many home users, getting a dial-up SLIP/PPP account.

3.3.2.1 Ethernet Connection To Internet

Once at a location where there is a direct connection to the Internet, getting connected is as simple as installing the Ethernet card in the computer and connecting the card with a cable to the existing network. Of course there are certain parameters that needs to be known to be totally ready, but that is true in any situation. These parameters are the IP address of the machine, the addresses of the Domain Name Servers. DNS allows one machine to connect to another machine on the Internet simply by calling its name instead of using its IP address.

3.3.2.2 SLIP/PPP Connection To Internet

Most people are not affiliated with an academic institution or a company which has a direct link to the Internet, so they have to get connected through other means which is usually through an Internet access company. For the home users, the most common way to connect is through a modem. This is called a SLIP/PPP connections. SLIP stands for Serial Line Interface Protocol and PPP stands for Point to Point Protocol. Either protocol can be used and there is no evident advantage for using one over the other. When users connect to the Internet access service, they literally join the network of that access service. They all will have there own IP address.

When any Winsock application is triggered, the computer dials out to the Internet access service and connects to the Internet. If a connection has already been established, then the computer simply executes the application.

4. Implementation

The scenario used for the prototypes developed for the thesis is a home banking application. Two prototypes that were developed for the home banking system. Both of them had the same type of features: the ability to check balances, recent account activity and graph data.

4.1 System Overview

As mentioned in the introduction, the home banking system, which was named Web Banking, has three major components: the front-end (Web browser or PowerBuilder GUI), the Web Server, and the RDBMS (see Figure 4-1).

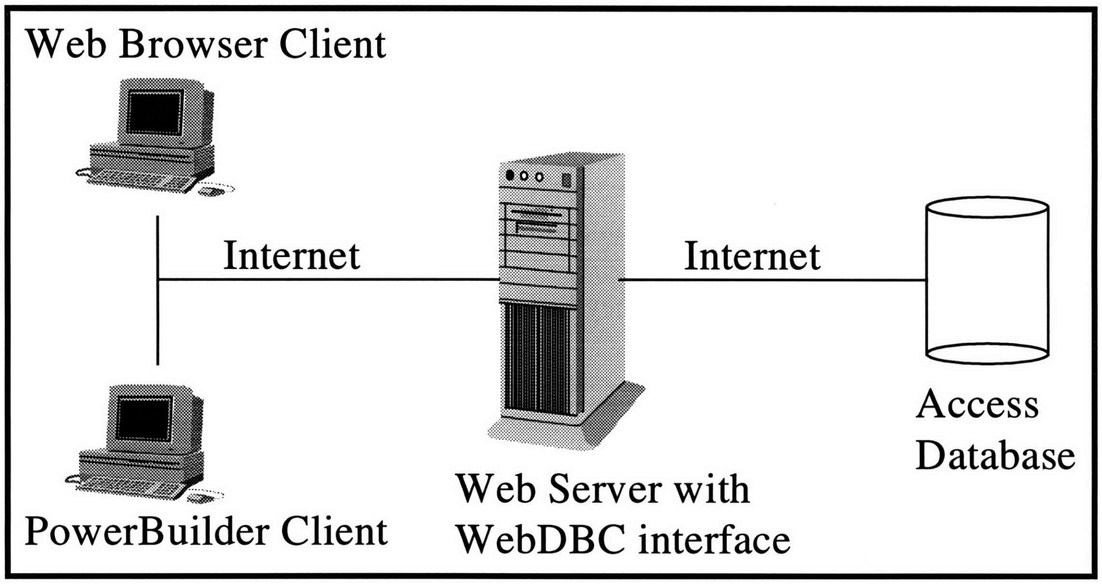


Figure 4-1: System Overview

4.2 Design

The goal of the Web Banking system was to offer a faster and more efficient way for a bank customer to do remote banking. This motivated the design of the Web banking system. The banking system has the ability to access multiple accounts from one session. In other words the user, after logging in, can see all of his/her accounts and continue their session from there, versus doing individual logins for each account.

After logging into the banking system the user comes to a main menu where he/she can choose which account to access. A Web page/window then will prompt the user for the particular information that the user wishes to access depending on whether the account is a checking, savings or money market account.

4.2.1 Back End Database

Because the Web Banking system is a data-intensive application, the database design is the heart of the whole Web Banking system. The RDBMS used for this system is Microsoft Access Version 2.0. Access has a table-like object called *queries* that are effectively virtual tables. There were a total number of 9 tables created and 5 *query* objects created. All of the *query* objects were built off of the tables, and only these objects were accessed during a user session with the Web Banking system. The schema and design of the tables and the *query* objects can be seen in the appendix.

4.2.2 Web Browser Client

The first system that was designed was the application using the Web browser as the front end. The whole banking application relies on successful transfer of data from the database back to the user. Therefore, the commands that are driving the request for data are SQL commands which are submitted from the Web browser side in the form of a URL. As mentioned in Chapter 3, WebDBC interfaces a Web browser with SQL databases.

Section 3.2.3.1 discussed how to query the database with the URL. It also mentioned that the data was returned back to the user through a .HTX file. A .HTX file is a specific file where column names can be specified so that the appropriate information can be returned to the user through that file. The .HTX file also has HTML in it so that the return page can create links to other pages. This is an important feature because it allows the links on that page to be created dynamically from information received from the database. The feature was vital to the design process of the Web banking system because one banking operation might involve several screens which are all created dynamically from information specific to the user.

A .HTX file can also be thought of as a template where data just gets inserted into a format that the user can interpret. These files resided on the Web server and they resided on three main directories. These directories were organized according to the account type (i.e. checking, savings, or money market). Figure 4-2 shows a picture of the directory structure.

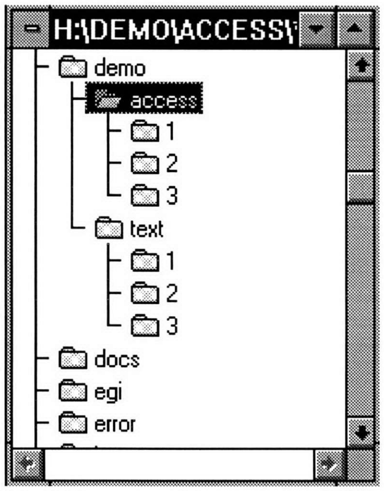


Figure 4-2: Directory Structure of Web Server

The *access* directory contains the main welcome screen that the user sees when first logging in. This screen then connects the user to yet another screen according to which account the user has chosen. The links to the bank accounts on the main welcome screen are created dynamically based on information from the database and connects the user to the appropriate .HTX file from directory related associated with the type of account. Table 4-1 shows the .HTX files used for the Web Banking system.

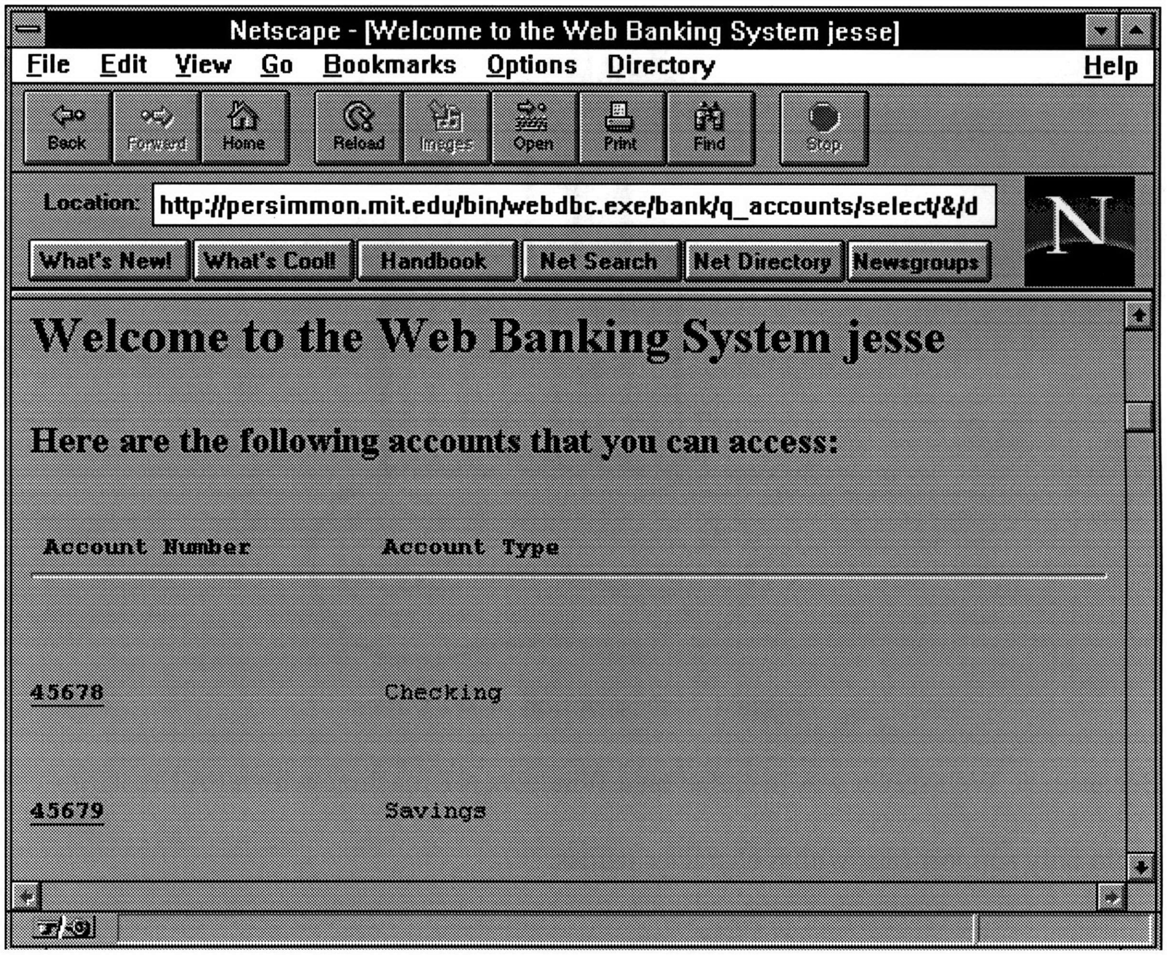
Filename	Directory	Description
welcome.htx	\demo\access	The main menu screen with all of the accounts related to the user.
welcome1.htx	\demo\access\1	The main screen for check checking accounts.
welcome2.htx	\demo\access\2	The main screen for savings accounts.
welcome3.htx	\demo\access\3	The main screen for money market accounts.
ch_d_res.htx	\demo\access\1	The results of recent cashed-checks inquiry.
trans.htx	\demo\access	All recent account activity transactions.
balances.htx	\demo\access	Balances of all the accounts the user has.

Table 4-1: Description of HTX files

Because the Web server was installed on a Windows NT machine with a FAT disk, it could only support files with 8 character names.

4.2.2.1 Web Browser Client: An Example

Here is an example of getting checking account information via the Web browser. After the user, **Jesse**, logs into the system, he sees the following screen which is the WELCOME.HTX file. All of the links on that page pertaining to the accounts were created dynamically. All of the accounts that are of the same type will use the same URL except for the last few characters which specify the account number.


Netscape - [Welcome to the Web Banking System jesse]
File Edit View Go Bookmarks Options Directory Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://persimmon.mit.edu/bin/webdbc.exe/bank/q_accounts/select/&/d
What's New! What's Cool! Handbook Net Search Net Directory Newsgroups

Welcome to the Web Banking System jesse

Here are the following accounts that you can access:

```
Account Number        Account Type

45678                 Checking

45679                 Savings
```

Figure 4-3: Main Menu on Web Browser

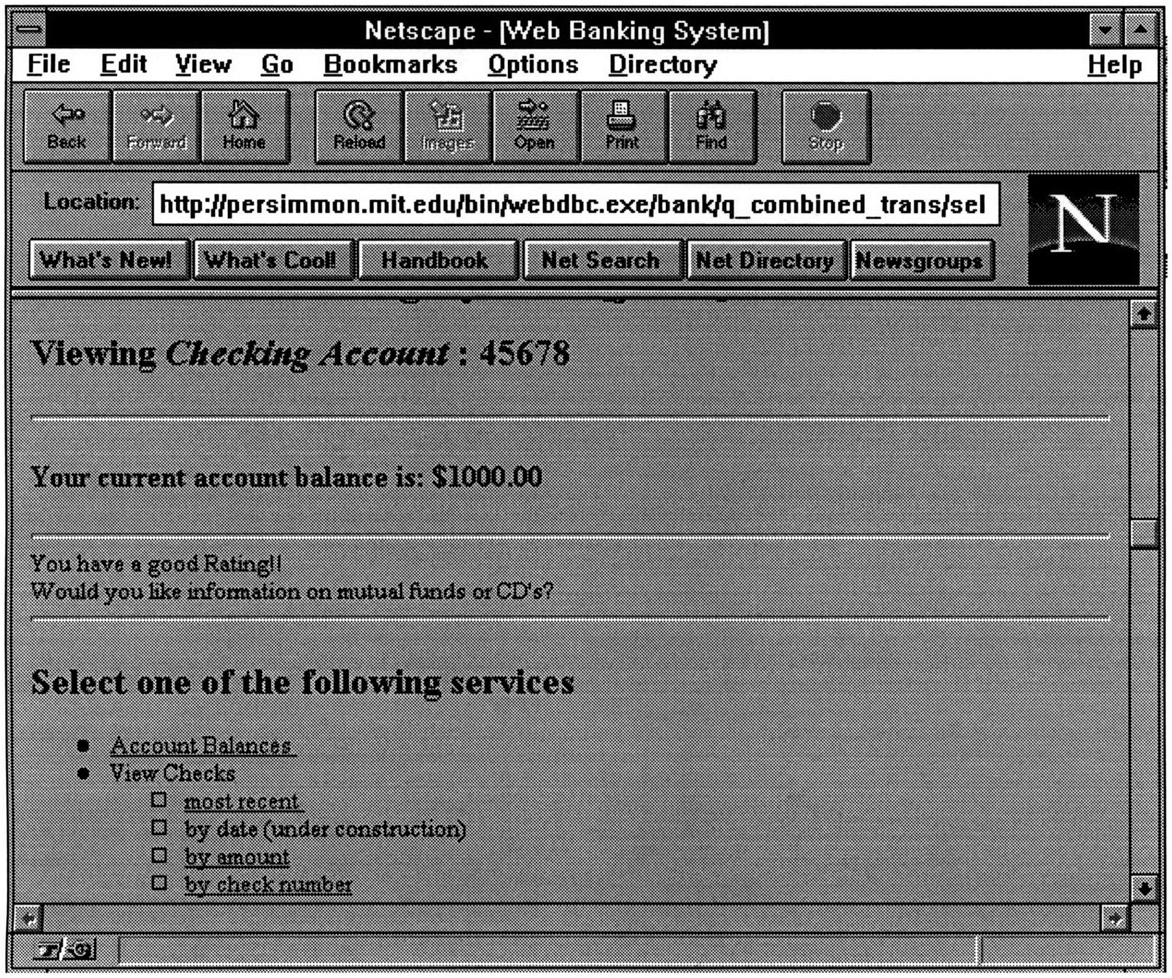
Clicking on the **45678** link will cause the following URL to be submitted.

http://persimmon.mit.edu/bin/webdbc.exe/bank/q_combined_trans/select/&/demo /access/**1**/welcome**1**.htx?&d_AccountNumber=45678

Since account, 45678 is a checking account which is also type 1 (savings is 2 and money market is 3), the number **1** gets inserted into the URL (above). If the user had chose account 45679 which is a savings account (type 2), then the URL submitted would have looked like this.

http://persimmon.mit.edu/bin/webdbc.exe/bank/q_combined_trans/select/&/demo /access/**2**/welcome**2**.htx?&d_AccountNumber=45679

Here is the result of clicking on **45678**:


Netscape - [Web Banking System]
File Edit View Go Bookmarks Options Directory Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://persimmon.mit.edu/bin/webdbc.exe/bank/q_combined_trans/sel
What's New! What's Cool! Handbook Net Search Net Directory Newsgroups

Viewing *Checking Account* : 45678

Your current account balance is: $1000.00

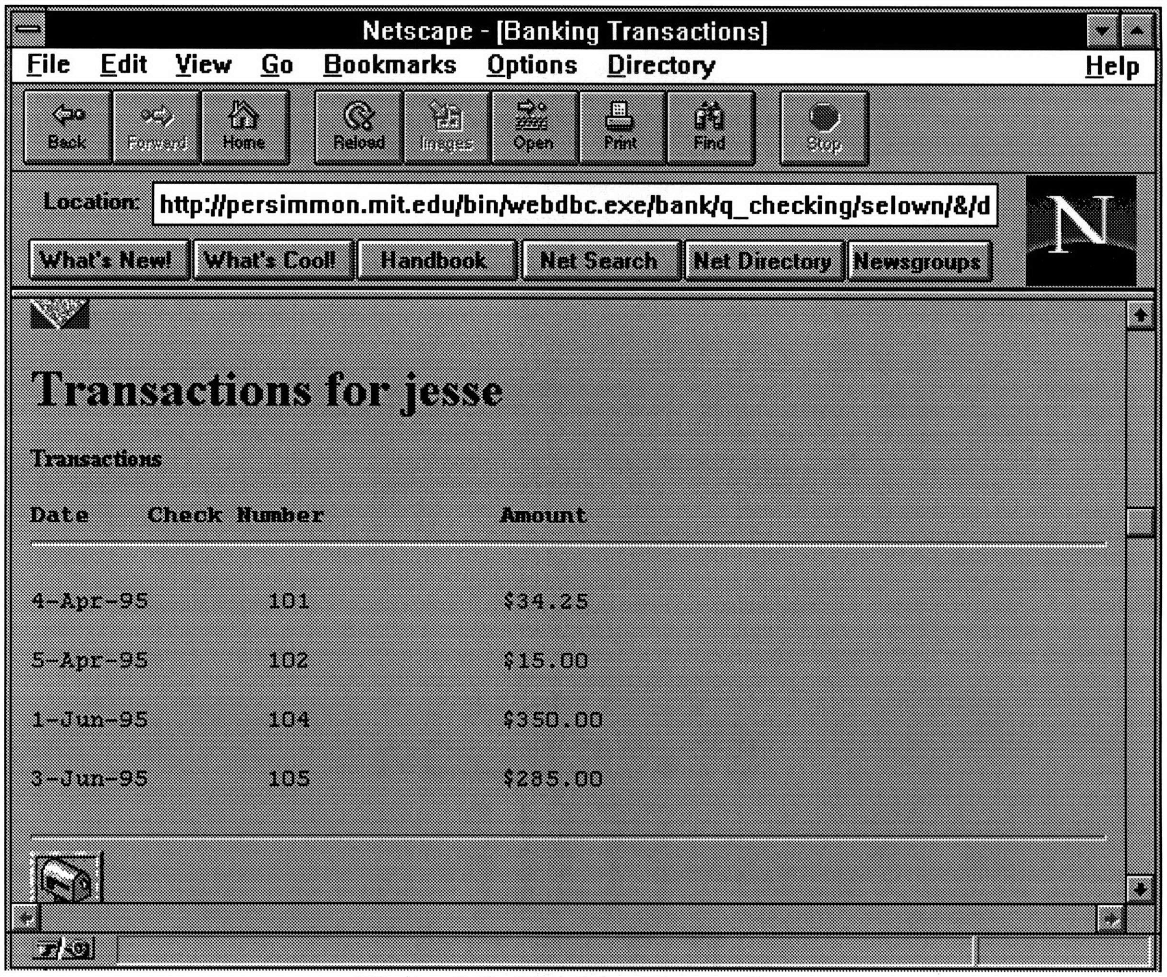
You have a good Rating!!
Would you like information on mutual funds or CD's?

Select one of the following services

- Account Balances
- View Checks
  - most recent
  - by date (under construction)
  - by amount
  - by check number

Figure 4-4: Account Information on **45678**

After entering 5 in the text box, here is what the Web Server will return.


Netscape - [Banking Transactions]
File Edit View Go Bookmarks Options Directory Help
Back Forward Home Reload Images Open Print Find Stop
What's New! What's Cool! Handbook Net Search Net Directory Newsgroups

Location: http://persimmon.mit.edu/bin/webdbc.exe/bank/q_checking/selown/&/d

Transactions for jesse

Transactions

```
Date      Check Number         Amount
4-Apr-95       101             $34.25
5-Apr-95       102             $15.00
1-Jun-95       104             $350.00
3-Jun-95       105             $285.00
```

Figure 4-5: Checking Transactions

4.2.3 PowerBuilder Client

The second part of this thesis was to create a customized front end that could communicate with the Web server. The design and the functionality of the PowerBuilder GUI is parallel to that of the Web browser system. The two systems are identical except for the front end and for the fact that PowerBuilder has the ability to do local processing. Like a Web browser, the PowerBuilder GUI submits a URL to the Web server asking it for specific data. Like with the Web browser client, the Web server returns the information back to the PowerBuilder GUI in a .HTX file except this file has no HTML formatting. Once the file has been received it is then brought to a data window object in PowerBuilder for display.

Like the Web browser front end, the PowerBuilder front end's windows are created dynamically depending on the account information. Because PowerBuilder is a true programming environment, applications are much easier to develop. Local, instance and global variables exist as well as the ability to reference other objects within the application.

PowerBuilder is an object oriented programming environment. Every object has attributes that can be set visually (while developing the application) or dynamically (while the application is running). Two of the higher level objects in PowerBuilder are the *window* object and the *data window* object. The *window* object is just a plain window like those which make up the Windows 3.1 environment. The *data window* object is usually placed within a *window* object. The *data window* object's purpose is to display the data that has been returned to the application from a database. PowerBuilder comes with drivers will connect directly to common RDBMS's like Oracle, Sybase, and Informix.

Table 4-2 shows a list of the all of the window objects and table 4-3 shows a list of the data window objects created for this system.

Window Object	Purpose
w_main	The login screen.
w_menu	The main menu screen with links to all accounts
w_account_info	Displays information on a particular account (i.e. checking, savings, money market.)
w_check_trans	Displays recent cashed-checks
w_all_trans	Displays recent activity in all accounts

Table 4-2: Window Objects for PowerBuilder GUI

Data Window Object	Related Window Object
d_menu	w_menu
d_account_info	w_account_info
d_checking_trans	w_check_trans
d_all_trans	w_all_trans

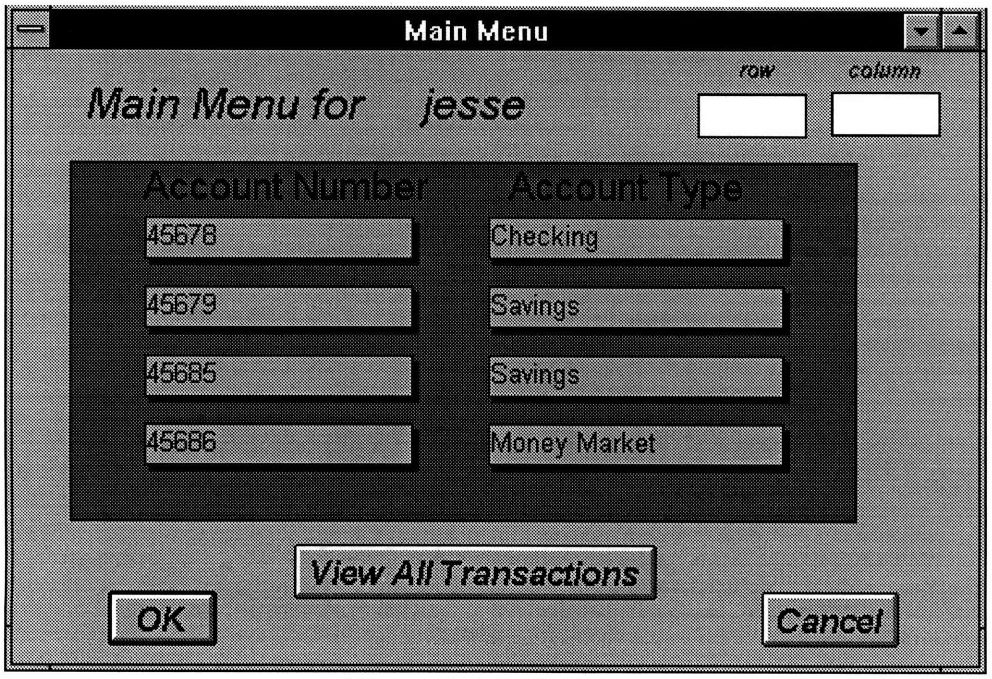
Table 4-3: Data Window Objects for PowerBuilder GUI

In the case of this system, the *data window* object was not associated with any RDBMS's but with a data file which is received from the Web Server. PowerBuilder has the ability to import a data files (in a tab delimited format) into a *data window* object by using the ImportFile() function.

There was socket programming involved which dealt with communicating with a Web server and retrieving specific files. The PowerSocket Library which was introduced in section 3.3.1.2.1 was used in this system. Retrieving files from a Web server was also discussed in that section.

4.2.3.1 PowerBuilder Client: An Example

Here is the PowerBuilder version to the example discussed in section 4.2.2.1. After logging into the system the user, Jesse, will see this.



Main Menu for jesse

Account Number	Account Type
45678	Checking
45679	Savings
45685	Savings
45686	Money Market

Figure 4-6 Main Menu

Accessing checking account 45678 only takes a double click on the box containing the account 45678 which will then open another window returning information specific to that account. Like the Web browser front end, a URL is submitted to the Web server. Like in the Web browser example, the URL will look like

http://persimmon.mit.edu/bin/webdbc.exe/bank/q_combined_trans/select/&/demo/**text**/1/welcome1.htx?&d_AccountNumber=45678

The only difference between the URL's in the two front ends is that the .HTX files for PowerBuilder reside within the TEXT subdirectory. Because PowerBuilder does not need

HTML in the .HTX file a duplicate set of .HTX files were created for the PowerBuilder version and were placed under the TEXT subdirectory.

Here is what the result looks like.

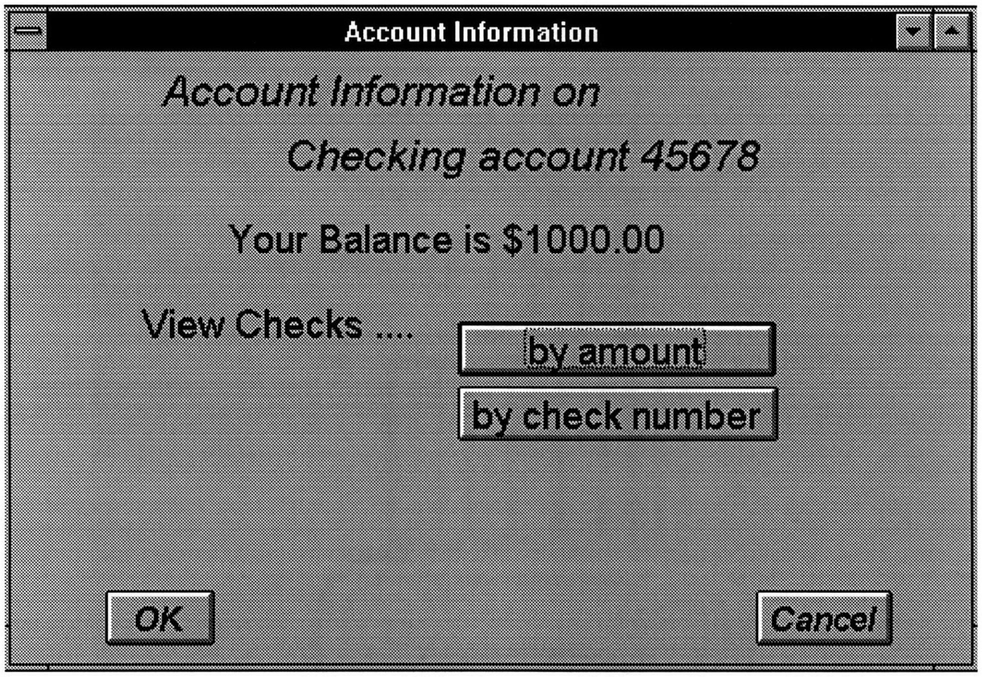
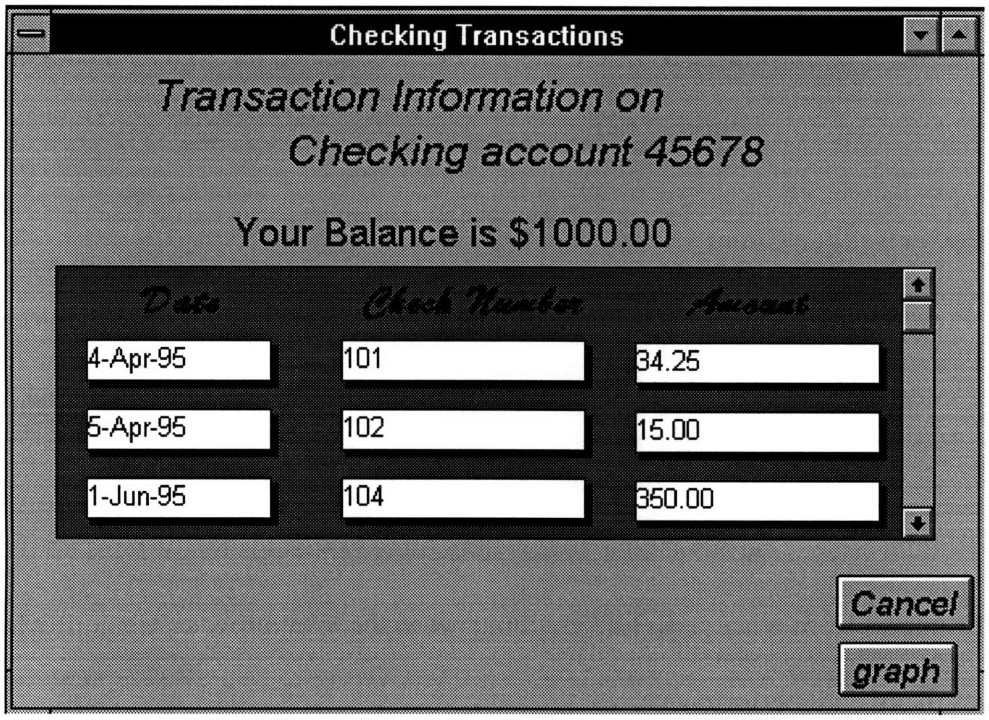


Figure 4-7: Account Information on 45678

To view checks over $5.00 that were cashed recently, click on the **by amount** button and enter 5 into the text box and click **OK**.

The results are as follows:



Transaction Information on
Checking account 45678

Your Balance is $1000.00

Date	Check Number	Amount
4-Apr-95	101	34.25
5-Apr-95	102	15.00
1-Jun-95	104	350.00

Figure 4-8: Information on Checking Transactions

4.3 Comparison: Web Browser vs. PowerBuilder GUI

4.3.1 Web Browser

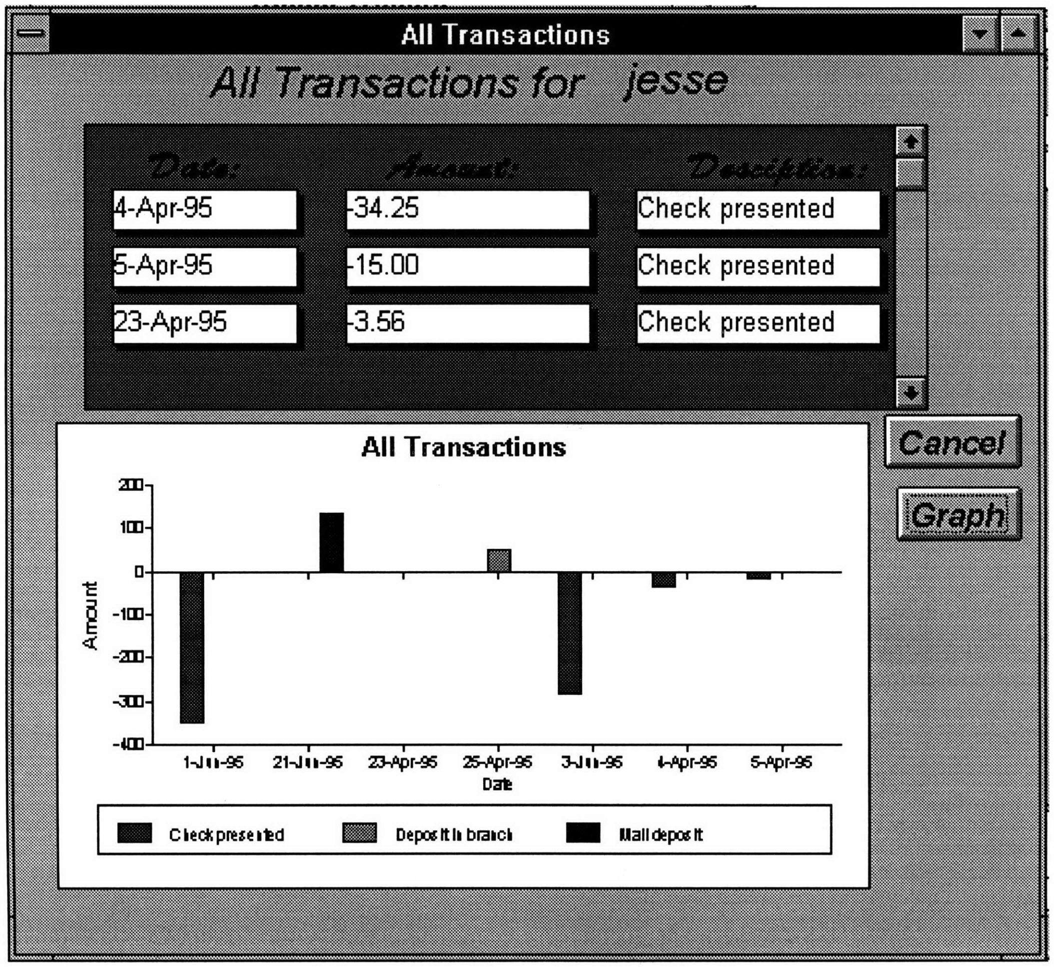
The Web Browser is very easy to use, whether or not the user is an experienced Web user. Everything function on the system is intuitive. It does, however, have its limitations. All Web browsers do is display text and images. This is very much like a dumb terminal interface. This setup follows the two-tiered model explained in chapter three.

The processing involved in making the graph cannot be done on the Web browser side. It is done on the server side, or it could be done on another server, but not on the client side. Another drawback to this graphing approach is that the output graph format must be converted to a GIF file before returning it to the client. This process can be very long when dealing with lots of numbers. Also, returning a large GIF file to the client can be very slow especially for SLIP/PPP users who are connected through an analog modem link.

4.3.2 PowerBuilder Client

PowerBuilder is the answer to all of the problems and limitations to the Web browser interface. Because the only requests that the PowerBuilder Interface makes to the Web Server are text files, the remaining processing is brought to the client (PowerBuilder) side. One advantage to this three-tiered model is that it frees up the server resources from other tasks like graphing (as in the Web browser version) which improves the overall performance.

Not only is the performance better with this model, but as mentioned before, it allows the application to utilize the superior graphing functionality of PowerBuilder. (See Figure 4-9.)



All Transactions for jesse

Date:	Amount:	Description:
4-Apr-95	-34.25	Check presented
5-Apr-95	-15.00	Check presented
23-Apr-95	-3.56	Check presented

Figure 4-9: PowerBuilder Graph

Since the database used in the Web browser version is the same database that is used in the PowerBuilder version, the data that is returned to the PowerBuilder client is the same data. The only major difference is that it is an unformatted version (non html) since the data is not being viewed through a Web browser. It is, however, just a plain text file (tab delimited). Therefore, the differences between the versions lie in the .HTX files that are returned to each of the clients.

Here is an example of a .HTX file for Netscape:

```
<HTML>
<HEAD>
<TITLE>Welcome to the Web Banking System <%r_username></TITLE>
</HEAD>
<BODY>
<img src="/images/arcade.gif">

<h1>  Welcome to the Web Banking System <%r_username> </h1> <br>
<h2>  Here are the following accounts that you can access: </h2><br>
<pre>
<b> Account Number                    Account Type</b><hr></pre>

<%detail>
<pre><b>
<A
HREF="/bin/webdbc.exe/bank/q_combined_trans/select/&/demo/access/<%r_AccountType>/welcome<%r
_AccountType>.htx?&d_AccountNumber=<%r_AccountNumber>">
<%r_AccountNumber></a></b>                              <%r_AccountName><p></pre>
<%/detail>

<hr>
<h2>  Other Options: </h2>
<ul>
<li> <A HREF="/bin/webdbc.exe/bank/q_balances/select/&/demo/access/balances.htx?"> View All
Account Balances </a>
<li> <A HREF="/bin/webdbc.exe/bank/q_accounts/select/&/demo/access/transfer.htx?"> </a>Transfer
money
</ul>
<h5>
<hr>
<IMG SRC="/images/icnmail2.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
Cambridge, Massachusetts.  All Rights Reserved.</h5>

</BODY>
</HTML>
```

The result can be seen in Figure 4-3.

Here is the corresponding .HTX file for the PowerBuilder client

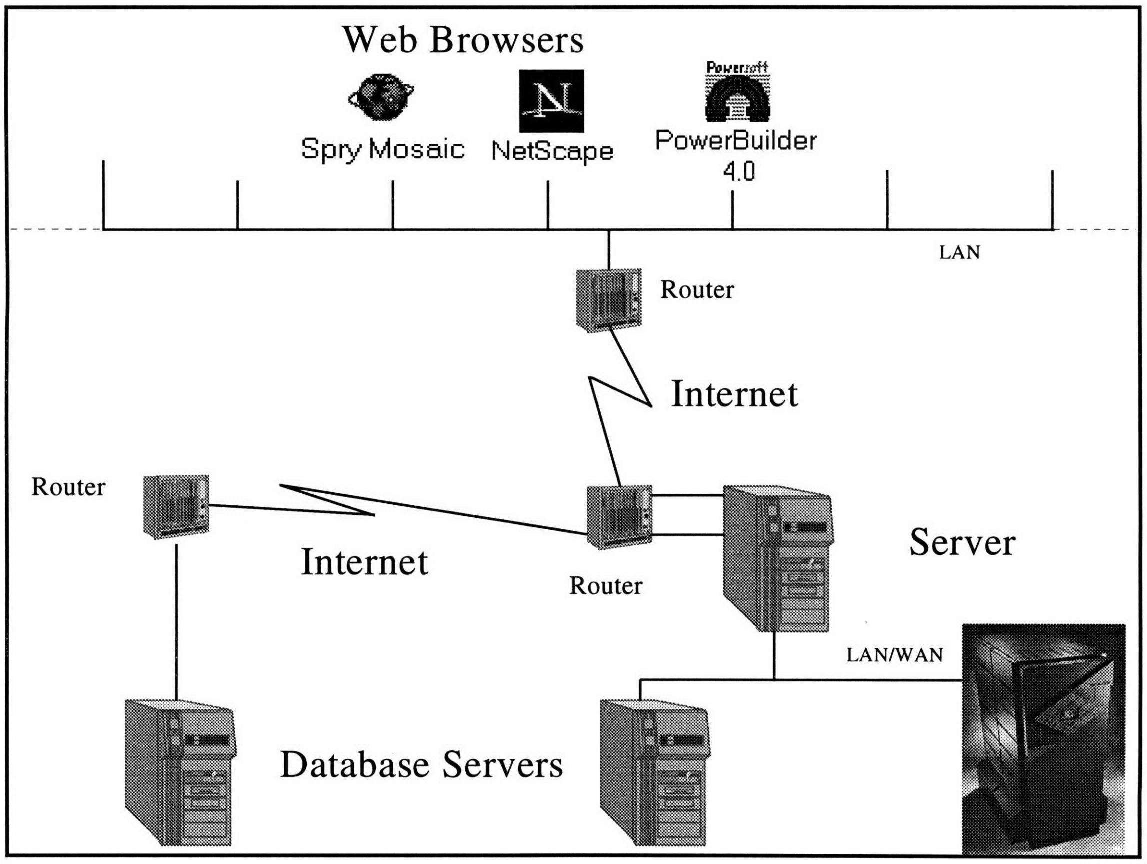
```
<%r_AccountNumber>    <%r_Balance>    <%r_AccountName>
```

The results can be seen in Figure 4-6.

5. Further Directions and Conclusions

What used to take one hour for a PC to do 10 years ago can be done in merely seconds now. Today, even the top of the line PC's can keep up with workstations that are as much as 10 times their price! Because PC's today are so powerful, many companies who would have otherwise bought a workstation are now buying PC's.

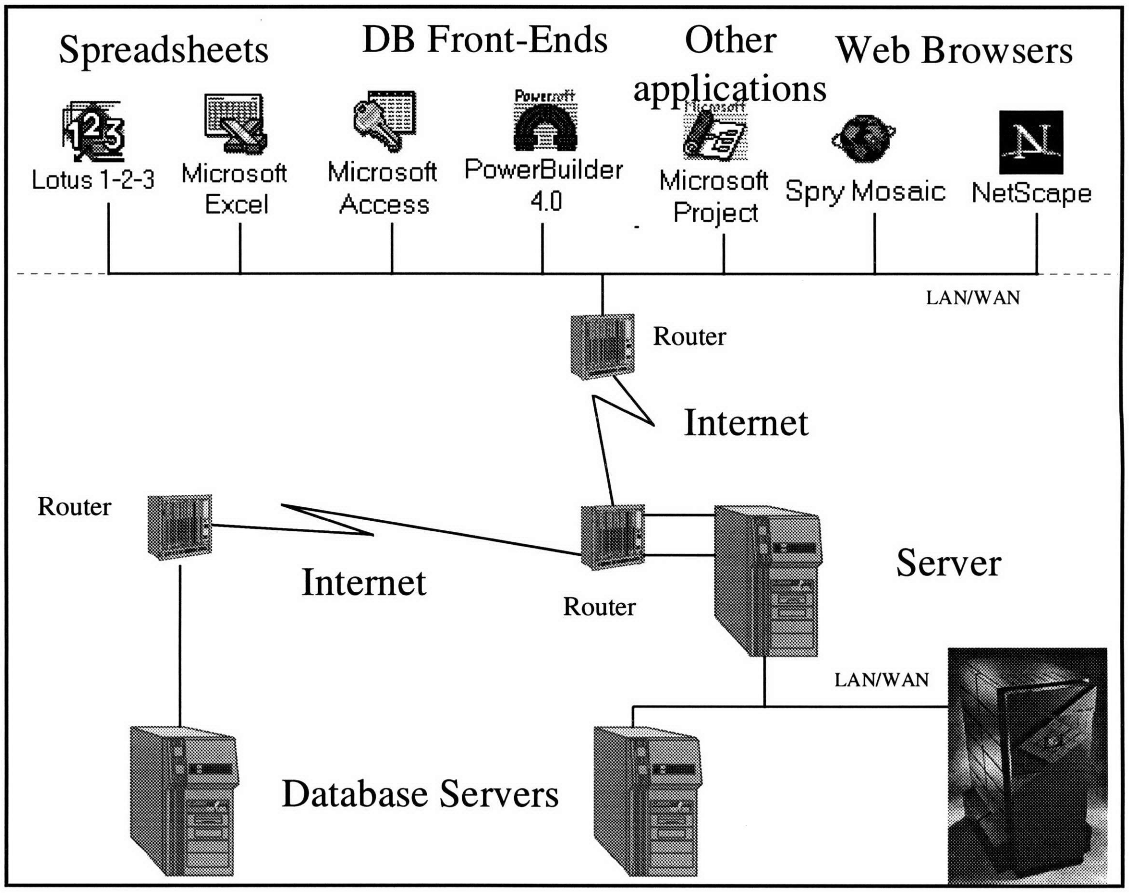
This thesis implemented two applications: one that gives access to a back-end database through a WWW server from a Web browser, and one that accesses the same back-end database through a WWW server from a PowerBuilder GUI.



©IW3

Figure 5-1: Thesis Architecture

The PowerBuilder GUI allows users to exploit the features of PowerBuilder as well as utilize the computing power of their desktop PC. This **innovative** concept which was implemented in this thesis can be taken one step further by using other industry standard applications to access information from remote sources via a public network (the Internet). (See Figure 5-2.)



©IW3

Figure 5-2: Futher Additions to Thesis Architecture

For example, Microsoft Excel or Lotus Notes could be a Web client that downloads information from Web servers, imports them into customized worksheets ready for the user to do online analytical processing (OLAP). These desktop applications are much more than meets the eye. Microsoft's Office Suite is not only a complete set of applications in itself, but allows users to add on to it with its built-in programming interface.

In summary, applications which require information to be processed and manipulated will work best with a front-end that allows processing to be done on the client side (i.e. PowerBuilder and Visual Basic). Today, the Web browser cannot do this. It can only display information. Therefore, applications which only provide information to the user which does not have to be processed (like plain text files) will work well with a Web browser.

The Internet is the Information Superhighway for today and it is here to stay. It is often the fastest and most efficient way to send and receive information. Although most of the hype of the Internet has been due to the World Wide Web, there are many other Internet applications that can be made for the marketplace that do not use the World Wide Web. For example, PowerBuilder does not have to communicate to the WWW server to retrieve the data from the back-end database. It can talk directly to the database or talk to another middle layer that is not a WWW server. This in fact would be a more secure connection than through an asynchronous connection that the Web uses and would yield better performance. So what is the best way to offer a service through the Internet? The usual answer: "It depends." It depends on the customer base and what sort of information is offered. No matter which method is chosen, several things to remember are utilize all of your resources (hardware and software) and be innovative. Just because things are done in one manner does not mean that doing it in a different way would be worse. It is these type of innovations that cause this field to develop as rapidly as it does.

6. Appendix

6.1 Database Table Schema

Account_Definition

Fields	DataType
ID	Number (Long)
AccountType	Number (Integer)
AccountName	Text

Account_Num_Type

Fields	DataType
AccountNumber	Text
AccountType	Number (Integer)
username	Text

Accounts

Fields	DataType
ID	Number (Long)
CustomerID	Number (Long)
AccountNumber	Text
DateEstablished	Date/Time
Closed	Yes/No
DateClosed	Date/Time

Balances

Fields	DataType
AccountNumber	Text
Balance	Number (Double)
Rating	Number (Integer)

Checking

Fields	DataType
TransactionID	Number (Double)
CheckNumber	Number (Integer)

Customers

Fields	DataType
ID	Number (Long)
CustomerID	Number (Long)
SSN	Text
Password	Text
Prefix	Text
FirstName	Text
MiddleName	Text
LastName	Text
Suffix	Text
Nickname	Text
Title	Text
OrganizationName	Text
Address	Text
City	Text
State	Text
Region	Text
PostalCode	Text
Country	Text
AreaCodeHome	Text
HomePhone	Text
AreaCodeWork	Text
WorkPhone	Text
MobilePhone	Text
AreaCodeFax	Text
FaxNumber	Text
AreaCodeAlternative	Text
AlternativePhone	Text
EmailAddress	Text
Birthdate	Date/Time
Nationality	Text
DateUpdated	Date/Time
DateFirstEntered	Date/Time
Photograph	OLE Object
Note	Memo

Transaction_Definition

Fields	DataType
ID	Number (Long)
TransactionType	Number (Integer)
Definition	Text
Symbol	Text

Transactions

Fields	DataType
TransactionID	Number (Long)
AccountNumber	Text
TransactionType	Number (Integer)
Amount	Currency
Date	Date/Time

Users

Fields	DataType
username	Text
password	Text
active	Number (Double)
why	Text
curvisit	Date/Time
lastvisit	Date/Time
fullname	Text

6.2 Database Queries

As mentioned before, a *query* in Microsoft Access is an object which can be viewed as a table. Here are the SQL statements involved for making the query objects used in this thesis.

```
/********************** q_accounts ***************************/
SELECT DISTINCTROW Account_Num_Type.AccountNumber, Account_Num_Type.AccountType,
Account_Num_Type.username, Account_Definition.AccountName
FROM Account_Definition INNER JOIN Account_Num_Type ON Account_Definition.AccountType =
Account_Num_Type.AccountType;


/********************** q_balances ******************************/
SELECT DISTINCTROW Account_Num_Type.AccountNumber, Account_Num_Type.username,
Balances.Balance
FROM Account_Num_Type INNER JOIN Balances ON Account_Num_Type.AccountNumber =
Balances.AccountNumber;
/******************** q_checking *****************************/
SELECT DISTINCTROW q_combined_trans.TransactionType, q_combined_trans.Amount,
q_combined_trans.username, q_combined_trans.TransactionID, Transactions.Date,
Checking.CheckNumber
FROM (Checking INNER JOIN q_combined_trans ON Checking.TransactionID =
q_combined_trans.TransactionID) INNER JOIN Transactions ON q_combined_trans.TransactionID =
Transactions.TransactionID
WHERE ((q_combined_trans.TransactionType=1))
ORDER BY q_combined_trans.username, Transactions.Date;
```

```
/******************** q_combined_check *******************************/

SELECT DISTINCTROW Transactions.TransactionID, Transactions.AccountNumber,
Transactions.TransactionType, Transactions.Amount, Account_Num_Type.username,
Transaction_Definition_1.Definition, Account_Num_Type.AccountType,
Account_Definition.AccountName, Transactions.Date, Transaction_Definition_1.Symbol,
Balances.Balance, Balances.Rating, Checking.CheckNumber
FROM Checking INNER JOIN (((Transaction_Definition AS Transaction_Definition_1 INNER JOIN
(Transactions INNER JOIN Account_Num_Type ON Transactions.AccountNumber =
Account_Num_Type.AccountNumber) ON Transaction_Definition_1.TransactionType =
Transactions.TransactionType) INNER JOIN Account_Definition ON Account_Num_Type.AccountType
= Account_Definition.AccountType) INNER JOIN Balances ON Transactions.AccountNumber =
Balances.AccountNumber) ON Checking.TransactionID = Transactions.TransactionID;

/*************************q_combined_trans ***************************/

SELECT DISTINCTROW Transactions.TransactionID, Transactions.AccountNumber,
Transactions.TransactionType, Transactions.Amount, Account_Num_Type.username,
Transaction_Definition_1.Definition, Account_Num_Type.AccountType,
Account_Definition.AccountName, Transactions.Date, Transaction_Definition_1.Symbol,
Balances.Balance, Balances.Rating
FROM ((Transaction_Definition AS Transaction_Definition_1 INNER JOIN (Transactions INNER JOIN
Account_Num_Type ON Transactions.AccountNumber = Account_Num_Type.AccountNumber) ON
Transaction_Definition_1.TransactionType = Transactions.TransactionType) INNER JOIN
Account_Definition ON Account_Num_Type.AccountType = Account_Definition.AccountType) INNER
JOIN Balances ON Transactions.AccountNumber = Balances.AccountNumber;
```

6.3 HTX Files

These files are specific to WebDBC Version 1.1.

For the location of the files refer to Table 4-1.

```
/***********************WELCOME.HTX**********************/

<HTML>
<HEAD>

<TITLE>Welcome to the Web Banking System <%r_username></TITLE>

</HEAD>
<BODY>
<img src="/images/arcade.gif">

<h1> Welcome to the Web Banking System <%r_username> </h1> <br>
<h2> Here are the following accounts that you can access: </h2><br>
<pre>
<b> Account Number                    Account Type</b><hr></pre>


<%detail>

<pre><b>
<A
HREF="/bin/webdbc.exe/bank/q_combined_trans/select/&/demo/access/<%r_AccountType>/welcome<%r
_AccountType>.htx?&d_AccountNumber=<%r_AccountNumber>">
<%r_AccountNumber></a></b>                              <%r_AccountName><p></pre>

<%/detail>
<hr>
<h2> Other Options: </h2>
<ul>
<li> <A HREF="/bin/webdbc.exe/bank/q_balances/select/&/demo/access/balances.htx?"> View All
Account Balances </a>
<li> <A HREF="/bin/webdbc.exe/bank/q_accounts/select/&/demo/access/transfer.htx?"> </a>Transfer
money
</ul>


<h5>
<hr>
<IMG SRC="/images/icnmail2.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
Cambridge, Massachusetts. All Rights Reserved.</h5>

</BODY>
```

```
</HTML>

/**********************WELCOME1.HTX***************************/

<HTML>
<HEAD>

<TITLE> Web Banking System</TITLE>

</HEAD>
<BODY>
<img src="/images/arcade.gif">

<h1> The Web Banking System [<%r_username>]</h1>

<h2> Viewing <i><%r_AccountName> Account </i>: <%r_AccountNumber> </h2>
<hr>
<h3> Your current account balance is: $<%r_Balance> </h3>
<hr>

<%if r_Rating="1">
You have a good Rating!! <br>
Would you like information on mutual funds or CD's?
<hr>
<%/if>


<h2> Select one of the following services </h2>
<ul>
<li> <A
HREF="/bin/webdbc.exe/bank/q_balances/select/&/demo/access/balances.htx?&d_AccountNumber=<%r_
AccountNumber>"> Account Balances </a>
<li> View Checks
        <ul>
                <li> <A
HREF="/bin/webdbc.exe/bank/q_combined_trans/select/&/demo/access/trans.htx?&d_AccountNumber=<
%r_AccountNumber>">most recent </a>
                <li> by date (under construction)
                <li> <A HREF = "/demo/access/1/ch_am.htm">by amount</a>
                <li> <A HREF = "/demo/access/1/ch_num.htm">by check number</a>
        </ul>
<li> Transfer Money...
        <ul>
                <li> <A
HREF="/bin/webdbc.exe/bank/q_accounts/select/&/demo/access/1/tr_from.htx?&d_AccountNumber.ne=<
%r_AccountNumber>&x_AccountNumber=<%r_AccountNumber>"><b>from account
[<%r_AccountNumber>]</b> to another account</a>
                <li> <A
HREF="/bin/webdbc.exe/bank/q_accounts/select/&/demo/access/1/tr_to.htx?&d_AccountNumber.ne=<%r
_AccountNumber>&x_AccountNumber=<%r_AccountNumber>">from another account <b>to account
[<%r_AccountNumber>]</b></a>
        </ul>
<li><A HREF="/bin/webdbc.exe/bank/q_combined_trans/selown/&/demo/access/trans.htx?&">View
Recent Transactions on all accounts
```

```
</a>
</ul>
<h5>
<hr>
<IMG SRC="/images/icnmail2.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
Cambridge, Massachusetts.  All Rights Reserved.</h5>

</BODY>
</HTML>

/*************************WELCOME2.HTX************************/

<HTML>
<HEAD>

<TITLE> Web Banking System</TITLE>

</HEAD>
<BODY>
<img src="/images/arcade.gif">

<h1>  The Web Banking System [<%r_username>]</h1>
<h2>  Viewing <i><%r_AccountName> Account </i>: <%r_AccountNumber> </h2>

<h2>  Select one of the following services </h2>
<ul>

<li><A
HREF="/bin/webdbc.exe/bank/q_combined_trans/selown/&/demo/access/trans.htx?&d_AccountNumber=
<%r_AccountNumber>">View Recent Transactions on <%r_AccountNumber>
</a>
</ul>
<h5>
<hr>
<IMG SRC="/images/icnmail2.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
Cambridge, Massachusetts.  All Rights Reserved.</h5>

</BODY>
</HTML>

/*************************WELCOME3.HTX**********************/

<HTML>
<HEAD>

<TITLE> Web Banking System</TITLE>

</HEAD>
```

```
<BODY>
<img src="/images/arcade.gif">

<h1> The Web Banking System [<%r_username>]</h1>
<h2> Viewing <i><%r_AccountName> Account</i>: <%r_AccountNumber> </h2>

<h2> Select one of the following services </h2>
<ul>

<li><A
HREF="/bin/webdbc.exe/bank/q_combined_trans/selown/&/demo/access/trans.htx?&d_AccountNumber=
<%r_AccountNumber>">View Recent Transactions on <%r_AccountNumber>
</a>
</ul>
<h5>
<hr>
<IMG SRC="/images/icnmail2.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
Cambridge, Massachusetts.  All Rights Reserved.</h5>

</BODY>
</HTML>

/***********************CH_D_RES.HTX*************************/

<HTML>
<HEAD>

<TITLE>Banking Transactions</TITLE>

</HEAD>
<BODY>
<img src="/images/arcade.gif"><h1>Transactions for <%r_username></h1>
<h2></h2>


<b>Transactions
<pre>
Date     Check Number             Amount</b><hr></pre>

<%detail>
<pre>
<%r_Date>      <%r_CheckNumber>                 $<%r_Amount><br></pre>
<%/detail>


<h5>
<hr>
<IMG SRC="/images/icnmail2.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
```

```
Cambridge, Massachusetts.  All Rights Reserved.</h5>

</BODY>
</HTML>

/************************TRANS.HTX****************************/

<HTML>
<HEAD>

<TITLE>Banking Transactions</TITLE>

</HEAD>
<BODY>
<img src="/images/arcade.gif"><h1>Transactions for <%r_username></h1>
<h2></h2>


<b>Transactions

<pre>Date              Amount                 Description</b><hr></pre>

<%detail>

<pre><%r_Date>          <%r_Symbol>$<%r_Amount>            <%r_definition></b></pre>
<%/detail>


<br>
Click <A HREF="http://context.mit.edu/cgi-bin/graph.cgi?username=<%r_username>"> here for a graph
of transactions </a> from all accounts.
<h5>
<hr>
<IMG SRC="/images/arcade.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
Cambridge, Massachusetts.  All Rights Reserved.</h5>

</BODY>
</HTML>

/**********************BALANCES.HTX**************************/

<HTML>
<HEAD>

<TITLE>Account Balances</TITLE>

</HEAD>
<BODY>
<img src="/images/arcade.gif"><h1>Account Balances for <%r_username></h1>
<h2></h2>
```

```
<b>Transactions
<pre>
Account Number          Balance</b><hr></pre>

<%detail>
<pre>
<%r_AccountNumber>                    $<%r_Balance></b></pre>
<%/detail>


<h5>
<hr>
<IMG SRC="/images/arcade.gif">
Please send questions and comments to <A HREF=mailto:spoon@mit.edu> spoon@mit.edu </A> <br>

&copy; 1994-1995 The Information Technology Group at the Sloan School of Management <br>
Cambridge, Massachusetts.  All Rights Reserved.</h5>

</BODY>
</HTML>
```

6.4 PowerBuilder Code

```
/****************** script for custom01 in w_main window *****************/

int iLen, iWhichEvent
blob blobtest                    // this is blob. Blob is your friend. Blob is a
                                                  // distant relative of Microsoft's Bob(tm).
int fp
string text

// this is the birth of Blob. The baptism comes next.
// the blob() function is a PowerBuilder function. It might help to
// read about it.
blobtest = blob(space(1024))

// the WSAGetSelectEvent() function returns the event code that
// occurred on the socket, causing the Winsock DLL to trigger this
// event. In this application, there are only two possible values
// for iWhichEvent: ws.FD_READ or ws.FD_CLOSE since the http()
// function performed a WSAAsyncSelect() on the socket requesting
// notification for these two events.
iWhichEvent = ws.WSAGetSelectEvent(message.longparm)

// make sure the socket object is still valid. For some reason
// this event was triggered again after the socket was destroyed,
// causing an error. The isvalid() check is here because I don't
// have time to figure out why WSAAsyncSelect() is being hyperactive
// when it comes to messaging for this socket. If you encounter the
// same thing in your program, add this isvalid() check as a work-
// around.
// Also check to see if this event was triggered due to a ws.FD_READ
// condition on the socket -- indicating that there is now data ready
// to receive.
if isvalid(sSocket) = TRUE and iWhichEvent = ws.FD_READ then

// iLen will contain the number of bytes received. This is important
// because your entire blob buffer might not be filled. You're
// only interested in the data copied by the recv() function, so
// capture its return value and only use the portion of the blob
// buffer that was actually populated with data by recv()
        iLen = sSocket.recv(blobtest,1024,0)

        // now write the data of interest to the file.
        // the blobmid() function is useful for doing this, as you can
        // specify the length of the data you want returned, thus
        // limiting what you write to the file to just the data
        // populated by the recv() function. (see note above)
        if iLen > 0 then
                filewrite(ifile,blobmid(blobtest,1,iLen))
        elseif iLen = ws.SOCKET_ERROR then  // oops, an error occurred.
                messagebox("Socket Error",string(ws.wsagetlasterror()))
        end if
end if // end of isvalid() if

// Now check to see if the ws.FD_CLOSE event triggered this script.
// If so, then the code within the previous if statement will have
// been skipped.

if iWhichEvent = ws.FD_CLOSE then
        fileclose(iFile)            // close the file since we're done writing
        sSocket.closesocket()       // close the socket
        destroy sSocket                    // destroy the socket object
        if status = 1 then open(w_menu)
        if status = 2 then open(w_account_info)
        if status = 3 then open(w_checking_trans)
        if status = 4 then open(w_all_trans)
        //fp = FileOpen(sFile, StreamMode!)
        //FileRead(fp, text)
        //mle_1.text = text
        //fileclose(fp)
        //ole_1.insertfile(sFile)     // put the file in the OLE 2.0 control
end if

/************* script for http()  window function which is********************/
/************* used in every window object ***************************/

ulong ulAddr      // the unsigned long version of the host's IP address
string sSend, sPath
int a

// the urlgetfilename() function strips the filename off of the URL
// NOTE: This is not an intelligent application. It assumes that the
// c:\temp\ directory exists and will convert the filename entered
// in the URL to DOS form: no more than 8 characters with a
// possible 3 character extension. A file with
// the extension .html is converted to .htm so that
// you can specify the URL for a UNIX-based HTML document
// without causing this program to choke. All other > 3 character
// extensions are suspect, however.
//
// sFileName is an instance variable of the window. Sloppy, but
// simple. The custom event triggered when data arrives on the socket
// needs to know this filename, and making it an instance variable
// was an easy solution.
//sFile = "c:\temp\" + urlgetfilename(URL)

sFile = "c:\temp\file.txt"

// sSocket is an instance variable of the window because it
// needs to continue to exist after this script is executed
sSocket = create socketstream       // create a new stream socket

// The URLGetAddress function accepts a URL and performs gethost
// DNS lookups to obtain the unsigned long address. No error
// checking is performed -- if the name or address resolution
// fails, you're out of luck.
ulAddr = URLGetAddress(URL)

// The URLGetPath function accepts a URL
sPath = URLGetPath(URL)

// perform a wsconnect() on the stream socket. Connect to port 80
// of the host specified in the URL. Port 80 is the standard http
// port.
a = sSocket.wsconnect(ulAddr,80)
if a = ws.SOCKET_ERROR then
        messagebox("Error","No connect")
end if

// Enable asynchronous messaging for this socket.
// the ws.wsor(ws.FD_READ,ws.FD_CLOSE) function returns the OR'ed
// value, which contains the appropriate bit-mask to tell the
// wsaasyncselect() function which events to monitor on the socket.
// handle(w_oletest) returns the window's handle.
// 1024 is the code for PowerBuilder's custom01 event: pbm_custom01
// this is the event that is executed every time either of the
// FD_READ or FD_CLOSE events occur on this socket.
sSocket.wsaasyncselect(handle(w_main),1024,ws.wsor(ws.FD_READ,ws.FD_CLOSE))

// this is the command sent to the HTTP server to retrieve the
// file identified in the URL. the ~r~n~r~n means carriage return (~r)
// linefeed (~n) carriage return (~r) linefeed (~n)
sSend = "get " + sPath + "~r~n~r~n"

// this sends the get command to the socket (which was connected to
// port 80 (the http server) of the specified host) -- again, there
// is no error checking or other intelligence here. If the connect
// failed for some reason, this is going to blow up.
sSocket.send(blob(sSend),len(sSend),0)

// check to see if the file that we want to write to exists.
// if it does, delete it just to be on the safe side. (not safe
// for the file in question, of course.)
if fileexists(sFile) = TRUE then
        filedelete(sFile)
end if

// open the output file
iFile = fileopen(sFile,StreamMode!,write!)

return

/**************** window function urlgetaddress() ******************/

ulong ulAddr
string sAddress
integer iPos
pbhostent pbheTmp
```

```
iPos = pos(URL,"://")+3

sAddress = mid(URL,iPos,pos(URL,"/",iPos)-iPos)

pbheTmp = ws.gethostbyname(sAddress)

if isnull(pbheTmp) = TRUE then
        ulAddr = ws.inet_addr(sAddress)
        pbheTmp = ws.gethostbyaddr(ulAddr)
        if isnull(pbheTmp) = TRUE then
                ulAddr = 0
        else
                ulAddr = pbheTmp.h_addr_list[1]
        end if
else
        ulAddr = pbheTmp.h_addr_list[1]
end if

return(ulAddr)
/****************** window function urlgetfilename()  *******************/

do while pos(URL,"/") > 0
        URL = mid(URL,pos(URL,"/")+1)
loop

if upper(right(URL,4)) = "HTML" then
        URL = mid(URL,1,len(URL)-1)
end if

if pos(URL,".") > 8 then
        URL = left(URL,8) + right(URL,4)
end if

return(URL)


/***************** winow function urlgetpath() **************************/

string sPath
int iPos

sPath = mid(URL,pos(URL,"://")+3)

iPos = pos(sPath,"/")

sPath = mid(sPath,iPos)

return(sPath)


/****************** window function  **********************************/

string sPrefix
```

```
sPrefix = left(URL,pos(URL,"://")-1)

return(sPrefix)
```

BIBLIOGRAPHY

Belford, Morgan. "WebDBC™ White Paper #1." Seattle: Nomad Development Corporation, 1995

Coombs, Jason. *PowerBuilder Power Toolkit.* Unpublished, 1995

Technical Empowerment Program Lecture Guide Boston: Open Environment Corporation, June 1995

"World Wide Web Consortium" **http://www.w3.org**

4271-84