

# **INTERACTIVE DATA-DRIVEN WEB APPLICATIONS**

by

Wai Man Raymond Chiu

B. Sc., Mathematics and Computing Science

Simon Fraser University, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE  
in the School  
of  
Computing Science

© Wai Man Raymond Chiu

SIMON FRASER UNIVERSITY

September 1997

All rights reserved. This work may not be  
reproduced in whole or in part, by photocopy  
or other means, without the permission of the author.



National Library  
of Canada

Acquisitions and  
Bibliographic Services

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

Bibliothèque nationale  
du Canada

Acquisitions et  
services bibliographiques

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

Your file *Votre référence*

Our file *Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de ~~cette~~ thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-24107-6

## APPROVAL

**Name:** Wai Man Raymond Chiu  
**Degree:** Master of Science  
**Title of Thesis:** Interactive Data-driven Web Applications

**Examining Committee:** Dr. Stella Atkins  
Chair

Dr. Wo-Shun Luk  
Senior Supervisor

~~Dr.~~ Ze-Nian Li  
Supervisor

Dr. Jiawei Han  
External Examiner

**Date Approved:**

September 30, 97

# Abstract

Great efforts have been made to develop mechanisms for delivering sophisticated applications over the Web in the past. Numerous technologies have recently been developed which not only make the Web an effective means for hypermedia information retrieval, but also give it a capability of executing interactive and high-impact Internet applications in a powerful and efficient manner. This is particularly true for Web database access technology. Traditional approaches basically drop the database connection once an operation has finished – hence operations are independent from each other. The newer on-line approaches either keep the database connection open throughout the whole session or effectively store the states of current users and possibly other information in the client-cache, thereby yielding better performance, higher capability, and a lower level of programmatic complexity.

Three basic issues are associated with Web database access technologies: (i) the efficiency of remote database access from a Web browser, (ii) the effectiveness of the graphical user interface (e.g. the level of user-friendliness and interactivity), and (iii) the effectiveness and flexibility of application development tools. This thesis investigates these three issues by comparing various architectures in order to evaluate the feasibility of using the newer technologies for developing sophisticated data-driven Web applications. To compare the newer techniques with traditional approaches, a series of quantitative and qualitative analyses will be presented, by means of experiments and sample applications.

# Acknowledgments

Getting enough and adequate information, knowledge, and experience to write a thesis of this size is definitely not an easy task. The completion of this thesis cannot be accomplished solely by the effort of one person. Many individuals have provided valuable advice and contributions. Although it is not possible to name all of them, I would like to take this opportunity to give them my thanks and express my appreciation to some who merit special recognition.

First, I especially would like to thank my senior supervisor, Dr. Wo-Shun Luk, who provided the guidance and substantial support for both hardware and software. Without his care and advice, I am quite sure this thesis would never have come to completion. His tireless support and direction has also made this learning process a very rewarding experience. I genuinely feel that I am very fortunate to have been his student.

I would also like to thank my supervisor, Dr. Ze-Nian Li for giving valuable comments on my thesis. Thanks also to Osmar Zaiane for helping me identify and understand certain relevant concepts from his insights and rich experience in the field of Web database access.

I am also thankful to the network support group of our School of Computing Science for their patient technical input. Finally, I would like to thank Dr. Kal Toth and Amy Wong, the proofreaders who corrected any mistakes and revised the structure of my final thesis.

# Contents

<b>Approval</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>iv</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 The HyperText Markup Language .....	2
1.2 The HyperText Transfer Protocol .....	2
1.3 Web-Based Applications .....	3
1.3.1 Data-driven Web Applications .....	4
1.4 Java .....	5
1.5 ActiveX .....	6
1.6 Objectives of the Thesis .....	6
1.7 Overview of the Thesis .....	7
<b>2 Traditional Web Database Access</b>	<b>9</b>

2.1	Web Browsers	11
2.1.1	Client-Side Scripting	11
2.1.2	Executable Content Approaches	12
2.2	Web Servers	13
2.2.1	Performance	13
2.2.2	Administration	14
2.2.3	Security	14
2.2.4	Application Development Environment	15
2.3	Interface between Web Server and Applications	15
2.3.1	Common Gateway Interface	15
2.3.2	Application Programming Interface	17
2.3.2.1	NSAPI	17
2.3.2.2	ISAPI	18
2.3.2.3	ASP	19
2.4	Interface between Web Server and DBMS	20
2.4.1	Microsoft Internet Database Connector	21
2.4.2	Netscape LiveWire	22
2.5	Stateless versus State-based Approaches	23
2.6	Off-line versus On-line Approaches	25

### **3 On-line Web Database Access 26**

3.1	The Java Language	27
3.1.1	Java Applets	28
3.2	ODBC	30
3.3	JDBC	31
3.3.1	JDBC-ODBC Bridge	32
3.3.2	Native-API Partly-Java Drivers	34

3.3.3	Native-protocol All-Java Drivers .....	35
3.3.4	Net-protocol All-Java Drivers .....	36
3.4	COM and ActiveX Technologies .....	38
3.4.1	Remote Data Object .....	39
3.4.2	Advanced Data Connector .....	41
<b>4</b>	<b>Performance Comparison</b> .....	<b>44</b>
4.1	Experimental Setup .....	45
4.1.1	JDBC with Symantec dbANYWHERE .....	48
4.1.2	JDBC with Intersolv JDBC/ODBC Bridge .....	48
4.1.3	Microsoft Remote Data Object .....	49
4.1.4	Microsoft Advanced Data Connector .....	50
4.2	Summary of Experimental Results .....	50
4.3	Interpretation of Experimental Results .....	52
4.3.1	Performance Difference in First and Subsequent Queries .....	54
4.3.2	Intersolv JDBC/ODBC Bridge VS. Microsoft RDO .....	55
4.3.3	Symantec dbANYWHERE VS. Microsoft ADC .....	56
<b>5</b>	<b>Prototype Web Database Application</b> .....	<b>58</b>
5.1	Evolution .....	58
5.2	Design Issues .....	59
5.2.1	User Interface and Interactivity .....	59
5.3	Implementation .....	65
5.4	Performance Issues .....	66
5.4.1	Session-oriented Experiment .....	67
5.4.2	Result and Interpretation .....	67
<b>6</b>	<b>Summary, Conclusion, and Future Work</b> .....	<b>72</b>



6.1	Summary.....	72
6.2	Conclusion.....	74
6.3	Future Work.....	75
<b>Appendix</b>		<b>77</b>
A	Complete Listing of Experimental Results: Chapter 4.....	77
B	Complete Listing of Experimental Results: Chapter 5.....	86
<b>Bibliography</b>		<b>87</b>

# List of Tables

Table 4.1:	Configuration of Web server and Web Client for the Experiment . . . . .	46
Table 4.2:	Querying MS Access 7.0 using Symantec dbANYWHERE . . . . .	51
Table 4.3:	Querying MS SQL Server 6.5 using Symantec dbANYWHERE . . . . .	51
Table 4.4:	Querying MS Access 7.0 using Intersolv JDBC/ODBC Bridge . . . . .	51
Table 4.5:	Querying MS SQL Server 6.5 using Intersolv JDBC/ODBC Bridge . . . . .	51
Table 4.6:	Querying MS Access 7.0 using MS RDO 2.0 . . . . .	52
Table 4.7:	Querying MS SQL Server 6.5 using MS RDO 2.0 . . . . .	52
Table 4.8:	Querying MS Access 7.0 using MS ADC 1.0 . . . . .	52
Table 4.9:	Querying MS SQL Server 6.5 using MS ADC 1.0 . . . . .	52
Table 5.1:	Querying MS Access 7.0 using MS IDC . . . . .	68
Table 5.2:	Querying MS Access 7.0 using MS ADC 1.0 . . . . .	68
Table 5.3:	Querying MS Access 7.0 using Symantec dbANYWHERE . . . . .	68
Table A.1:	Query 1 on MS Access 7.0 using Symantec dbANYWHERE . . . . .	77
Table A.2:	Query 2 on MS Access 7.0 using Symantec dbANYWHERE . . . . .	78
Table A.3:	Query 1 on MS SQL Server 6.5 using Symantec dbANYWHERE . . . . .	78
Table A.4:	Query 2 on MS SQL Server 6.5 using Symantec dbANYWHERE . . . . .	79
Table A.5:	Query 1 on MS Access 7.0 using Intersolv JDBC/ODBC Bridge . . . . .	79
Table A.6:	Query 2 on MS Access 7.0 using Intersolv JDBC/ODBC Bridge . . . . .	80
Table A.7:	Query 1 on MS SQL Server 6.5 using Intersolv JDBC/ODBC Bridge . . . . .	80

Table A.8:	Query 2 on MS SQL Server 6.5 using Intersolv JDBC/ODBC Bridge . . .	81
Table A.9:	Query 1 on MS Access 7.0 using MS RDO 2.0 . . . . .	81
Table A.10:	Query 2 on MS Access 7.0 using MS RDO 2.0 . . . . .	82
Table A.11:	Query 1 on MS SQL Server 6.5 using MS RDO 2.0 . . . . .	82
Table A.12:	Query 2 on MS SQL Server 6.5 using MS RDO 2.0 . . . . .	83
Table A.13:	Query 1 on MS Access 7.0 using MS ADC 1.0 . . . . .	83
Table A.14:	Query 2 on MS Access 7.0 using MS ADC 1.0 . . . . .	84
Table A.15:	Query 1 on MS SQL Server 6.5 using MS ADC 1.0 . . . . .	84
Table A.16:	Query 2 on MS SQL Server 6.5 using MS ADC 1.0 . . . . .	85
Table B.1:	Querying MS Access 7.0 using MS IDC . . . . .	86
Table B.2:	Querying MS Access 7.0 using MS ADC 1.0 . . . . .	86
Table B.3:	Querying MS Access 7.0 using Symantec dbANYWHERE . . . . .	86

# List of Figures

Figure 2.1:	Traditional Web database access model . . . . .	10
Figure 2.2:	Interaction between CGI executables and the Web server . . . . .	16
Figure 2.3:	Interaction between ISAPI Application DLLs and the Web server . . . . .	19
Figure 2.4:	Interaction between the Web Server and Databases via IDC . . . . .	22
Figure 2.5:	Stateless nature of HTTP client-server architecture . . . . .	24
Figure 2.6:	Maintaining user status during an HTTP session . . . . .	24
Figure 3.1:	Execution of Java program in a Java-enabled machine . . . . .	28
Figure 3.2:	Execution of Java applet in a Java-enabled Web browser . . . . .	29
Figure 3.3:	ODBC architecture . . . . .	30
Figure 3.4:	Database-enabled Java applet connection model . . . . .	31
Figure 3.5:	JDBC/ODBC Bridge model . . . . .	33
Figure 3.6:	Native-API Partly-Java Driver model . . . . .	34
Figure 3.7:	Native-Protocol All-Java Driver model . . . . .	35
Figure 3.8:	Net-Protocol All-Java Driver model . . . . .	37
Figure 3.9:	RDO model . . . . .	40
Figure 3.10:	ADC client/server model . . . . .	42
Figure 4.1:	Experiment Client/Server Test-Bed . . . . .	46
Figure 4.2:	Average access time to MS Access 7.0 . . . . .	53
Figure 4.3:	Average access time to MS SQL Server 6.5 . . . . .	53

Figure 5.1: Top level screen .....	60
Figure 5.2: Activity Logger window.....	61
Figure 5.3: Expanded menu bar.....	62
Figure 5.4: Edit activity window.....	62
Figure 5.5: Dialog window displaying error messages.....	63
Figure 5.6: Dialog window confirming activity update .....	63
Figure 5.7: View Preferences window .....	64
Figure 5.8: View Preferences window with corresponding Sort tab.....	64
Figure 5.9: Activity Logger window after filtering.....	65
Figure 5.10: Average individual query access time .....	68
Figure 5.11: Average cumulative query access time.....	69

# Chapter 1

## Introduction

The Internet is undoubtedly the most influential medium in our lives today, providing powerful and universal connectivity for information access; and its growth has been phenomenal during the past few years. As accessibility to the Internet has continued to grow and develop, the Web's capabilities have also moved ahead, no longer restricted it to simple document viewing. The Web is also ready for accessing interactive and dynamic contents. Additional Web-based applications are expected to evolve as currently available capabilities are ever being expanded. Recently, a lot of research has been carried out on designing better ways of developing and running Web applications. With the advances of computer and communication technologies, previously infeasible means of delivering interactive content through the Web have become reality. Innovative Web-based systems, including data-driven applications capable of linking live data as well as providing users interactive features have resulted. These refined systems not only provide an application environment with powerful new functions and features, but also dramatically decrease the client-server communications overheads consumed by traditional Web-based applications.

## 1.1 The HyperText Markup Language

The *HyperText Markup Language* [DH96, W3C97b], or HTML, provides a set of well-defined symbols specifying a single universal standard format for Web documents. Essentially, all data formats are supported including text, graphics images, and even streaming video. The most remarkable feature of HTML, perhaps, is its support for navigation enabling users to easily move among related documents. Although the specification of HTML is being constantly revised to extend its functionality, the interactivity of the Web supported by HTML is limited to selecting which material to view from the choices presented. The introduction of gateway programs that use files of hypertext on the Web for interface purposes allows some degree of interactivity. However, any computation must be performed on the server and true interaction is not possible through gateway programming alone. To bring the Web alive with a higher degree of interaction, more advanced technologies are needed.

## 1.2 The HyperText Transfer Protocol

The World Wide Web is built on a client-server model. Clients and servers communicate with each other using a common protocol. The *HyperText Transfer Protocol* [DH96, W3C97a], or HTTP, is a protocol for computers to speak as they exchange information through the Web. This protocol provides the necessary connectivity and interface for the Web. The HTTP was designed to efficiently access information across the Internet to handle a wide variety of data types. In fact, a file's data is only useful if its underlying type of data is known. With HTTP, the Web understands the corresponding data types of Web documents and passes that information along. Moreover, the HTTP offers the lightness and speed necessary for distributed and hypermedia information systems. The HTTP is based on a request/response paradigm. Typically, a client establishes a connection with a server and sends a request to the server in the form of request method, URL, and possibly other information. The server then responds with information including certain server information, body content, etc.

Although a connection is established between a client and a server, the HTTP protocol is known as connectionless or stateless because the connection is dropped and forgotten once the request has been responded. Each individual request is treated as discrete and brand-new, unrelated to any previous ones. Some other protocols, in contrast, are state-based and the connection is kept open. For instance, an FTP server keeps track of a client's information in an FTP session when a client is moving around in remote directories. An advantage of stateless systems is that they are relatively easy to write. However, it is exactly the stateless nature of HTTP that makes traditional Web applications incapable and inefficient, which leads to tremendous research into more advanced Web technologies.

### 1.3 Web-Based Applications

The popularity, simplicity, and performance of the Internet make it an excellent medium for conducting many applications. By combining versatile and sophisticated techniques for information retrieval and hypermedia, the World Wide Web has become the most popular service to access Internet information. Web documents can include numerous data formats such as text, graphics, sound, or video with little effort which makes the document highly multimedia. Hence, resources can be stored in different formats and existing resources can be easily made available with slight modifications. Another strength of the Web is that it provides a common user interface for Internet utilities such as FTP and Gopher. As a consequence, users can use their familiar Web browsers to reach everything offered by the Internet [Woo95]. Moreover, the ability to include active links and references to other Web pages also implies several benefits for a wide range of applications. It allows easy references traceable by following the links to various kinds of information in a consistent manner [Pim96]. Moreover, it provides great opportunities for structuring information and simplifying grasp of overall content by actively linking related documents. Therefore, it seems there are needs to build sophisticated Web applications that fit well with the Internet based environment.



On the other hand, Web documents today are largely static – they simply present information or a friendly interface for retrieving information from the user. Recent developments in Web technology, Web servers, and Web browsers further enhance the formatting of Web documents and encourage the creation of more “active” or “smarter” Web pages. As a result, highly interactive Web-based applications have become possible. Researchers then start designing even more advanced techniques to develop Web-based systems which allow information to be published in any favorite format within the context of Web browsers. Applications developed by these advanced and emerging technologies are even qualified to be compared with desktop applications to some extent. The successful introduction of these new techniques also provides promising resources to bring important changes in Web-based application systems.

### 1.3.1 Data-driven Web Applications

Central to the development of many applications would be data connectivity. The Internet phenomenon has propagated to the database community as Web data access opens up a number of options for interactive Web sites such as transaction processing and search engines. Moreover, database-enabled Web sites have the capability of providing valuable information in an organized, searchable, and easily modified format. However, there were very few, although inefficient, approaches for data connectivity through the Web in the past. Traditionally, the *Common Gateway Interface*, or CGI, is the only popular approach to generate dynamic Web documents. Although writing CGI programs to enable simple interactive features is not complicated, it is very inefficient since any interactivity means a communication with the Web server is required. For database applications, using CGI is even less efficient and not trivial at all. As a result, efficient and effective means for Web database access has become an imminent research issue.

The introduction of light-weight client-side scripting languages does improve the efficiency of interactive applications to a certain extent. Nevertheless, delivering data-centric Web contents using these languages is not feasible at all due to their limited functionality. Being a hot new field, Web database development has attracted the focus

of many researchers recently. The use and acceptance of executable contents in the Web allow highly interactive and data-centric Web-based applications to be created. Two of the representatives of Web executable content approaches are Java applets and ActiveX controls. Basically, they are light-weight reusable programming components which can be embedded in a Web page to increase the limited functionality of Web documents and can be used for data connectivity as well. More details of these two different kinds of components will be introduced in later chapters.

## 1.4 Java

Nothing has recently captured the attention of the Internet community as much as Java. As part of an advanced consumer electronics project at Sun Microsystems at the beginning, Java was designed to be a reliable and portable object-oriented programming language. Due to its tremendous capability, Java possesses all the essentials for extending the Web in ways that were previously inconceivable. Java brings true interactivity to the Web. Highly interactive applications such as games and database applications can now be encountered through the Web at remote network sites. Fundamentally, software implemented in Java can be safely distributed across the Internet and run on many different kinds of computers. Moreover, the resulting executable content shifts the site of activity from the Web server to the Web client.

Class libraries are continuously developed to extend the functionality of Java for creating advanced applications. One of such useful libraries, the *Java Database Connectivity*, or JDBC, API is developed to intimately tie connectivity to databases with the Java language. The JDBC defines every aspect of developing database-enabled Java applications while the low-level database-translations are performed by JDBC drivers. The implementation of the actual connection to the data source, whether it is local or remote, is left entirely to the JDBC driver. A whole bunch of vendors have endorsed the JDBC and sophisticated JDBC drivers are already available. However, early JDBC drivers are less capable and mature than recent ones. In essence, some early JDBC

drivers are LAN-based instead of Internet-based. More details of Java and JDBC will be covered in later chapters.

## 1.5 ActiveX

ActiveX is a specification developed by Microsoft for building reusable software components that can be integrated into a complete software solution. While the use of ActiveX is diversified, its use in the Web attracts the most attention. In fact, ActiveX can be used to develop virtually anything that can be achieved in traditional desktop applications. Moreover, any programming languages can be used in the implementation and the resulting native code will efficiently execute on appropriate platforms. Similar to the Java approach, ActiveX software components can be distributed across network and executed solely on the client side, which brings true real-time interactivity to the Web.

ActiveX is tightly integrated into the Microsoft's COM specification. While COM objects are suitable to be used in desktop applications, ActiveX addresses its focus to Web's usage. Due to the tremendous capability and efficiency of ActiveX, many different ActiveX components were developed to solve complicated problems that existed in applications implemented in other approaches. One of the most useful components recently developed is the *Advanced Data Connector*, or ADC, which provides a flexible yet efficient database connectivity model to Internet and Intranet applications. Details of ActiveX and specific components will be given in later chapters.

## 1.6 Objectives of the Thesis

In order to develop a highly sophisticated Web-based system which links to live data, various powerful technologies and software components will be employed in the development process. In almost all situations, the primary concerns of building Web applications will be user-friendliness, cost, and performance efficiency. The recently introduced Web technologies are increasingly adopted due to their tremendous capability and proven efficiency. This thesis evaluates the functionality and feasibility of different

technologies, tools, and components to be used in building Web-based data-driven applications. The benefits and tradeoffs of using them will also be discussed.

The objective of this thesis is a “proof of concept” attempt to develop a user-friendly yet effective Web database application, by using different technologies. As this is a “proof of concept” attempt, the intention of this thesis is not to develop a complete system. In order to demonstrate the concept, appropriate experiments will be performed and a simple Web database application will be implemented such that evaluations can be given from both a quantitative and qualitative viewpoint. Accordingly, experiments will be performed in order to compare the efficiency of traditional and newer approaches. The implementation of a prototypical data-driven Web application further demonstrates the vast flexibility of the user interface options available for building similar applications using the latest Web technologies.

## 1.7 Overview of the Thesis

In chapter 2, an introduction to traditional technologies for developing dynamic Web applications will be reviewed. First, descriptions of various components involved in a traditional Web database application will be given. An overview of a few representative techniques of early Web technologies for building generic and database specific Web applications will then be presented, together with a discussion of various issues regarding the mentioned approaches.

In chapter 3, more recent and advanced techniques for building Web applications will be introduced while concentration will be given to database specific development. Both the advantages and limitations of each approach will be presented together with a brief comparison to traditional approaches in a high-level sense.

In chapter 4, the set-ups, results, and interpretations of a series of experiments will be presented in order to give a quantitative analysis of the efficiency of the newer and more advanced Web technologies. The information presented in this chapter basically serves as a guideline for evaluating the feasibility of deploying these new approaches in building efficient Web database applications in different situations.

In chapter 5, an evaluation of the newer technologies will be given in a qualitative point of view. In essence, the capability and characteristics of a prototypical Web database application that deploys a new Web technology will be described. In particular, several screen shots showing the user-friendly interface of the underlying application will be presented to demonstrate the superiority of the newer technologies in comparison with traditional ones. The result and interpretation of a simple experiment will also be presented in order to compare the efficiency of traditional and the newer architectures.

In chapter 6, a summary and conclusion of the thesis will be given. It also describes any possible future advancement in Web technologies and outlines some ideas for possible future work.

## Chapter 2

# Traditional Web Database Access

The search to enhance interactive Web browsing techniques has been the enthusiastic development of powerful computer, communication, and programming technologies. However, the power of early Web applications is rather limited in terms of both the systems' functionality and communication ability. Early attempts in Web applications generally emphasized the use of the Internet with hypermedia to deliver static multimedia content rather than the delivery of dynamic information. Today, advances in computer and communication technologies provide powerful environment to develop sophisticated Web-based systems that can effectively support dynamic and live-data delivery and interaction. These experiences also serve as a foundation and informative guide for future developments in the area of Web and database applications. Traditional approaches such as the CGI and proprietary APIs are all capable of accessing remote Database Management Systems (DBMS), though quite vary in programming complexity and performance. The following diagram simply demonstrates such generic database access model through the Internet or Intranet.

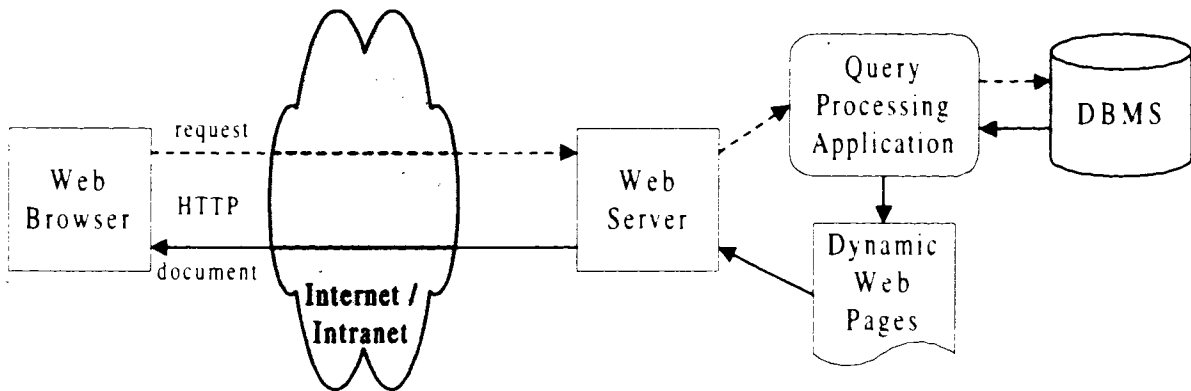


Figure 2.1: Traditional Web database access model

A Web browser serves as a classic thin client and provides a common interface, HTML, across many platforms. When the Web browser generates a request, the data-driven Web application communicates with the Web server through CGI or proprietary APIs using HTTP. The query processing application, which often resides on the same machine as the Web server, then handles the desired queries and manages state based on the request passed from the Web server. Resulting HTML page will be generated and passed on to the Web server according to the results retrieved from the back-end database. The Web server then passes the dynamically generated document to the Web browser for display.

In fact, a successful Web-based application cannot be built without the use of a good Web server and advanced programming technologies. Moreover, Web browsers have to support the more advanced technologies in order for any advanced technique to take effect. However, choosing the right Web server with appropriate programming models is not an easy task. In general, speed and extensibility are two of the important features of Web Servers. A good Web server platform should be capable of delivering high speed and secure information publishing as well as providing opportunities for developers to extend the Internet's standard communication capabilities. Apart from the Web server, certain technologies and components also play important roles in the development process. This section briefly introduces Web browser and Web server in general and various programming models that have been commonly employed in the development of Web-based application systems.

## 2.1 Web Browsers

A Web browser is a program that allows viewing of contents on the Web. The advantage is that a common interface can be used across various platforms. On the other hand, Web applications need to present its interface in terms of the relatively simple HTML format. In fact, the Internet is just another platform for client/server computing. However, it is a fundamentally different field that demands entirely new solutions. Traditional Web browsers are only capable of viewing plain HTML documents. Moreover, any execution must be performed in the server and client processing power is completely ignored in the picture. Recent technologies in Web browsing allow viewing of additional multimedia contents and even local execution of small applications. This section briefly introduces two of such advancements including client-side scripting and executable content approach.

### 2.1.1 Client-Side Scripting

The introduction of scripting languages, which were designed to help the non-programmer in creating simple interactive Web applications, brings Web application to a milestone. Scripting languages can actually be used for developing both client-sided and server-sided Internet applications. However, their use in the client side for Web-based scripting is more influential. Traditionally, no interaction can be performed in a Web client. Even simple task has to be performed in the server, for instance, by CGI script. In a client's application for Web navigation, scripting statements embedded in an HTML page can recognize and respond to user events such as mouse clicks, form input, and mouse movement over a link. Common usage of client-side scripting includes input validation and performing appropriate activity with respect to the user's action such as entering or exiting a page. Better still, all these tasks can be performed solely on the client side without any network transmission. In other words, client-side scripting offers the benefits of reducing network traffic and response times by keeping simple interactive tasks local.



Scripting languages are interpreted languages. It means that the scripting application code is downloaded as text into the Web browser along with the HTML text and executed directly within the browser and requires no compilation. Therefore, dynamic binding is being used and all object references are checked at runtime. Unlike most programming languages, scripting languages usually do not require any special declarations for their methods. Moreover, they support a run-time system based on a small number of data types. Most scripting languages have a simple instance-based object model that provides significant capabilities. That is, scripting languages are object-based that rely on their built-in and extensible objects, but they have no classes or inheritance, which existed in most object-oriented programming languages. Popular scripting languages include JavaScript and VBScript, which are very similar to Java and Visual Basic in their syntax respectively.

### **2.1.2 Executable Content Approach**

Today Web surfers are connected to heavy-loaded Web servers via relatively low bandwidth lines across the Internet. With today's powerful client computers, it makes sense to shift the workload of Web servers to Web clients, and to bypass as much as possible the growing traffic of the Internet. Although the introduction of client-side scripting languages does enhance the performance of Web browsing to a certain extent, scripting languages are designed to be lightweight and not intended to perform complicated and CPU-intensive tasks. Therefore, it is desirable to allow the local execution of certain programs so that they can take advantage of a host computer's processing power without increasing the load on remote Web servers. The missing link is a technology for safely distributing trustworthy executable content across the Internet.

With the advances in Web browsing technique, three different solutions tailored to these problems were recently developed. Plug-ins are software programs that add new capabilities to Netscape Navigator including cool audio, video, and other special formats on the Web. However, only Netscape Navigator natively supports plug-ins and users have to manually download and install the specific plug-in before using it. Sun

Microsoft's Java is perhaps the most influential recent Web technology. Java applets, which range from simple animations to full-featured Web applications, can be dynamically downloaded across the Internet and virtually support all platforms and browsers. Details of Java and Java applets will be given in the next chapter. Microsoft's ActiveX controls are program building blocks that can be assembled into Web applications. Moreover, ActiveX controls can be developed in virtually any programming languages and run as native code and hence better performance will be expected. However, only Microsoft Internet Explorer natively supports ActiveX controls. Browsers that support these new technologies are expected to give innovative changes to the Web by enriching its communication, information, and interaction.

## **2.2 Web Servers**

A Web server communicates with a Web browser using HTTP, which is a simple protocol for delivering distributed and collaborative hyper media information. A Web server receives request from a client that has established a connection to it. The Web server then processes the request, returns a response to the browser, and then closes the connection. Web servers can store and serve out any kind of file. HTML files and graphics are two typical examples. A Web server also runs applications such as search engines or database connectivity processes. While performance is undoubtedly an important feature of Web servers, issues such as setup, configuration, server management, administration, content management, security, access control, transaction management, and application development features are also important in evaluating Web servers. The following sub-sections give an overview of some of the most important ones.

### **2.2.1 Performance**

The question of how important the performance of a Web server is depends on what sort of Web site is being set up. Early phase of most Web servers did not focus on performance as an inherent part of their design. However, performance becomes a more

important issue for today's Web servers, which are expected to handle hundreds of requests received simultaneously. Moreover, Web servers are now frequently used to access other server-based applications such as database publishing and collaboration. Furthermore, Web pages with more dynamic contents such as 3-D, video, and audio have been moved to the Internet. Hence, the amount of required computing power possessed by Web servers have to be increased dramatically that leads to the increasing focus on the performance of Web servers.

### 2.2.2 Administration

Tools and services that ease administration are also important features of Web servers. Popular administrative features of today's Web servers include the following: *Virtual server support* is the ability to allow a single server to be configured to support as many TCP/IP addresses as desired. This feature is especially useful if the Web server serves as an Internet Service Provider hosting several Web sites or hosts multiple department's site in an Intranet. *Remote administration* is the activity of managing the Web server over the Internet in a secure and simple manner such that Web administrators will feel comfortable to do so. *Virtual directory management* is the option for Web administrators to distribute the physical storage of their published information while providing a different structure to external clients. This action is done by mapping logical URLs to physical directories. Although most Web servers have the features mentioned above, the evaluation of these features is usually based on efficiency, simplicity, and whether an intuitive interface is provided.

### 2.2.3 Security

Security, in the context of the Internet, includes protecting a Web site, restricting access to a Web site, and the degree of safety of data transfer between the server and the client, etc. Needless to say, security issues are crucial as Internet becomes more prevalent. *Basic authentication* is the most common way to provide security. For example, access restrictions can be achieved by the use of user names and passwords.

*Secure Sockets Layer*, or SSL, is a standard for encrypting data which provides a higher level of security than basic authentication does between the server and its clients when private communication is required. Other security features might be presented in a Web server's security model that suits other sorts of secure connections.

## 2.2.4 Application Development Environment

Server-based applications and database connectivity are in the forefront of the extension of Web servers' capabilities. More and more technologies for application development are emerging recently with intent to increase the appearance and possibility of dynamic and interactive Web documents. Among these approaches, the Common Gateway Interface, or CGI, is supported by almost all Web servers. Some Web servers also support other application programming interfaces, which allow developers to access specific functions on the Web server directly. Internet Database Gateway is another achievement in Web server technology, which is a powerful gateway for easy interfacing HTML documents with database information. Evaluation of these features is usually based on their capabilities, efficiency, and the degree of ease of use.

## 2.3 Interface between Web Server and Applications

In order to deliver high-impact and live data content via the Internet, Web sites have to move beyond the delivery of static HTML files. The ability to generate pages with information targeted at an individual client is also needed. Hence, Web servers must offer a comprehensive and efficient yet simple programming model for developers to deliver this enhanced functionality. Various techniques capable of generating dynamic and data-aware documents are available in Web servers nowadays. Descriptions of a number of representative programming models will be given in the following sections.

### 2.3.1 Common Gateway Interface

The Common Gateway Interface (CGI) [MC96, Ncs94] is a standard of interface for running external gateway programs under information servers such as HTTP or Web servers in traditional systems. It is also the most popular approach for developing dynamic Web applications and is supported by almost all Web server implementations. The Web server responds to a CGI execution for every request from a client browser by forking a new process. The data received from the client browser will then be passed to the CGI program through the environment variables and the script's standard input (*stdin*). In the case of using environment variables, the variables are set when the server executes the CGI program. Results generated by the CGI program will be sent to the script's standard output (*stdout*) of the newly created process. The output can be either documents generated by the CGI program, or instructions to the server for retrieving the desired output.

The following illustration shows the interaction of CGI executable files with a Web server.

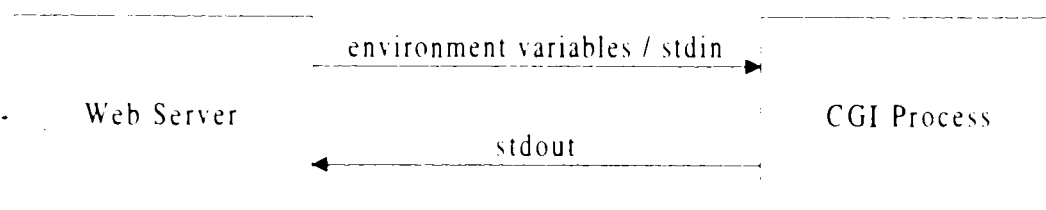


Figure 2.2: Interaction between CGI executables and the Web server

With the employment of CGI approach, Web servers create a separate process for each request received. However, this approach is time-consuming and executing a program frequently by the server is an expensive operation in terms of the server's main memory and other resources. Other consequences include slowing down performance and increasing client-waiting times on the Web. As services available through the Web are expected to increase continuously, more and more server-based applications will be developed. Advanced interfaces need to be designed to increase the performance of the existing server-executed CGI applications. As a result, more powerful and efficient approaches were designed to overcome the mentioned disadvantages. Nevertheless, CGI will continue to be used for quite some time due to their wide support by all major Web servers and Web browsers.

## 2.3.2 Application Programming Interface

Apart from the CGI, other proprietary APIs are supported by more advanced Web servers in order to ease the programming complexity and increase the performance of dynamic Web applications. Although APIs usually outperform CGI, several drawbacks are shared by all APIs. For example, most APIs only work on a limited number of servers and operating systems. Moreover, applications developed by API are easier to crash the server if code is poorly written. Furthermore, API code has to be sometimes written more carefully in order to deal with multi-threading, clean up, etc. Anyhow, most issues can be worked around by experienced programmers. The *Netscape Server Application Programming Interface*, or NSAPI, is a proprietary method used by a limited number of Netscape's Web server implementations. The *Internet Server Application Programming Interface*, or ISAPI, is another proprietary approach used by Microsoft's and some other vendors' Web server. Finally, the *Active Server Page*, or ASP, is an even more recent and advanced approach developed by Microsoft for delivering dynamic Web pages. The following sub-sections briefly introduce each of the three APIs approaches.

### 2.3.2.1 NSAPI

The Netscape Server Application Programming Interface (NSAPI) [NCC96] is an extension developed by Netscape Communications to extend the functionality of the Netscape server in order to solve performance and efficiency limitations of CGI functionality. The subtle design of NSAPI is mainly based on a logical breakdown of the HTTP request-response process. The definition of these logical steps is taken from experience with the feature sets of common Web servers. The steps should be chosen in a way such that the result of one step affecting the next while the methods employed in carrying out each step should not affect the next one.

After the logical steps have been identified, a set of *server application functions* determined by the inputs must be applied to accomplish each of the identified steps. The

inputs of these functions consist of the request itself and the server configuration database while a response will be returned to the client as output.

Seven classes of server applications existed while each of which corresponds to the request-response step it helps implement. *Initialization* is a special class of application function used to initialize static data such as logging and file typing for various server modules. *Authorization translation* is the class of functions for authentication. *Name translation* class functions translate a logical URL given by the client into a physical path as used by the server. *Path checks* class consists of those functions to verify whether or not a given path is safe to return to a given client by performing actions such as system-specific URL filtering and access control, etc. *Object type* class functions take the path resulting from the previous directives and try to locate a file system object for the path or return an error to the client if none exists. *Service* is the class of functions that sends the server's reply to the client. *Transaction log* functions simply log all transactions established by a client. Whenever any of these functions fail, the error must be handled by another function. The client must be informed by responses which can be customized by the administrator with site-specific information about the error.

### 2.3.2.2 ISAPI

The Internet Server Application Programming Interface (ISAPI) [MC97g] is a technique developed by Microsoft Corporation that serves as a powerful and high-performance alternative to CGI for delivering dynamic interaction and value-add extensions. The core of the differences between CGI and ISAPI is that CGI scripts are executable files while ISAPI applications are dynamic-link libraries, or DLL, containing functions that are compiled, linked, and stored separately from the processes which use them. As mentioned before, a server responds to a CGI execution request by creating as many processes as the number of requests received. However, this approach is inefficient in terms of both server time and resources. On the other hand, ISAPI application DLLs can be loaded and made resident in memory once a request is received such that it is ready to serve other requests until the server decides to respond to the requests.

Moreover, unlike CGI script-executable files, the ISAPI application DLLs are loaded in the same address space as the Web server which results in minimal overhead since all the sever available resources are also available to the ISAPI application DLLs.

In the case of CGI, Web servers communicate with the created process through environment variables and stdin/stdout. In contrast, interaction between Web servers and the ISAPI application DLLs is accomplished through *extension control block*, or ECB, which is a data structure containing all necessary client and server information for invoking ISAPI applications. The following illustration shows the interaction of ISAPI DLLs with a Web server.

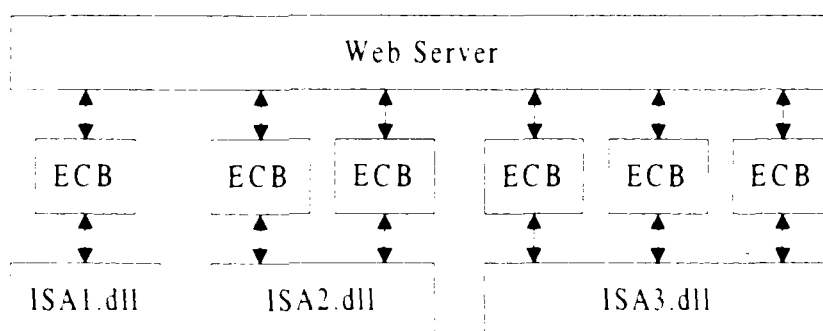


Figure 2.3: Interaction between ISAPI Application DLLs and the Web server

As mentioned earlier, less overhead and faster client/server interaction are expected in ISAPI application DLLs than CGI executables, especially under heavy load. On the other hand, more programming expertise is usually involved in developing ISAPI applications. For instance, multithreaded-safe ISAPI applications DLL must be developed since multiple requests will be received simultaneously.

### 2.3.2.3 ASP

The Active Server Page (ASP) [MC97a] is another application environment developed by Microsoft Corporation that allows the combination of HTML, scripts, and reusable components to create powerful interactive Web documents. Currently, ASP is only supported by Microsoft Internet Information Server [MC97f], which enables server-side scripting with virtually any scripting language while built-in support is provided for



VBScript [MC97j] and Javascript compatible JScript [MC97h]. In fact, ASP is very similar to server side includes in some aspects. The Web server parses ASP files and replaces the HTML-like tags with their value or output in the ASP file.

ActiveX components are objects which can be accessed from a Web page or other application to reuse packaged functionality someone else programmed. With proper server-side scripting, ASP can use ActiveX server components for a variety of tasks. For example, to retrieve records from a database, or access all Web server variables such as browser properties and referring page. Perhaps, this is the main difference and superiority of ASP over server side includes. A set of key ActiveX server components are shipped with IIS 3.0 while customized components can also be written to access virtually any kind of information accessible from the network. Moreover, since the scripts are processed by an engine on the Web server with standard HTML as the output, ASP works with any Web browser in any platform.

An advantage of ASP is that it is compile-free. That means a changed script is automatically compiled the next time it is requested. Moreover, ASP and CGI differ in their performance since ASP runs in-process with the server, and is optimized to handle large number of users. In general, ASP provides the flexibility of CGI programs and scripts without the significant performance tradeoff and development difficulties. However, one disadvantage of ASP is that it requires a fair amount of server CPU and memory overhead since the Web server has to scan through an entire active server page to find scripts and take the appropriate action.

## **2.4 Interface between Web Server and DBMS**

All programming models described in the previous section are capable of database access by periodical extraction of databases' data and generating dynamic Web pages based on the retrieved data. Database queries are built from user-input parameters, hidden variables, or cookies. However, the programming model involved is fairly complicated. For example, CGI requires more than 10 lines of script code for each individual field to be extracted from the database. Therefore, several proprietary tools for

Internet database gateway have been developed to ease the development of Web database applications. Most tools closely adhere to the SQL standard such that the programming model is both familiar to database developers and relatively easy to implement. This section introduces two of such approaches – the Microsoft Internet Database Connector and Netscape LiveWire, which were built into two popular Web servers.

### 2.4.1 Microsoft Internet Database Connector

The *Internet Database Connector* [MC97d], or IDC, is a component of Microsoft Internet Information Server that allows the Web server to efficiently gain access to databases. In fact, the IDC runs as a very thin server-based application (ISAPI DLL) that communicates with databases via *Open Database Connectivity*, or ODBC. The Internet Database Connector controls the access of databases and construction of resulting Web pages using two types of files one for each. The first one (descriptor) contains the query information necessary to connect to the appropriate data source and execute the SQL command. It also contains the name and location of the corresponding second type of file. The second one (template) is the template for the resulting Web page to be returned, which is a standard HTML file with special syntax for referencing the query. The following diagram briefly illustrates the components involved in connecting the Web server to databases.

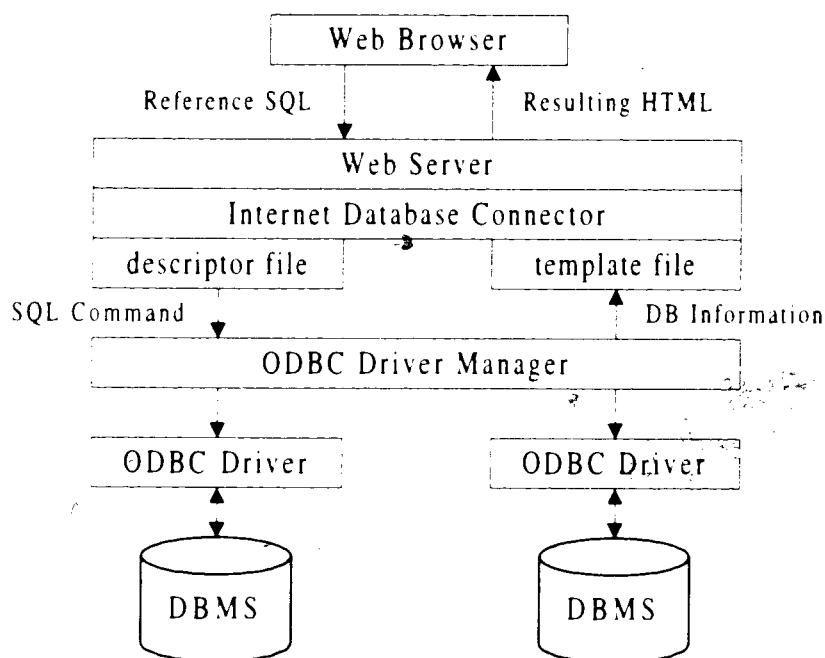


Figure 2.4: Interaction between the Web Server and Databases via IDC

The URL received by the Web server is parsed by IDC and the IDC process then loads a stored script (descriptor) that defines and invokes communications with the appropriate ODBC driver. The ODBC driver then communicates with a database and the retrieved results are converted into HTML pages using templates for delivery back to the client browser. The IDC is a very simple approach that requires virtually no special programming knowledge. Instead, developers using IDC are limited to templates and descriptor files to accomplish database access. This solution is well suited primarily to simple database queries that requires relatively little expertise. However, IDC opens and closes a connection with each incoming request, which can possibly slow down performance in many cases [Lin96].

## 2.4.2 Netscape LiveWire

The Netscape *LiveWire* [NCC97] is a proprietary Internet database gateway tool comes with Netscape Enterprise server for integrating database content into an HTML page. As a widely supported Internet scripting language that adds functionality to Web pages, JavaScript is being used with LiveWire to provide a complete development

environment that can work with data in relational databases. The database connectivity library of LiveWire supports native SQL client-server connectivity to ODBC compliant databases. Similar to the IDC approach, the Web server finds the request for LiveWire by parsing the incoming URL and the interaction with the database occurs from within the JavaScript code. Unlike IDC, JavaScript retrieves database results using the conventional database cursor model. A cursor is a pointer to rows in an answer set returned from the database of the requested query. The server-side JavaScript application simply reads each row from the cursor and converts it to the resulting HTML file [Lin96].

In comparison with the IDC approach, LiveWire requires relatively more programming expertise. However, LiveWire is capable of more feature-rich development than IDC. Moreover, LiveWire maintains a connection with the database throughout each session rather than opens and closes a connection with individual request, which might speed up performance. Anyhow, the choice of solution highly depends on the particular application being developed and the level of programming knowledge of developers [Lin96].

## **2.5 Stateless versus State-based Approaches**

Perhaps the most significant challenge facing early Web database development is the “stateless” nature of the Web. This characteristic makes every server interaction independent of all other interactions, so there is no notion of persistence. A Web server responds to page requests either by returning an HTML page or by triggering an external application via CGI or server API. Once the single request has been satisfied, the transaction is complete and the connection closes. The Web server makes no provision for storing vital information about the application and the user within the application. Although this approach is fine for delivering most Web documents, it creates huge problems for designing a highly interactive data-driven Web application. A database application, for instance, usually issues many queries based on user’s request, incurring the overhead of repeated connections. The following diagram briefly illustrates the stateless nature of HTTP architecture.

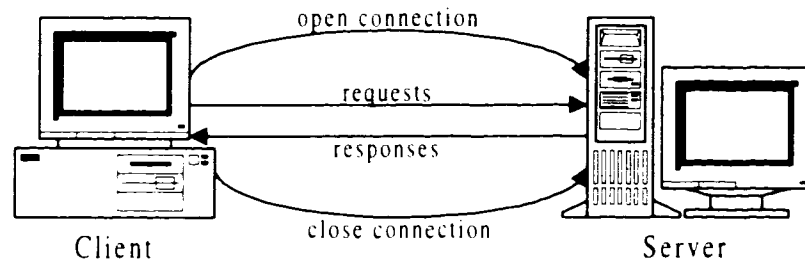


Figure 2.5: Stateless nature of HTTP client-server architecture

Regardless of the programming complexity involved, both the CGI and proprietary APIs are capable of accessing remote databases by maintaining session information or passing state information back and forth to the client. In general, a unique session identifier has to be generated on the server end by encoding the state or a state identifier in hidden fields, the path information, or URLs in the HTML form being returned. The specific information a Web database application maintains and how smoothly it is available to the application greatly affects the effectiveness of the system. A simple illustration is given in the following diagram.

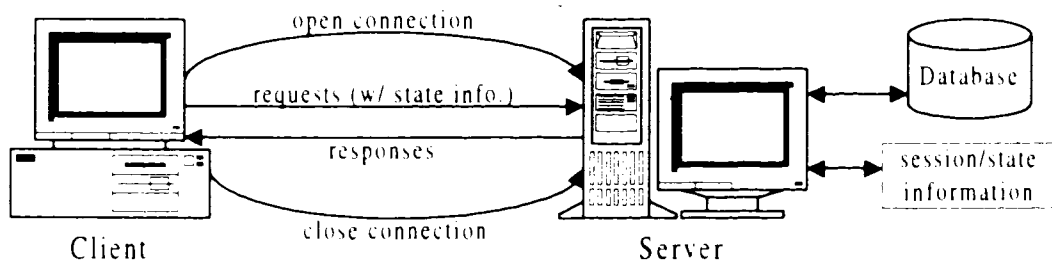


Figure 2.6: Maintaining user status during an HTTP session

In fact, it makes sense to employ a stateless rather than a state-based approach in certain situations. For example, it might not be appropriate for a heavy-loaded server to keep the states for thousands of users who may never complete the operations they start. Nevertheless, the stateless limitation makes these approaches not capable of developing highly interactive yet efficient database applications. One obvious disadvantage of stateless approaches is that connection needs to be re-established for every single database operation, which implies degrade of performance. Another drawback of such approaches is that considerable programming complexity is involved in order to simulate

a state-based connection using a stateless approach. Developers thus look into more advanced techniques for effectively accessing Web database. The newer state-based approaches typically keep the connection open within the whole session. One of such new technologies will be discussed in the next chapter.

## 2.6 Off-line versus On-line Approaches

Although the use of gateway programming mentioned in this chapter allows a Web server to intelligently interact with data and build HTML documents for the client dynamically, the data is static and is not really live. Users can work on the data and then later send it back to the server for update, but it is not interactive like traditional client/server applications. In order for actual on-line database access to take place, data can be transparently cached locally on the client side by deploying sophisticated mechanisms, which minimizes round-trips to the server. In essence, an advanced model must be present that provides the client with the capability to manipulate and update data. Therefore, sophisticated means must be designed to bring the meta-data and the data to the client so that a user can work on it on the client side. When the client application finished updating the data, the data will then be delivered to the server.

Database access via traditional off-line approaches is quite limited since it provides low flexibility of access paths by navigating pages via static links generated in the HTML pages. Hence, available operations are much more restricted than on-line accessible approaches. Moreover, highly dynamic and interactive services are not possible based on the use of CGI or server APIs since all active tasks must be performed at the server and no interaction is allowed in the client machine at all. As a result, the client remains completely passive in this case. Furthermore, a huge number of network transfers will be resulted since even simple input validation has to be done in the server [Kra97]. Several executable content approaches have already been developed to achieve on-line database access through the Internet. A few representative models of such configurations will be presented in the next chapter.

# Chapter 3

## On-line Web Database Access

As mentioned in the previous chapter, traditional approaches fail to develop highly interactive yet efficient database applications due to the low degree of interactivity of HTML and stateless nature of HTTP. Executable content approaches are recently being used for developing sophisticated Web applications. High level of interactivity becomes possible through executable content that has the ability to engage Web surfers in continuous, real-time, and complex interaction. Executable content approaches can also be used to access Web database on-line so that data can be manipulated and updated on the client side. Among various executable content approaches, component technologies and compiled languages are two of such useful techniques while the use of Java being the more important and widely adopted one. In essence, Java applications interface with data sources through JDBC, which is a specification of database specific programming model similar to the ODBC industry standard. This chapter will discuss various issues regarding on-line Web database access using Java Applets with JDBC, COM components and ActiveX controls while focus will be given to the Java approach.

## 3.1 The Java Language

The Java programming language [SM97] is used to create executable content that can be distributed through networks and was developed by Sun Microsystems and released in public alpha and beta versions in 1995. The development of Java began at Sun Microsystems in 1991 with the goal to create a programming language for a new set of consumer-electronics products. The focus of the language design is such that it can create processor-independent code to support a distributed network of communicating heterogeneous devices. While C++ was used as the starting point to implement this platform-independence, the team eventually abandoned C++ since C++ was not capable to do everything they wanted. The team then started developing Java as a small-footprint object-oriented programming language loosely based on C++. As an object-oriented programming language, Java possesses object-oriented properties such as inheritance and polymorphism, but has rather simple syntax by discarding the overwhelming complexities of similar object-oriented programming languages.

Java source code is compiled into byte-code, which is a high-level, machine independent and architecture-neutral intermediate format designed to transport code efficiently to multiple hardware and software platforms instead of platform-specific code native to any particular processor and operating system. The native architecture of Java is the *Java Virtual Machine (VM)*, a specification of an abstract machine such that executable code can be generated by Java compilers, which exists only in software today but will soon exist in hardware as well. The Java VM of a computer will be invoked to interpret and execute the Java byte code. Java byte-code is interpreted, which means that each byte-code instruction has to be parsed and interpreted by an interpreter and the process is fairly slow. Recent implementation of the virtual machine includes a *just-in-time (JIT)* compiler capable of compiling Java byte-code into native machine code on the fly which greatly improves the performance of Java byte-code.

As a result, Java applications are portable to any software and hardware platform that has a Java run-time environment. The environment consists of the Java VM, standard Java class libraries, a byte-code verifier for security purpose, and a byte-code



interpreter or JIT compiler that executes Java applications without requiring programmers to rewrite or even recompile their source code. Due to its cross-platform compatibility, Java transcends from being a programming language to being a software platform. Java developers, whether they realize it or not, are supporting a new platform that exists independently of the underlying operating system and hardware. The following diagram simply illustrates the execution process of Java programs.

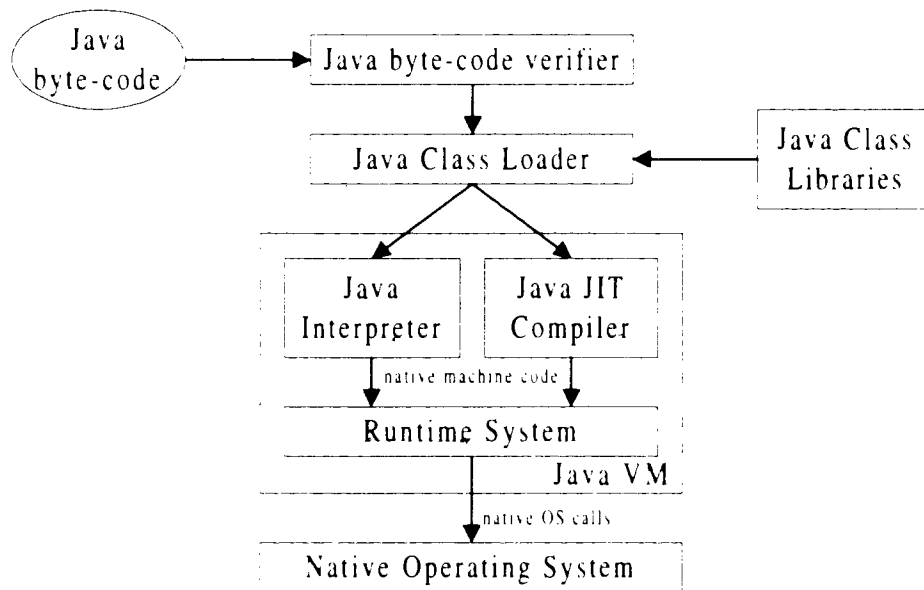


Figure 3.1: Execution of Java program in a Java-enabled machine

In fact, the performance of executing byte-code with JIT compiler is almost indistinguishable from direct execution of native C or C++ programs according to Sun Microsystems's testing. Moreover, it is exactly this level of "indirection" that gives the power, flexibility, and portability of Java code and makes Java so successful.

### 3.1.1 Java Applets

Applets are Java programs that run on top of a Java-enabled Web browser. When a page with an embedded applet is accessed by a user, either over the Internet or corporate Intranet, the applet will be automatically download from the server and run on the client machine. Applets are downloaded, hence they tend to be designed small or modular, to avoid large download times. Since Java applets may be loaded into systems

from random "uncontrolled" parts of the Internet, potential danger may enter a user's computer and an organization's Intranet. The Java language was designed to protect against both unintentional and malicious attacks against the integrity of the client's system. The underlying security restriction is the so-called "sandbox" approach. All conforming Java-enabled browsers provide a protected space known as the sandbox that restricts the range of things an applet can do on the client machine. For example, applets are not allowed to write to local file systems, access to memory, and spawn or exit a local process. The sandbox confines executable code to a run-time environment, seeking to neutralize any problem by limiting the reach of the code. The following diagram simply illustrates the execution process of Java applets within a Java-enabled Web browser.

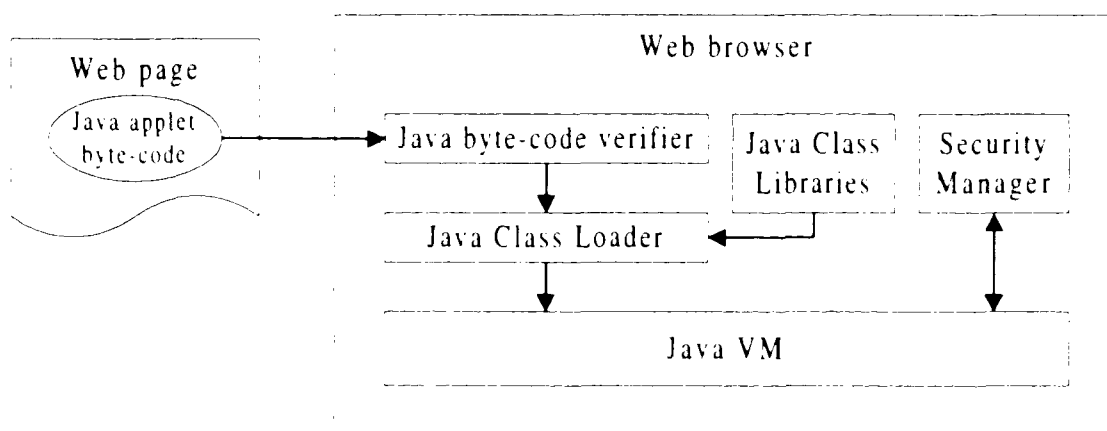


Figure 3.2: Execution of Java applet in a Java-enabled Web browser

Java security for downloaded applets relies on three components: the byte-code verifier, the applet class loader, and the security manager. These three components, together, perform load and run-time checks to restrict proper access. The byte-code verifier first performs format checks and static type checking for the untrusted code. The tests performed by the byte-code verifier range from simple verification of correct format for code fragment to passing each code fragment through a simple theorem prover to establish that it plays by certain rules. The class loader then attempts to load and instantiate all applets and the corresponding referenced classes. The security manager, the last defense of the Java security model, performs run-time checks based on the calling class's origin before a method is executed. The security manager thus has a chance to

forbid any dangerous operation if any is attempted. These three pieces of the Java security model enforce a program to perform particular operations only on particular kinds of objects so that untrusted Java applet can be safely executed on a client's system.

## 3.2 ODBC

The *Open Database Connectivity* (ODBC) [MC97k] specification defines a standard, database-independent interface for accessing data stored in heterogeneous SQL databases and is currently the most widely used programming interface for accessing relational DBMSs. The ODBC standard is based on work done by Microsoft and X/Open's SQL Access Group (SAG) with the aim of providing maximum interoperability so that a single application can access different SQL Database Management Systems (DBMS) through a common set of code. This characteristic enables a developer to build and distribute a client/server application without targeting a specific DBMS. Any ODBC API calls use the ODBC Driver Manager which manages interactions with ODBC drivers to link the application to the user's choice of DBMS. Each driver handles transactions with an actual database, using the corresponding DBMS client software and API. The following diagram simply illustrates the architecture of ODBC-based applications.

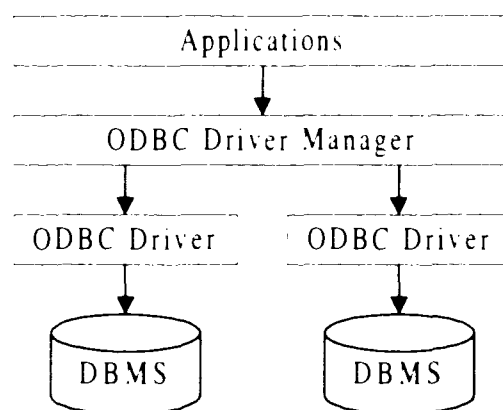


Figure 3.3: ODBC architecture

ODBC drivers are usually written in C or C++ and implemented as DLLs or shared libraries. The fact that ODBC drivers are not native Java components means that they cannot be downloaded from the Internet and interpreted at runtime in a Web

browser. Any ODBC drivers must be pre-installed on all client machines that use them. Moreover, since ODBC drivers are coded in C or C++ but not Java, they are not platform-independent and individual implementations must be developed for each operating system on which they will run. Furthermore, the native nature of ODBC drivers might pose security problems in Java's security model. As a result, the ODBC API is not suitable for database connectivity with Java and more sophisticated solutions must be developed to cope with the security, robustness, and platform-interopability characteristics of the Java language.

### 3.3 JDBC

In order to extend the functionality of Java as a serious platform for creating powerful and scalable client/server applications, a complete yet simple database connectivity model must be carefully developed. *Java Database Connectivity (JDBC)* [SM96a] is a specification developed by JavaSoft in 1996 that provides a uniform interface to tie connectivity to DBMS with the Java language. An important characteristic of Java applet is that Java connection can have an application session and store state information. The state-based connection of Java network programming is provided by socket objects that use TCP/IP as their transport mechanism, which is well suited for interactive applications. Moreover, Java applets run on the client side and can totally bypass Web browser/Web server connection. As a result, Java applets that communicate with databases using JDBC can efficiently access databases on-line throughout the whole database session. The following figure simply illustrates the connection model of database-enabled Java applets.

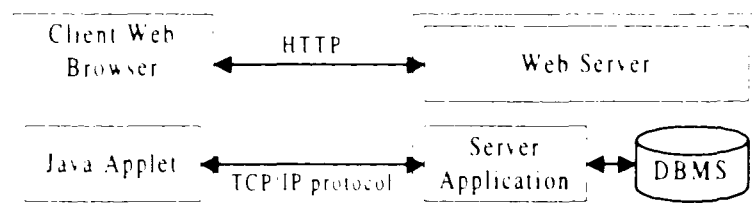


Figure 3.4: Database-enabled Java applet connection model

The JDBC API defines Java classes representing database connections, SQL statements, query result sets, database meta-data, etc. Moreover, the JDBC API is implemented via a driver manager that can support multiple drivers connecting to different databases. Experienced programmers can also use the JDBC to create and use low-level drivers to communicate with data sources. In the JDBC model, Java applications or applets use the JDBC API to load JDBC drivers which manage interactions with databases. Similar to ODBC, JDBC drivers are central to the architecture of JDBC. JDBC drivers can either be entirely written in Java so that they can be downloaded as part of an applet, or they can be implemented using native methods to bridge to existing database access libraries. JDBC drivers are generally segmented into four categories – JDBC/ODBC Bridge, Native-API Partly-Java drivers, Native-Protocol All-Java drivers, and Net-Protocol All-Java drivers. Descriptions and characteristics of each category will be given in the following subsections.

### **3.3.1 JDBC/ODBC Bridge**

The JDBC/ODBC Bridge [II97, SM96b] is a joint development of JavaSoft and Intersolv as a thin translation component that does low-level conversion from JDBC function calls into ODBC function calls. As JDBC is designed to be efficiently implementable on ODBC, the bridge is the best way to utilize ODBC from Java applications. The JDBC/ODBC Bridge allows Java developers to code JDBC-compliant applications and applets, then deploy them with any existing ODBC drivers readily available in the market today. As the JDBC specification is still very new, the JDBC/ODBC is an early attempt to allow database access for Java programs. The following diagram simply illustrates the place of JDBC/ODBC Bridge in the overall architecture of the JDBC model.

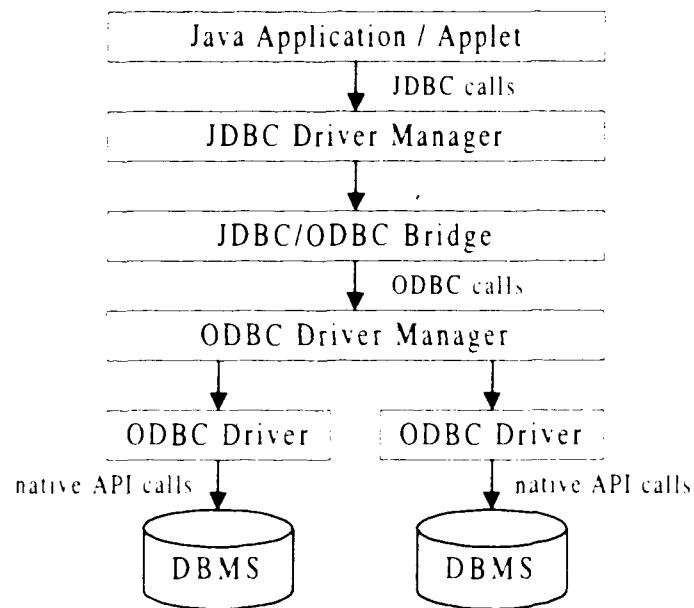


Figure 3.5: JDBC/ODBC Bridge model

The query process for a general database-enabled Java applet using the JDBC/ODBC Bridge can be described by the following steps:

1. The user launches a Web browser and starts the database applet.
2. The applet verifies and connects to the appropriate data source.
3. Loop
4. The user submits the required information specifying the query.
5. The applet passes the JDBC query to the JDBC Driver Manager.
6. The JDBC driver manager loads the JDBC/ODBC Bridge and passes the JDBC query to the bridge.
7. The bridge translates any JDBC calls to ODBC calls and passes the results to the ODBC driver manager.
8. The ODBC driver manager passes the ODBC query to the appropriate ODBC driver.
9. The ODBC driver translates any ODBC calls to DBMS native calls and submits the results to the remote DBMS.
10. The DBMS processes the query.
11. The DBMS passes the query result back to the invoking applet.
12. The applet displays the result.
13. The applet closes the database connection.

With limited availability of sophisticated JDBC drivers, the JDBC/ODBC Bridge allows developers to begin coding data-centric Java applications and leverage existing ODBC technology already deployed in organizations today. However, pre-installation of the bridge and any ODBC drivers in all client machines is necessary since both the bridge and ODBC drivers are written in native code. Its native nature also means that the bridge

approach is not platform-independent. Moreover, the JDBC/ODBC Bridge can only be used with trusted applets which must be pre-installed on the client's machine. Due to its limitations, the JDBC/ODBC Bridge should only be used with LAN-based Java applications or applets and not suitable for use with downloaded applets across the Internet or Intranet.

### 3.3.2 Native-API Partly-Java Drivers

A native-API partly-Java driver [H97, SM96b] translates JDBC calls into client API calls of the specific targeted DBMS. Since Java classes cannot directly access the native client libraries of network transport software without going through a special Java bridge DLL or shared library, some binary code must be loaded on each client machine. The following diagram illustrates the architecture of native-API partly-Java drivers.

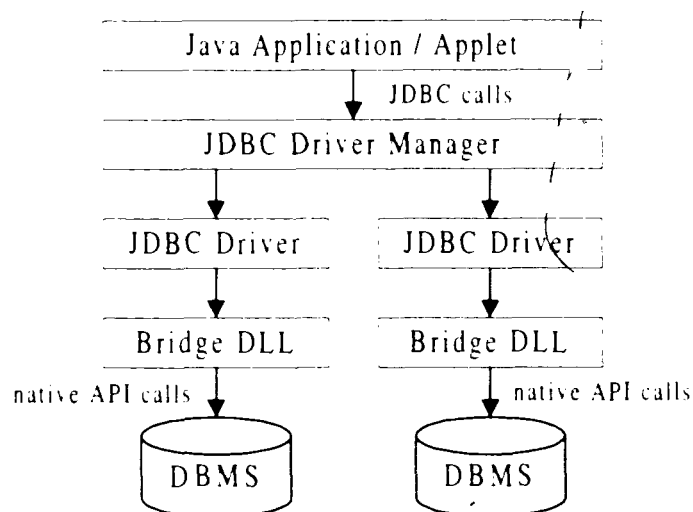


Figure 3.6: Native-API Partly-Java Driver model

The query process for a general database-enabled Java applet using a native-API partly-Java driver can be described by the following steps:

1. The user launches a Web browser and starts the database applet.
2. The applet verifies and connects to the appropriate data source.
3. Loop
4. The user submits the required information specifying the query.
5. The applet passes the JDBC query to the JDBC driver manager.
6. The JDBC driver manager passes the JDBC query to the appropriate JDBC driver.

7. The JDBC driver loads the required Bridge DLL and passes the JDBC query to the Bridge DLL.
8. The Bridge DLL translates any JDBC calls to DBMS native calls and submits the results to the remote DBMS.
9. The DBMS processes the query.
10. The DBMS passes the query result back to the invoking applet.
11. The applet displays the result.
12. The applet closes the database connection

Native-API partly-Java drivers share lots of similarities with the JDBC/ODBC Bridge. A difference between them is that the resulting calls from a native-API partly-Java driver are native to a specific DBMS while those from the JDBC/ODBC Bridge are ODBC calls. One benefit over the bridge approach is that no ODBC layer is required and hence better performance will be expected. However, similar to the JDBC/ODBC Bridge, the native architecture of these drivers makes them only suitable to be used in corporate networks since tremendous client pre-installation is required.

### 3.3.3 Native-Protocol All-Java Drivers

A native-protocol all-Java driver [II97, SM96b] translates JDBC calls into the network protocol used by DBMSs directly, which allows clients to make direct calls to database servers. Basically, specific pure Java JDBC driver is used instead of the network transport software of the particular DBMS. The following diagram illustrates the architecture of native-protocol all-Java drivers.

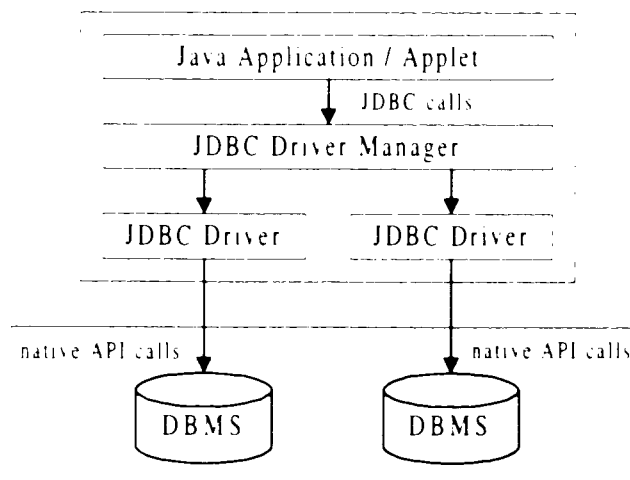


Figure 3.7: Native-Protocol All-Java Driver model



The query process for a general database-enabled Java applet using a native-protocol all-Java driver can be described by the following steps:

1. The user launches a Web browser and starts the database applet.
2. The applet verifies and connects to the appropriate data source.
3. Loop
4. The user submits the required information specifying the query.
5. The applet passes the JDBC query to the JDBC driver manager.
6. The JDBC driver manager passes the JDBC query to the appropriate JDBC driver.
7. The JDBC driver translates any JDBC calls to DBMS native calls and submits the results to the remote DBMS.
8. The DBMS processes the query.
9. The DBMS passes the query result back to the invoking applet.
10. The applet displays the result.
11. The applet closes the database connection.

The use of native-protocol all-Java drivers eliminates the need of server-sided applications for APIs translation, which might in turn improve performance. This kind of drivers can be used with Java applications, downloaded applets in any Java-enabled platforms across both the Internet and Intranet environments. On the down side, DBMS client network transport software is typically proprietary, hence the database vendors themselves will be the primary source for this style of driver. Moreover, a Native-Protocol All-Java driver is for a specific database, as opposed to being a universal driver that can connect to a variety of databases. Therefore, an applet that requires connections to multiple databases will need to download multiple versions of this style of JDBC driver, which can be inefficient.

### 3.3.4 Net-Protocol All-Java Drivers

A net-protocol all-Java driver [II97, SM96b] translates JDBC calls into a DBMS-independent net protocol, which is then translated to a DBMS specific protocol by a server middleware. This net server middleware consists of a single universal all-Java driver that is able to connect its all Java clients to many different databases. Depending on the design of the server middleware, the middleware communicates with DBMS either directly or through the use of ODBC drivers. The ability to connect to DBMS through ODBC allows data access to a large number of ODBC data sources, but perhaps, with a

little tradeoff of performance. In general, this approach is the most flexible alternative that is suitable for Intranet use. In order for these drivers to also support Internet access they must handle the additional requirements such as security, access through firewalls that the Web imposes. Several vendors are adding JDBC drivers to their existing database middleware products. The following diagram simply illustrates the architecture of net-protocol all-Java JDBC drivers.

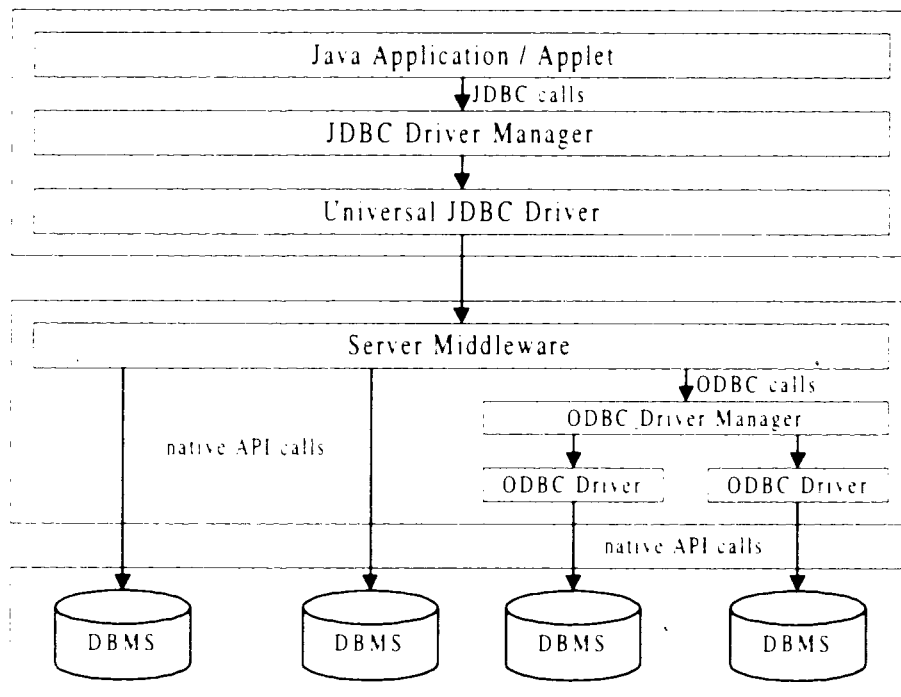


Figure 3.8: Net-Protocol All-Java Driver model

The query process for a general database-enabled Java applet using a net-API all-Java driver can be described by the following steps:

1. The user launches a Web browser and starts the database applet.
2. The applet verifies and connects to the appropriate data source.
3. Loop
4. The user submits the required information specifying query.
5. The applet passes the JDBC query to the JDBC driver manager.
6. The JDBC driver manager passes the JDBC query to the universal JDBC driver.
7. The JDBC driver passes the JDBC query to the server middleware.
8. If (ODBC layer is used)
9. The middleware translates any JDBC calls to ODBC calls and passes the results to the ODBC driver manager.
10. The ODBC Driver Manager passes the ODBC query to the appropriate ODBC driver.

11. The ODBC driver translates any ODBC calls to DBMS native calls and submits the results to the DBMS.
12. ELSE
13. The middleware translates any JDBC calls to DBMS native calls and submits the results to the DBMS.
14. The DBMS processes the query.
15. The DBMS passes the query result back to the invoking applet.
16. The applet displays the result.
17. The applet closes the database connection.

In general, net-protocol all-Java drivers are the most flexible JDBC connectivity solution. As the driver is completely implemented in Java, no client pre-installation is required. In addition, a net-protocol all-Java driver runs on any Java-enabled platform. The major benefit of all-Java drivers is that they can be fully downloaded and do not need to be pre-installed on the clients. Similar to native-protocol all-Java drivers, this kind of drivers can be used with Java applications, downloaded applets across both the Internet and Intranet environments. One limitation of this approach is that downloaded applets can connect back only to the host from which the applet was downloaded. However, the use of server-based middleware allows a true three-tier network database architecture.

## 3.4 COM and ActiveX Technologies

COM and ActiveX are Microsoft's component models that cover a broad range of network and multimedia technologies. COM, which stands for Component Object Model, is the underlying object architecture of the 32-bit Windows interface. The primary responsibility of COM is to allow software components to behave consistently without imposing design and implementation restrictions. Software components need only to adhere to a binary external standard, but their internal implementation is completely unconstrained. Objects conforming to COM can communicate with each other without being programmed with specific information about each other's implementations. Therefore, software components can be easily designed to cooperate with one another, even though they were written in different programming languages by different developers at different times [MDE95].

Being an industry standard for implementing reusable PC Windows components adhering to the COM specification, Microsoft's OLE architecture is being used as the underlying infrastructure of the ActiveX technology. Simply speaking, ActiveX can be visualized as OLE on the Web, which was developed by Microsoft to primarily enhance the Internet and multimedia products. With the goals of extending Internet standards, the key part of ActiveX strategy is building a Windows interface for working on both the PC and the Web. Being largely based on the COM specification, ActiveX is an unalterable set of rules introduced for interoperability between software components. Basically, ActiveX absorbed the OLE technologies and extended them to facilitate the development of Web applications. As a result, ActiveX controls can both be deployed in desktop and Web applications. In fact, ActiveX controls can virtually do anything that can be done by desktop applications. Therefore, ActiveX technology brings a new level of interactivity to Web browsing. The following subsections introduce two data-aware components, which were deployed in the experimental application.

### **3.4.1 Remote Data Object**

The *Remote Data Object*, or RDO, is a COM object providing interfaces to ODBC data sources. Basically, RDO is a thin object layer interface to the ODBC API with some special features like server-side cursors for efficient access to database server in traditional client/server applications. RDO is especially designed for building and executing queries against stored procedures and handling all types of result sets. A remarkable feature of RDO is that RDO is fully asynchronous and event-driven, so there is no need to poll for task completion as an event is fired. Moreover, RDO is thread-safe. Therefore, the ability of 32-bit Windows environment to run multiple threads of execution can be fully utilized.

Access to remote ODBC data through RDO is achieved by an interface for using code to create and manipulate components of an ODBC compliant database system. Objects within the RDO framework have properties that describe the characteristics of database components and methods used to manipulate them. Using the containment

framework, relationships can be created among objects, and these relationships represent the logical structure of the database system. Connections to databases are established via the thin code layer over the ODBC layer and the driver manager. The following diagram simply illustrates the interface model of applications that use the RDO.

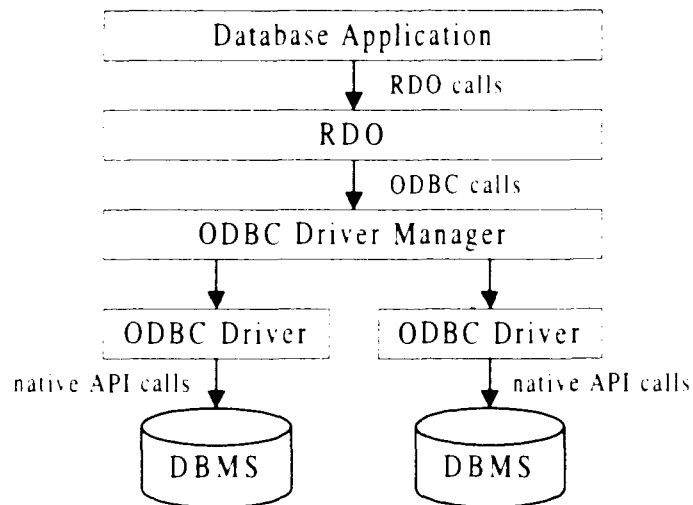


Figure 3.9: RDO model

Although RDO is primarily designed to be used in traditional client/server applications, it can also be integrated with Java for developing Web database applications. As RDO is based on ODBC, RDO is a state-based approach such that a database connection is kept open. The query process for a general database-enabled Java applet using the RDO can be described by the following steps.

1. The user launches a Web browser and starts the RDO-based applet.
2. The applet verifies and connects to the appropriate data source.
3. Loop
4. The user submits the required information specifying the query.
5. The applet passes the RDO-based query to the RDO.
6. The RDO translates any RDO calls to ODBC calls and passes the results to the ODBC driver manager.
7. The ODBC driver manager passes the ODBC query to the appropriate ODBC driver.
8. The ODBC driver translates any ODBC calls to DBMS native calls and submits the results to the remote DBMS.
9. The DBMS processes the query.
10. The DBMS passes the query result back to the invoking applet.
11. The applet displays the result.
12. The applet closes the database connection.

Although a thin code layer is present on top of the ODBC layer, RDO's performance is, in most cases, virtually identical to the ODBC API but with radically reduced coding time. However, the architecture of RDO is based on persistent sockets/pipes connections to databases, so it is suitable to be used in a LAN-based environment rather than in an Internet environment.

### **3.4.2 Advanced Data Connector**

The Advanced Data Connector [MC97b, MC97e, Rau97], or ADC, is a technology (ActiveX control) developed by Microsoft that tightly integrated with Microsoft Internet Information Server (IIS) and ActiveX Data Objects (ADO) to provide flexible database connectivity to Internet and Intranet applications. ADO is a high-level database programming model for developers to write database applications on top of OLE DB. While OLE DB defines a set of low-level C/C++ interfaces designed to efficiently build database components, ADO provides a programming model that is suitable to be called directly from high-level programming languages such as Visual Basic and Java, or scripting languages such as VBScript and JavaScript. With universal data access as the primary goal, an OLE DB layer is used instead of ODBC so that a variety of data sources can be accessed from the same programming model instead of only data from relational databases. OLE DB achieves this by identifying common characteristics between different data providers and services, including ODBC, by defining common interfaces to expose those characteristics. The following diagram simply illustrates how ADC and other components work together for remote data access.

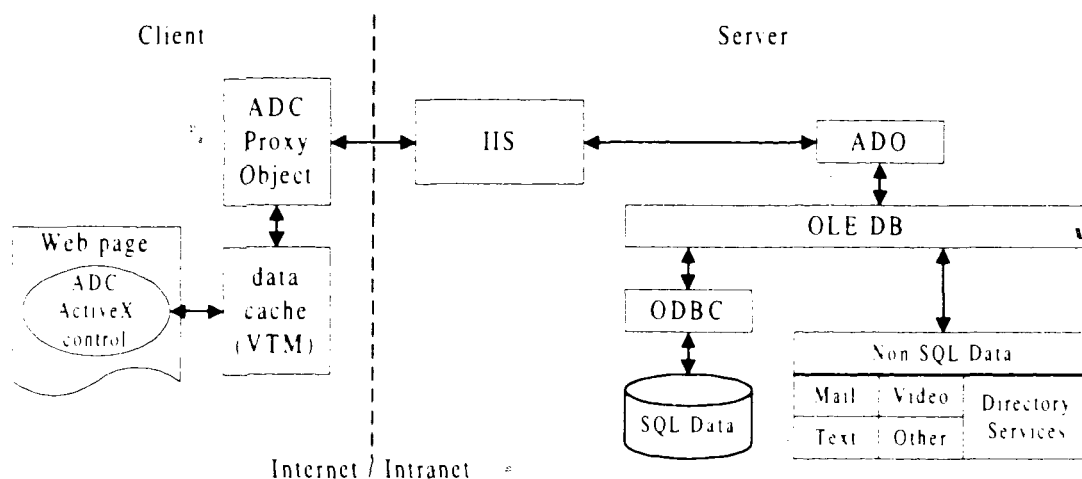


Figure 3.10: ADC client/server model

As illustrated in the above figure, two key components are present in the client space. The first one is the ADC Proxy object, which basically packages up the method requests from the ADC and sends them to the Web server over HTTP. Another one is known as the Virtual Table Manager, or VTM, which is a key component in ADC's client-caching model. The VTM is an in-memory relational data cache exposing OLE DB interfaces for data access and manipulation. The VTM supports state marshaling of its contents through special interfaces among multiple server tiers while providing client-side disconnected cursor models over its cache elements. It also maintains relational data, client updates, and records status information. The query process for a general Web database application using the ADC can be described by the following steps:

1. The user launches a Web browser and starts the application.
2. Loop
3. The user submits the required information specifying the query.
4. The ADC verifies and connects to the appropriate data source.
5. The ADC passes the query to the Web server (IIS) via the ADC Proxy object.
6. The Web server passes the query to the ADO.
7. The ADO translates the query into appropriate API and passes it to the OLE DB layer.
8. The OLE DB layer translates the query into ODBC API and passes it to the ODBC layer.
9. The ODBC layer translates the query into DBMS native calls and passes it to the DBMS.
10. The DBMS processes the query.
11. The DBMS passes the query result to the calling application.
12. The application displays the result.
13. The application closes the database connection.

As noted in the above algorithm, connection needs to be reopened and closed for individual database operation. In other words, ADC is basically a stateless database access approach. However, ADC deploys a sophisticated client-side caching mechanism that minimizes connections to DBMS. As a compromise between state-based and stateless database access technologies, ADC achieves a performance comparable to most state-based approaches while avoiding the server to remember the states of its tremendous number of clients. Such improvements are especially noticeable when accessing data across the Internet. Moreover, similar to Java, the ADC can be used in conjunction with other ActiveX controls for developing Windows-style user interface. This feature will be discovered in more details in chapter 5.



# Chapter 4

## Performance Comparison

The performance of on-line Web database applications varies according to the underlying database access technologies being used. In fact, some approaches can be used to develop Internet-based applications while others are suitable only for applications to be used within a corporate network. Although the performance of any technology should be sufficient for simple database access within a particular environment, large variations may be experienced in complex applications. In order to get an idea of how various technologies perform, a single operation experiment was carried out to compare the response times of a few database queries using these technologies with different database management systems. Four approaches (or products) were tested including:

- JDBC using Symantec dbANYWHERE Workgroup Server 1.0
- JDBC using Intersolv JDBC/ODBC Bridge 1.01
- Microsoft Remote Data Object (RDO) 2.0
- Microsoft Advanced Data Connector (ADC) 1.0.

A data-centric Web application accesses a remote database server according to the user's query and displays the extracted information. Although the exact steps involved vary, the query process for a general Web database application can be described by the following steps.

1. The user launches a Web browser and starts the application.
2. The application verifies and connects to the appropriate data source.
3. The user submits the required information specifying the query.
4. The application submits the query to the remote data source.
5. The DBMS processes the query.
6. The DBMS passes the query result to the invoking application.
7. The application displays the result.

The objective of the experiment is to determine the overall efficiency of various approaches. Hence fairly simple database applications were implemented for each. Although there are slight differences in the implementation of these applications, measurements were only made on database specific aspects of the execution of these functionally equivalent applications. Two quantities were recorded for all tests. Firstly, the average connection time indicates the time required to complete a client's request for setting up a database connection – this corresponds to the time required for step 2 of the above process. Secondly, the average query time indicates the time required to complete and respond to a database query – this corresponds to the time required for steps 4 to 6. These two quantities are expected to consume considerable processing time and are considered to be sufficient to reflect the overall efficiency of each approach. Note that step 5 is a common element of all approaches and should be dropped out. However, steps 4, 5, and 6 usually comprise a single indivisible operation. Nevertheless, the same processing time by step 5 can be expected by all approaches since this step is performed by the database management system, not the underlying technology being used. Details and interpretations of the experiment are discussed in the following sections.

## 4.1 Experimental Setup

An internal network with an Internet connection was established to perform the experiment because some of the approaches are not Internet-ready and only work in a LAN-based configuration. Microsoft Internet Explorer 3.02 was used as the Web browser for all except one approach – a Netscape plug-in was required for that particular approach. In that case, Netscape Navigator 3.01 was used. In both cases, the JIT compiler for Java was used to speed up the performance of Java-based applications. The

performance tests were only carried out on a single test suit given our primary interest was the performance difference among various database access approaches rather than different operating systems, Web servers, or server hardware. The configuration of the Web server and Web client are profiled by the following table.

Table 4.1: Configuration of Web server and Web Client for the Experiment

	Web Server	Web Client
<b>Hardware Platform</b>	<ul style="list-style-type: none"> <li>• Single Intel P5-166 CPU</li> <li>• 512KB cache</li> <li>• 3.1 Gigabyte Hard Disk</li> </ul>	<ul style="list-style-type: none"> <li>• Intel 486DX2 66MHz CPU</li> <li>• 256KB cache</li> <li>• 540 Megabyte Hard Disk</li> </ul>
<b>Operating System</b>	Windows NT Server 4.0 with Service Pack 3	Windows 95
<b>Web Server</b>	Microsoft IIS 3.0	N/A
<b>RAM Memory</b>	64MB	16MB

The server and workstations were connected to a 10Base-T Hub over an internal network using their internal 10Base-T adapters to factor out connection limitations such as modems or communications links. The internal network was connected to the Internet using its local LAN T1 communication link. Among the workstations, one of them was consistently used as the Web client for all experiments. The test-bed of the Web server is illustrated in the following figure.

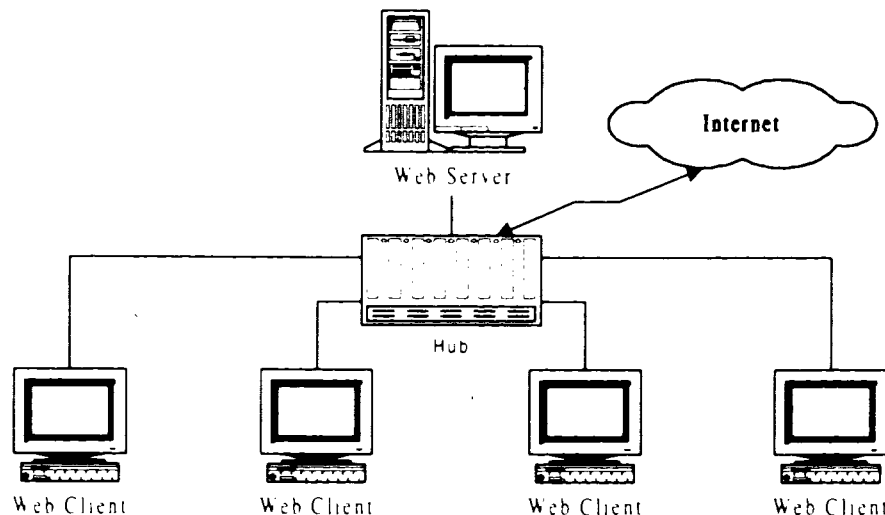


Figure 4.1: Experiment Client/Server Test-Bed

All tests were executed against two database management systems – Microsoft Access 7.0 and Microsoft SQL Server 6.5. An ODBC layer is present in all approaches and hence the use of ODBC drivers is necessary. The Microsoft Access ODBC driver 3.50 was used for Microsoft Access while the Intersolv SQL Server ODBC driver 3.00 was used for SQL Server. The Intersolv SQL Server ODBC driver was used instead of the Microsoft equivalent which could not handle multiple queries within the same database session.

Two queries, with fairly distinct levels of complexity were consistently used throughout the experiment. Both queries perform a selection from some relations (or tables) in the database. The database consists of five relations with appropriate information for assigning grades to students. Detailed design issues relating to the database itself are not directly relevant and are therefore not presented.

The five relations managed by the database may be specified as follows (note that a \* symbol is associated with attribute(s) that represent the primary key):

```
account_info(*login_id, first_name, last_name, position, student_id, email)
courses(*course_id, *login_id, priority)
activity(*activity_id, activity_name, course_id, out_of, percentage, display_order)
marks(*login_id, *activity_id, mark)
grade_range(*course_id, *letter_grade, min_mark, max_mark)
```

The first query (Q1), a fairly simple one, selects all records from a particular table. The second query (Q2), relatively speaking a much more complex and time consuming one, selects all records from the result of the natural join of the five tables. These queries are specified as follows:

```
Q1:  select * from account_info
```

```
Q2:  select * from account_info, courses, activity, marks, grade_range
where account_info.login_id = courses.login_id
and account_info.login_id = marks.login_id
and courses.course_id = grade_range.course_id
and activity.activity_id = marks.activity_id
```

Referring to the tabular results (Tables 4.2 to 4.9) in the next section, five attributes were measured. The column "connection" refers to the time required for

establishing a connection before the query is processed. Only one connection is required for multiple queries within the same database session. This attribute represents all preprocessing time required for querying the database. However, the amount of processing the approaches perform might vary according to the architecture of the underlying technologies. The columns "1<sup>st</sup> Q1" and "1<sup>st</sup> Q2" refer to the execution of query 1 and query 2 for the first time respectively, while "Sub. Q1" and "Sub. Q2" refer to subsequent executions of the corresponding queries. The times for the first and subsequent executions of the same query are separated out since significant differences in their values were observed in all cases. Although a general description of the experimental setup has been given, specifics of experimental issues regarding the individual approaches will be described in the following subsections.

#### **4.1.1 JDBC with Symantec dbANYWHERE**

The Symantec dbANYWHERE is a net-protocol all-Java JDBC driver, which has been described in Section 3.3.4. To test the performance of this type of driver, a fairly simple Java applet that makes JDBC calls for database connectivity was implemented and its database specific execution time was measured. In essence, the connection time and query time correspond to step 2 and steps 5 to 15 respectively of the corresponding algorithm described in Section 3.3.4. As mentioned earlier, Java applets using this type of JDBC driver can be dynamically downloaded from the Web server to the Web client. Therefore, no specific client installation is required and only a Java-enabled Web browser is needed to execute the implemented applets. This characteristic also implies that this type of JDBC driver is capable of implementing database-enabled Web applications across the Internet.

#### **4.1.2 JDBC with Intersolv JDBC/ODBC Bridge**

The Intersolv JDBC/ODBC Bridge is a joint development of JavaSoft and Intersolv which translates JDBC calls into ODBC calls as described in Section 3.3.1.

Again, a simple database-enabled Java applet was implemented in order to measure the performance of the bridge. In essence, the connection time and query time correspond to step 2 and steps 5 to 11 respectively of the corresponding algorithm described in Section 3.3.1. However, some ODBC binary code and database client code must be loaded on each client that uses the bridge and hence client pre-installation is inevitable. In particular, a special Netscape Navigator 3.0 plug-in that allows the bridge to be used within a locally loaded applet is required. Dynamically downloaded applets are not supported by the bridge. Therefore, the plug-in has to be pre-installed on the client machine and the applet must be loaded from the client's Java home directory. Due to the necessary client configuration required, the JDBC/ODBC Bridge is only appropriate to be used within corporate network but not over the Internet environment.

### 4.1.3 Microsoft Remote Data Object

As described in Section 3.4.1, Microsoft's Remote Data Object (RDO) is a COM object model providing access to remote data sources through ODBC. In fact, RDO is not primarily designed for Web database access and can be integrated with many desktop applications. In order to measure the performance of RDO based applications, a simple data-aware Java applet, which integrates with RDO, was implemented for the experiment. Again, the connection time and query time corresponding to step 2 and steps 5 to 10 respectively of the corresponding algorithm described in Section 3.4.1 were measured. However, since COM objects can access local system resources, which violates the Java security model, applets that use any COM libraries must be digitally signed for security purposes. Nevertheless, such applets are currently only supported by Microsoft Internet Explorer 3.0 or later. Similar to the bridge approach, an appropriate ODBC driver has to be present on the client machine. Using the cabinet [MC97c] technology, the ODBC driver can be dynamically downloaded from the Web server to the Web client together with any required RDO related classes. However, applets integrated with RDO only work within corporate networks due to the architecture of RDO as described in Section 3.4.1.

#### 4.1.4 Microsoft Advanced Data Connector

As described in Section 3.4.2, the Advanced Data Connector (ADC) is an efficient Web-based technology developed by Microsoft that brings database connectivity to the Internet environment. A simple HTML document with an embedded ADC ActiveX control was written in order to measure the performance of ADC. It turns out that the connection time and query time corresponding to steps 4 to 11 of the corresponding algorithm described in Section 3.4.2 cannot be separated as ADC is a stateless approach and its API only provides a single indivisible method for those steps. Although the ADC's API has a "connect" method, this method only updates the server's IP address and data source name attributes rather than actually establishing a database connection. Similar to COM integrated applets, ActiveX controls have to be digitally signed for security purposes and are currently only natively supported by Microsoft Internet Explorer. ActiveX controls can be dynamically downloaded across the Internet. Therefore, no special client installation is required for Web database access using ADC. Moreover, unlike Java applets, ActiveX controls will reside on the client machine permanently thereafter and hence no download is needed for subsequent browsing of the same Web page unless a newer version of the same control has been updated.

## 4.2 Summary of Experimental Results

The following tables summarize the results for Web access and querying of the databases by the various approaches. Base statistics including the number of measurements, minimums, maximums, and average measurements represent the summarized results. Refer to Appendix A for a complete listing of the experimental data.

Table 4.2: Querying MS Access 7.0 using Symantec dbANYWHERE

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	2700	3130	1270	5820	2850
Max. (ms)	3460	4280	1650	7140	3190
Avg. (ms)	3021.0	3684.0	1474.2	6272.0	3033.2

Table 4.3: Querying MS SQL Server 6.5 using Symantec dbANYWHERE

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	2910	3740	1650	6150	3240
Max. (ms)	3740	4120	2090	7970	3840
Avg. (ms)	3192.0	3924.0	1862.0	6778.0	3444.2

Table 4.4: Querying MS Access 7.0 using Intersolv JDBC/ODBC Bridge

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	12310	1050	440	1870	870
Max. (ms)	15110	1710	1210	2690	1540
Avg. (ms)	13064.0	1364.0	626.6	2180.0	1109.0

Table 4.5: Querying MS SQL Server 6.5 using Intersolv JDBC/ODBC Bridge

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	13120	880	390	1260	870
Max. (ms)	14510	990	990	1520	1490
Avg. (ms)	13841.0	918.0	660.2	1378.0	1151.4



Table 4.6: Querying MS Access 7.0 using MS RDO 2.0

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	5820	940	50	1420	210
Max. (ms)	7250	1160	170	1590	330
Avg. (ms)	6406.0	1030.0	70.0	1480.0	253.4

Table 4.7: Querying MS SQL Server 6.5 using MS RDO 2.0

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	5220	380	50	760	380
Max. (ms)	6090	550	110	900	440
Avg. (ms)	5548.0	482.0	66.2	828.0	405.8

Table 4.8: Querying MS Access 7.0 using MS ADC 1.0

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	50	1050	220	4500	2910
Max. (ms)	170	2370	330	6090	4180
Avg. (ms)	110.0	1366.0	263.2	5236.0	3211.0

Table 4.9: Querying MS SQL Server 6.5 using MS ADC 1.0

	connection	1 <sup>st</sup> Q1	Sub. Q1	1 <sup>st</sup> Q2	Sub. Q2
Count	10	5	50	5	50
Min. (ms)	50	1040	210	4610	3570
Max. (ms)	220	1750	440	6480	4780
Avg. (ms)	127.0	1352.0	274.4	5052.0	3780.8

### 4.3 Interpretation of Experimental Results

Based upon the entire set of experimental data given in the previous section and other observations, a comparison of the four configurations is now presented. The

significance of the experiment is to discover how the performance of Web database applications can be affected by the underlying technologies. Moreover, it is highly desirable to determine the reasons for performance discrepancies, if any. Therefore, explanations of any unusual experimental behavior have been attempted by performing other smaller and, perhaps, informal tests. A graphical representation of the experimental data helps illustrate the experimental data in a more meaningful way. The following two charts provide a comprehensive summary of the experimental results in a graphical manner.

**Average Access Time (MS Access 7.0)**

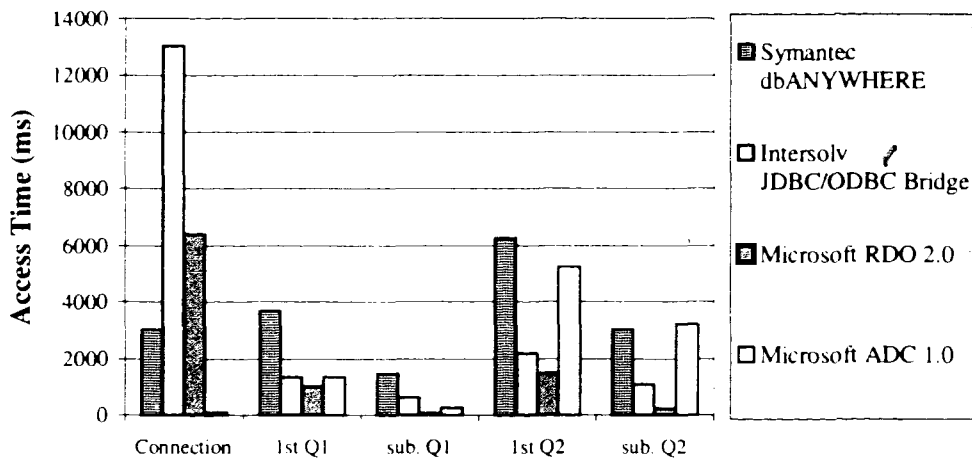


Figure 4.2: Average access time to MS Access 7.0

**Average Access Time (MS SQL Server 6.5)**

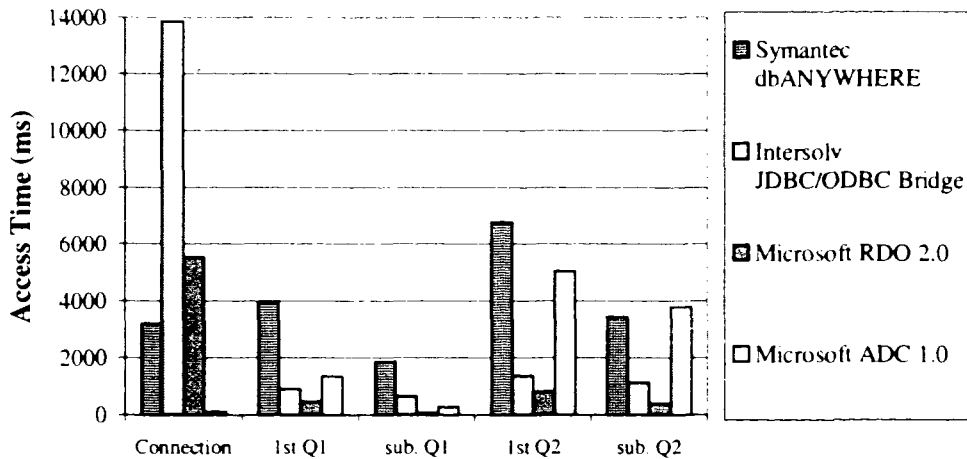


Figure 4.3: Average access time to MS SQL Server 6.5

It can be observed from the above charts that there is no significant performance difference among the four approaches operated with the two database management systems. Therefore, the following discussion will be neutral to the underlying DBMSs (keeping in mind that two of the approaches in the experiment only operate in internal networked environments while the other two approaches work across the Internet as well). Needless to say, the processing required for an Internet-based application is much more than that of an internal network based one. For example, a certain amount of processing time will be spent in resolving the server's IP address in an Internet-based application. Moreover, the data transmission and communication time is also much longer in an Internet-based application than an internal networked based application. Hence, a comparison would be fair only between approaches working under similar situations. In order words, comparing the two LAN-based approaches: JDBC using Intersolv JDBC/ODBC Bridge and Microsoft Remote Data Object, and comparing the two Internet-ready approaches: JDBC using Symantec dbANYWHERE and Microsoft Advanced Data Connector. The following subsections describe a few interesting observations of the experiment and comparisons between similar approaches.

### **4.3.1 Performance Difference in First and Subsequent Queries**

One of the most obvious observations is the significant difference in the average response time of the same query executing for the first time as opposed to subsequent times in all approaches. Perhaps this observation seems to be very unusual at the beginning since these queries perform exactly the same operation and thus should yield similar response times. However, on a second glance, this behavior is reasonable since a fair amount of time is spent retrieving the meta-data of the targeted database (tables names, columns names, and fields types). Moreover, certain amounts of processing time are possibly spent in other initialization processes such as resolving the IP address and verifying the targeted data source. For the ADC, it is reasonable that certain amounts of overhead in the first query will be expended in client caching as well. Undoubtedly,

these initialization processes are only required when the application accesses the database for the first time within the same session but not in any subsequent operations.

In order to examine this behavior further, a completely different and more complex query (one that retrieves data from different tables) was executed during the same session after the simpler query had been executed. The response time for the complex query was indeed similar to the response time of the same complex query executed multiple times in a different database session and surprisingly, even shorter than that of the simpler query executed for the first time. Another small test is to execute the same simple query for subsequent times with changed data in the database. The response time was also as fast as was usually experienced and updated data could also be successfully retrieved. These simple tests verify that subsequent queries do access the database more efficiently and extra processing needs to be done by any query executing for the first time in a session. Unfortunately, the detailed internal architectures and implementations of architecture components are not known and hence there is no simple way of further investigating the facts.

### **4.3.2 Intersolv JDBC/ODBC Bridge VS. Microsoft RDO**

Although both the Intersolv JDBC/ODBC Bridge approach and the Microsoft RDO approach only work in LAN-based environments, they are appropriate in certain situations. According to the experimental results, the JDBC/ODBC Bridge approach typically took a very long time to make a database connection. Moreover, it took about twice the time for RDO to finish a given query operation. Although the RDO based application was written in Java, many database-related function calls are based on the RDO engine which has been compiled into native code instead of Java byte code. Therefore, the RDO based applications can be expected to run faster than the ones that use the JDBC/ODBC Bridge. Moreover, all JDBC calls in the application that used the JDBC/ODBC Bridge needed to be translated into ODBC calls. This level of translation does consume considerable amounts of processing time.

Apart from its inefficient performance for database operations, the JDBC/ODBC Bridge requires a lot of installation work on each client. Certain software, including the ODBC driver, a small library for the bridge, and the compiled Java byte code, must be installed and configured on each system that will be using the bridge and it cannot be accomplished automatically. This task is quite undesirable not only from the standpoint of having the required components installed and configured properly, but the appropriate ODBC drivers and bridge libraries may not be readily available or may be very expensive. Considering the implied additional work and the slow performance, other alternatives should be considered rather than the JDBC/ODBC Bridge approach for developing Web database applications.

### 4.3.3 Symantec dbANYWHERE VS. Microsoft ADC

Both the ADC and dbANYWHERE configurations are Internet-based. Although the ADC is a stateless approach, whereas dbANYWHERE is a state-based one, the advanced client-caching mechanism of ADC makes its performance comparable to state-based approaches. It is observed that the processing time for the Microsoft ADC approach is much less than that of the Symantec dbANYWHERE approach in most measurements. One exception is the average response time for executing Q2 for both the first and subsequent times. Note that the efficiency of the stateless ADC architecture depends highly on the Advanced Data Virtual Table Manager (VTM) which contains buffers for metadata such as tables, rows, columns, and keys, as well as the actual table data itself. Since Q2 is a fairly complicated query and a fair amount of actual data will be retrieved, it is not feasible to cache the entire set of data. As a result, extra overhead is required in subsequent database operations and hence relatively large amount of time is required in this particular case. However, the difference is so small (< 10%) that it doesn't deserve special attention.

The average response time of the ADC approach is much shorter than that of the dbANYWHERE approach in all other measures. In particular, the average response time for subsequent execution of Q1 of the ADC approach is about one-sixth of that of the

dbANYWHERE approach. An obvious explanation is that the ADC approach runs in native code whereas the dbANYWHERE approach runs in byte code that must be interpreted. Moreover, ADC's sophisticated client-caching mechanism tremendously speeds up its execution regardless of its stateless nature. Furthermore, the dbANYWHERE uses the middleware approach for any database-related communication, which possibly slows down the communication process. Unfortunately, none of the log entries of the middleware is timestamped and hence there is no way to further determine the degree of performance distortion.

On the other hand, the average connection time of the ADC approach is much shorter than all other approaches, including the LAN-based ones. In most situations, it is common that a database application will resolve the server's IP address and verify the data source when establishing a connection. However, the connection time of the ADC based application is so small that it seems that nothing has been done at all. In order to further examine the work done for making connections by the various approaches, invalid HTTP DNS entries and data source names were intentionally supplied to the applications to observe the effect. All except the ADC approaches resulted in an error and could not continue to run. In contrast, the ADC based application continued to execute until a database query was submitted. The work associated with making the connection is delayed in the ADC approach until a query is executed at which time the error is finally experienced. In spite of the extra work to be done, the response time of a query executed for the first time in a database session was also comparable or much faster than that of the middleware approach according to the test results. Therefore, the ADC approach is indeed much more efficient than the JDBC middleware approach. Unless cross-platform compatibility is a primary issue, the ADC approach should be the way to go.

# Chapter 5

## Prototype Web Database Application

To further demonstrate the superiority of the newer Web database access approaches over traditional ones, a prototypical Web database application was developed using one of the emerging technologies. Three main areas were investigated: (i) the variety of user interface options available, (ii) the degree of interactivity between the user and the application, and (iii) the performance of remote database access. In this chapter, several interesting screen shots are presented to illustrate the user-friendliness and interactivity of the application while the results of a simple experiment will be used to evaluate its performance.

### 5.1 Evolution

The prototype was given the name *Personal Software Manager*, or PSM, which is a software engineering lab tool for managing the Personal Software Process<sup>1</sup> (PSP). The development of the application is initiated by Dr. Kal Toth<sup>2</sup>, who developed a PC Windows version of the tool. The intention of developing an equivalent Web-based

---

<sup>1</sup> Details of PSP can be found in Humphrey, Watts S. *A Discipline for Software Engineering*, Addison-Wesley, 1995

<sup>2</sup> Dr. Kal Toth is an Adjunct Professor in the School of Computing Science at Simon Fraser University

version is to allow students engaged in an advanced software engineering course at Simon Fraser University (SFU) to be able to access the tool through the World Wide Web, in conjunction with distantly delivery of courses using tools such as SFU's Virtual-U<sup>3</sup>. For students to be able to access the application on various platforms with a universal user interface, it was decided to use Java as the underlying technology. Since the objective of the PSM tool is to analyze and manage software engineering processes based on data gathered at different points in the software engineering process, a completely data-driven application had to be developed. Moreover, a highly interactive application is desired with an intuitive and easy-to-use user interface. These and several lesser considerations led to a product with not only the look and feel, but also the capability that is comparable to its Windows counterpart.

## **5.2 Design Issues**

Simply speaking, the PSM supports the entry, updating, deletion, and printing of data related to time, software defects, estimating, and planning of software development process. The PSM captures software engineering concepts that are outside the scope of this thesis. With this in mind, this section discusses general design issues regarding the quality of the user interface and the degree of interactivity of the application will be given. Our purpose is to evaluate the quality of applications might use Java with JDBC as the underlying programming model, from an end user point of view.

### **5.2.1 User Interface and Interactivity**

The user interface is undoubtedly a critical part of any application, including a Web-based one. In fact, user interface design for Web applications is even more challenging as it is constrained by the limited capability of HTML. Executable content approaches bring a variety of new user interface options to Web applications. Instead of adhering to the constraints imposed by the HTTP protocol with HTML forms, a graphical

---

<sup>3</sup> An overview and general information about Virtual-U can be found at <http://virtual-u.es.sfu.ca>



user interface emulating traditional Windows applications was created. After “logging in”, each user gets a customized view of the application based on the profile of the user. The interface is organized into a number of windows and the user is free to navigate among different windows within the same screen of the Web browser. The figure below is the top level screen of the application that displays all available operations linked together by arrows representing data flow within the application.

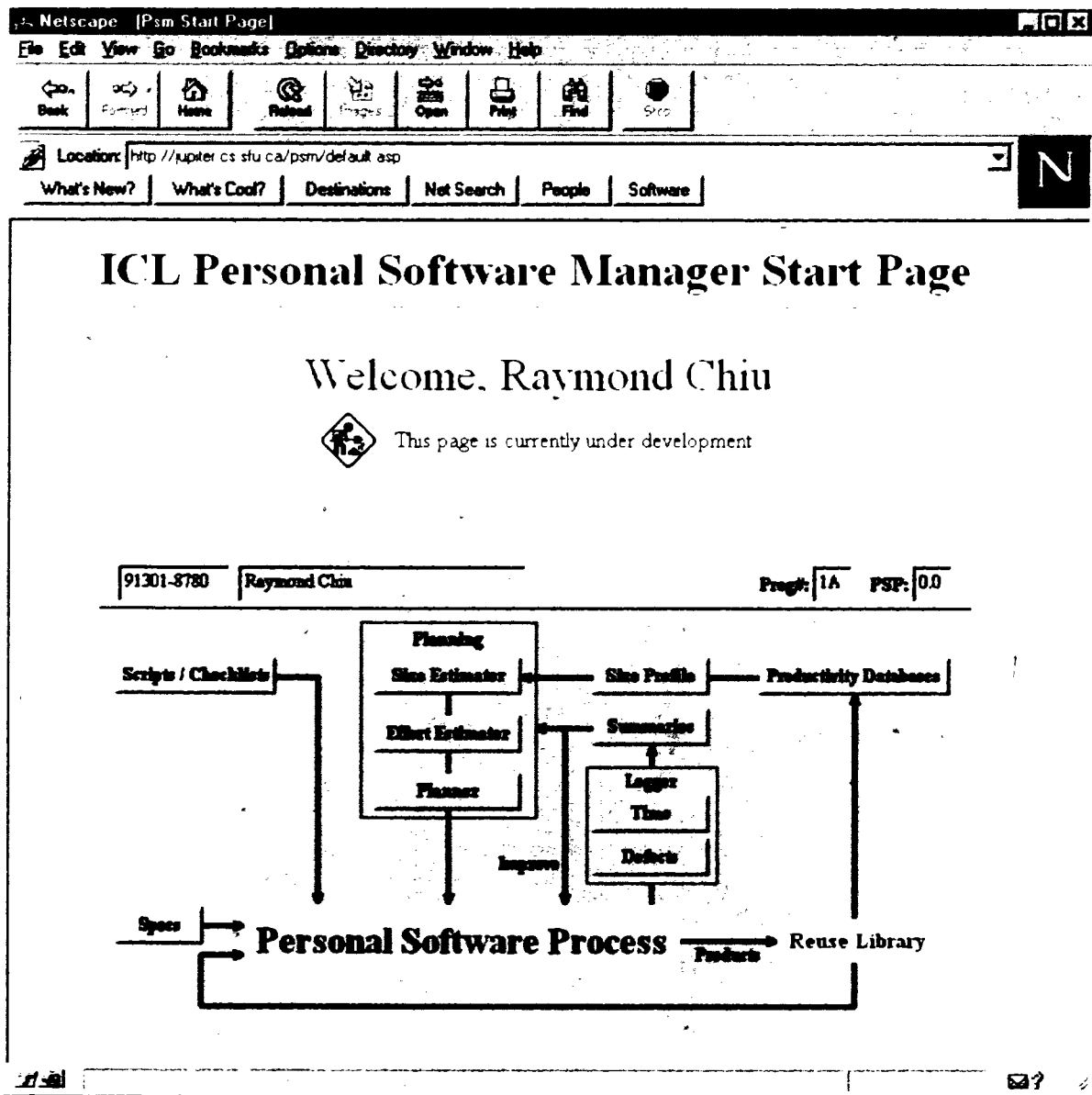
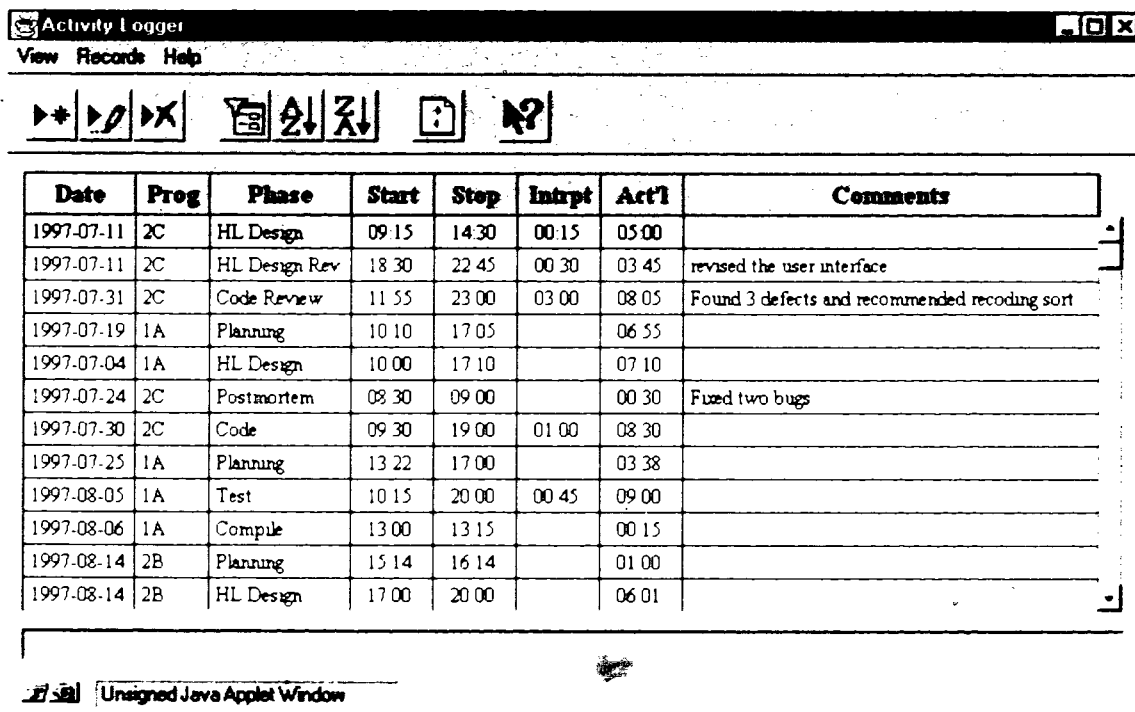


Figure 5.1: Top level screen

Although the prototypical application is not a complete implementation, sufficient functionality of the PSM was used to demonstrate the capability of the underlying technology. One such function is the activity logger, which logs all activities of the user throughout the software development process. The information kept by the activity logger can be used to derive useful statistics and analyses of software development activity. The figure below is the activity logger window being displayed after the “Time” button in the top level screen (Figure 5.1) was pressed. It displays a list of logged activities for the user.



The screenshot shows a window titled "Activity Logger" with a menu bar containing "View", "Records", and "Help". Below the menu bar is a toolbar with icons for navigation and editing. The main area contains a table with the following data:

Date	Prog	Phase	Start	Stop	Intrpt	Act'l	Comments
1997-07-11	2C	HL Design	09:15	14:30	00:15	05:00	
1997-07-11	2C	HL Design Rev	18:30	22:45	00:30	03:45	revised the user interface
1997-07-31	2C	Code Review	11:55	23:00	03:00	08:05	Found 3 defects and recommended recoding sort
1997-07-19	1A	Planning	10:10	17:05		06:55	
1997-07-04	1A	HL Design	10:00	17:10		07:10	
1997-07-24	2C	Postmortem	08:30	09:00		00:30	Fixed two bugs
1997-07-30	2C	Code	09:30	19:00	01:00	08:30	
1997-07-25	1A	Planning	13:22	17:00		03:38	
1997-08-05	1A	Test	10:15	20:00	00:45	09:00	
1997-08-06	1A	Compile	13:00	13:15		00:15	
1997-08-14	2B	Planning	15:14	16:14		01:00	
1997-08-14	2B	HL Design	17:00	20:00		06:01	

The window title bar also includes "Unsigned Java Applet Window".

Figure 5.2: Activity Logger window

As the height of the Windows-style component containing the records might not be long enough to hold all the logged activities, the records' container was implemented to feature automatic vertical scrolling. A vertical scroll bar appears automatically when needed, which provides a mean for the user to scroll through the whole list of activities. A number of operations such as editing, filtering, and sorting can be performed against the activities. While all the available operations are listed in the menu bar, a number of more popular operations appear in the tool bar as well for quick access. These operations

can either be triggered by selecting the appropriate menu item or clicking the corresponding image button in the tool bar. When the mouse pointer is over a particular button, a short description about the use of the button is displayed in the status bar at the bottom of the window to help the user. The following figure shows the expanded views of the menu bar, one for each corresponding menu.

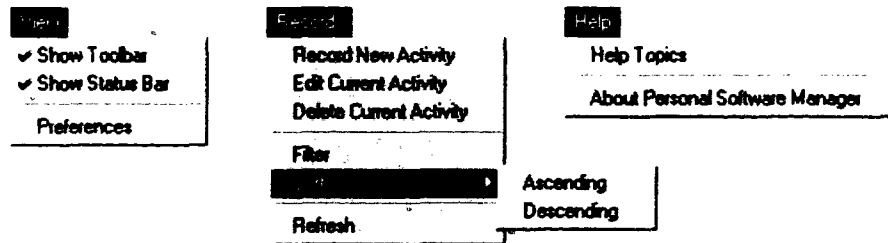


Figure 5.3: Expanded menu bar

Users can opt to show or hide the tool bar or/and status bar by toggling the corresponding menu item on and off such that components within the window will be positioned and the window will be resized accordingly. Typical operations of the activity logger include creating a new record, editing an existing record, and deleting an existing record. As an illustration, a tour of editing an existing record is presented. A user can either select the "Edit Current Activity" item from the "Records" menu or click the corresponding button to edit the currently highlighted activity, or double click a record in the activity listing to edit the desired activity. An edit activity window filled with the information of the corresponding activity will then be displayed as follows.

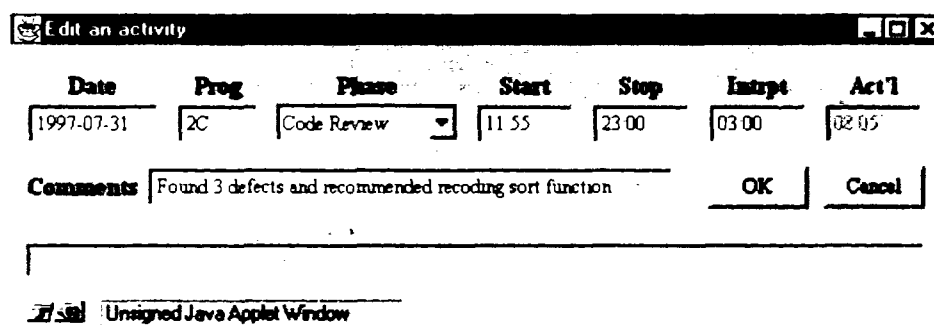


Figure 5.4: Edit activity window

Suppose the user has finished editing the activity and presses the "Ok" button to submit the information. A verification process will then be invoked to validate all the

inputs. In case of erroneous input, a modal dialog window with one or more appropriate error messages similar to the following one will be displayed. The user must close the dialog window first before proceeding to any other activity.

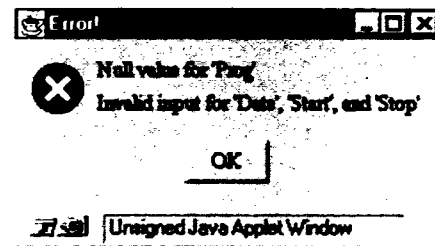


Figure 5.5: Dialog window displaying error messages

Otherwise, a modal dialog window similar to the one below will be displayed to confirm that all the inputs have been entered correctly.

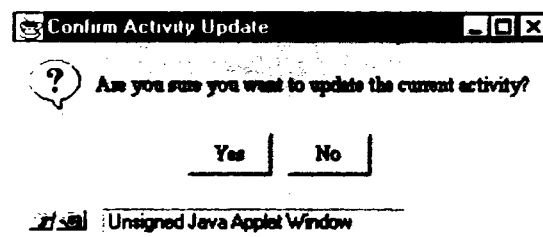


Figure 5.6: Dialog window confirming activity update

Note that, in contrast to traditional Web technologies where most of the interaction occurs across the client-server connection, all interactivity described above took place solely on the client side. As mentioned earlier, this kind of true client interaction has the advantage of reducing the heavy traffic of the Internet, by utilizing the computing power of the host computer. Suppose now the user really decides to update the activity by pressing the “Yes” button in the dialog window. The application will then send the appropriate query and possibly other information to the server. The remote database in the server end together with the listing of the activity logger window will then be updated to reflect the change.

In fact, several other operations apart from query operations can be performed against the records. Typical examples include filtering and sorting of the activities based on certain predefined preferences. The following description shows the interactivity

involved in record filtering. The user starts the operation by entering the preferences of the operation – selecting the “Preferences” menu item in the “View” menu. A window like the following one then appears.

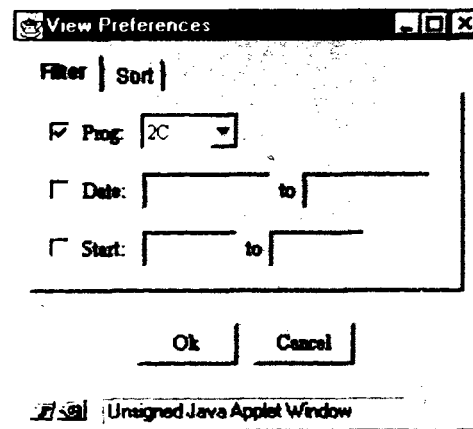


Figure 5.7: View Preferences window

As shown in the above figure, two tabs, each associated with its independent set of visual components, appear. This form of component layout has the advantage of conserving window space and arranging related components and information in a structured and organized manner. The appearance of the corresponding “Sort” tab is shown in the following figure.

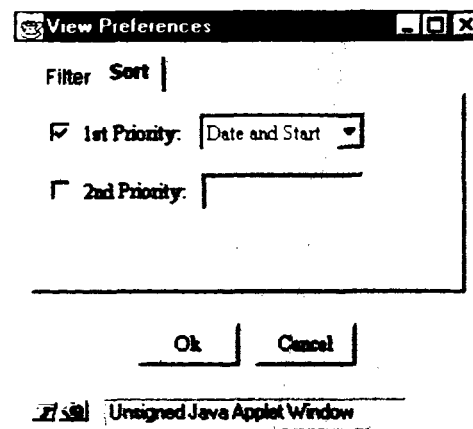
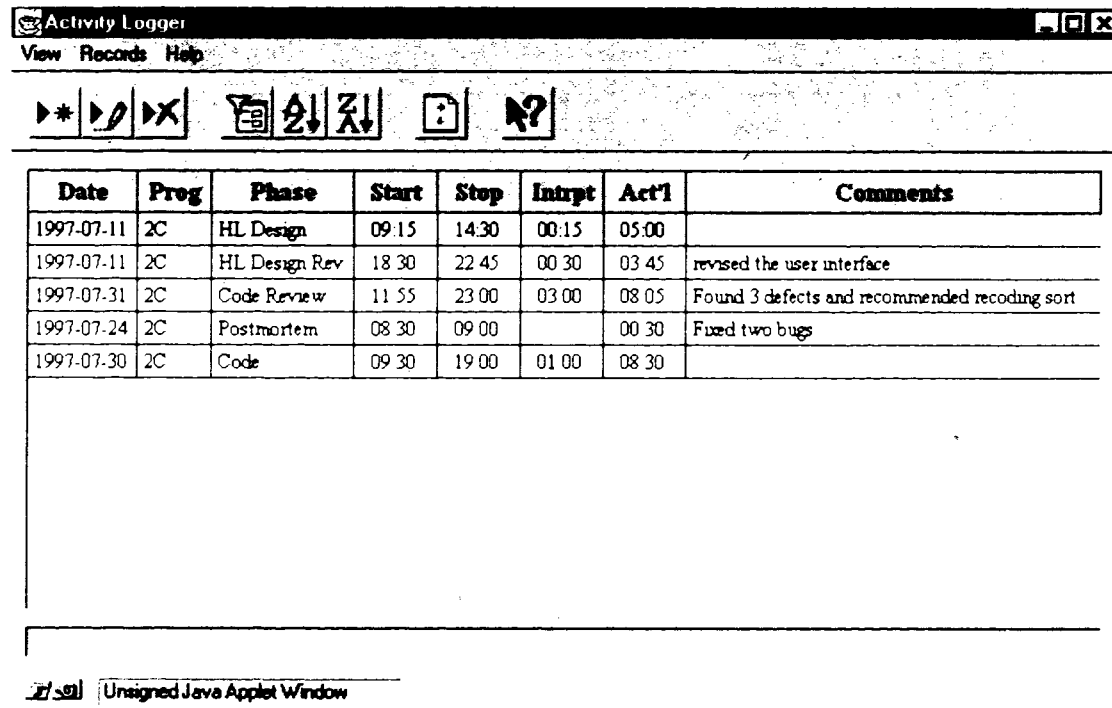


Figure 5.8: View Preferences window with corresponding Sort tab

The user supplies the preference information by checking the corresponding check box and choosing or entering the relevant information. Suppose the information such as that shown in Figure 5.7 was supplied and the “Ok” button was pressed for any

changes to take effect. Whenever the user selects the “Filter” item from the “Records” menu or clicks the corresponding button from the tool bar, the records are filtered based on the predefined preferences and the result is reflected in the activity listing. The following figure shows the result of applying the filter operation. Note the difference between this listing and that of Figure 5.2.



The screenshot shows a window titled "Activity Logger" with a menu bar containing "View", "Records", and "Help". Below the menu bar is a toolbar with several icons: a list, a filter, a refresh, a print, and a help icon. The main area contains a table with the following data:

Date	Prog	Phase	Start	Stop	Intrpt	Act'l	Comments
1997-07-11	2C	HL Design	09:15	14:30	00:15	05:00	
1997-07-11	2C	HL Design Rev	18:30	22:45	00:30	03:45	revised the user interface
1997-07-31	2C	Code Review	11:55	23:00	03:00	08:05	Found 3 defects and recommended recoding sort
1997-07-24	2C	Postmortem	08:30	09:00		00:30	Fixed two bugs
1997-07-30	2C	Code	09:30	19:00	01:00	08:30	

The window title bar at the bottom indicates it is an "Unsigned Java Applet Window".

Figure 5.9: Activity Logger window after filtering

Note that the scroll bar originally appeared in Figure 5.2 now disappears, since all records can be displayed in the window and there is no need to scroll through them. Although the operation of filtering seems non-trivial, it is solely performed at the client machine and there is no need for any client-server interaction. The illustration given in this section shows some of the available graphical user interface options together with the degree of interactivity between the user and this prototypical application. In fact, it is exactly these advantages make executable content approaches so powerful and widely adopted in the Web community.

## 5.3 Implementation

The PSM uses Java with the JDBC API instead of other executable content approaches and works with most clients and servers. The entire application is written using JDK 1.02 API for Java making it work well with most popular Web browsers. Moreover, a net-protocol all-Java JDBC driver in the form of middleware for database connectivity is being used. This implementation makes dynamically downloadable applets possible and allows a true three-tier network database architecture to be configured with little modification required on any server system.

## 5.4 Performance Issues

It should be obvious from previous sections that a much more flexible user interface design is made possible by using executable content approaches rather than traditional Web publishing technologies. Moreover, the degree of interactivity between users and the newer technology-based applications is much higher than traditional CGI or API-based Web applications because of the real-time client-side interaction achieved by executing operations locally as much as possible.

As mentioned in earlier chapters, traditional and the emerging on-line approaches not only differ in their capability, but also in their efficiency in handling multiple operations within the same session. In order to investigate this performance issue, an experiment was performed among three approaches. They include: 1) the Microsoft Internet Database Connector (IDC, section 2.4.1), which is an easy-to-use traditional Internet database gateway built into the Microsoft Internet Information Server, 2) the Microsoft Advanced Data Connector (ADC, section 3.4.2), which is an executable content approach based on the ActiveX technology, and 3) Java applet with the Symantec dbANYWHERE (section 3.3.4), which is also an executable content approach adhering to the JDBC standard for Java. The JDBC/ODBC Bridge and RDO approaches are not being tested as they only operate on LAN-based rather than in Internet environments. The following subsections briefly describe the experiment and its findings.

### 5.4.1 Session-oriented Experiment

Several single-operation experiments were performed (described in Chapter 4) to compare the performance of a few on-line Web database access technologies. In this chapter, the result of an experiment for executing multiple database operations within the same application session is presented. As each of the three approaches under our test handles multiple operations very differently, significant performance differences are expected. The objective of the experiment is thus to compare the efficiency of three different approaches under a session-oriented environment.

The hardware setup of the experiment was exactly the same as that described in Section 4.1 and the tests were performed against the Microsoft Access 7.0 database management system. A typical Web database application was implemented for each of the three approaches. The application allows a user to search for student information based on the first letter of the student's last name. Multiple operations can be performed sequentially during the same application session. For each operation, the user starts by entering the first letter of the students' last name. A query based on the user's input is generated and submitted to the server for execution when the "submit" button is pressed. Assume *input* is the first letter of the last name being entered for the search. A query like:

```
select * from account_info where last_name like 'input%'
```

would extract the desired information from an appropriate table in the database. Refer to Section 4.2 for more details of the underlying database and associated tables. When the server finishes executing the query, the results are returned to the application for display. Individual time spent in executing the query and displaying the results for each operation were measured. The results and interpretation of the experiment are described in the following section.

### 5.4.2 Result and Interpretation

The following tables summarize the results of the experiment. Five query operations were performed sequentially within the same application session. Since the objective of the



experiment was to find out whether there is any significant difference performing multiple query operations in applications implemented in various Web database approaches, both the average individual as well as average cumulative execution times for the five tests were recorded. Refer to Appendix B for a complete listing of the experimental data.

Table 5.1: Querying MS Access 7.0 using MS IDC

Query #	1	2	3	4	5
Avg. Individual Time (ms)	1642	1154	1030	1342	1096
Avg. Cumulative Time (ms)	1642	2796	3826	5168	6264

Table 5.2: Querying MS Access 7.0 using MS ADC 1.0

Query #	1	2	3	4	5
Avg. Individual Time (ms)	1728	540	506	560	496
Avg. Cumulative Time (ms)	1728	2268	2774	3334	3830

Table 5.3: Querying MS Access 7.0 using Symantec dbANYWHERE

Query #	1	2	3	4	5
Avg. Individual Time (ms)	5754	682	836	1066	844
Avg. Cumulative Time (ms)	5754	6436	7272	8338	9182

The following column and line charts provide additional appreciation for the experimental data.

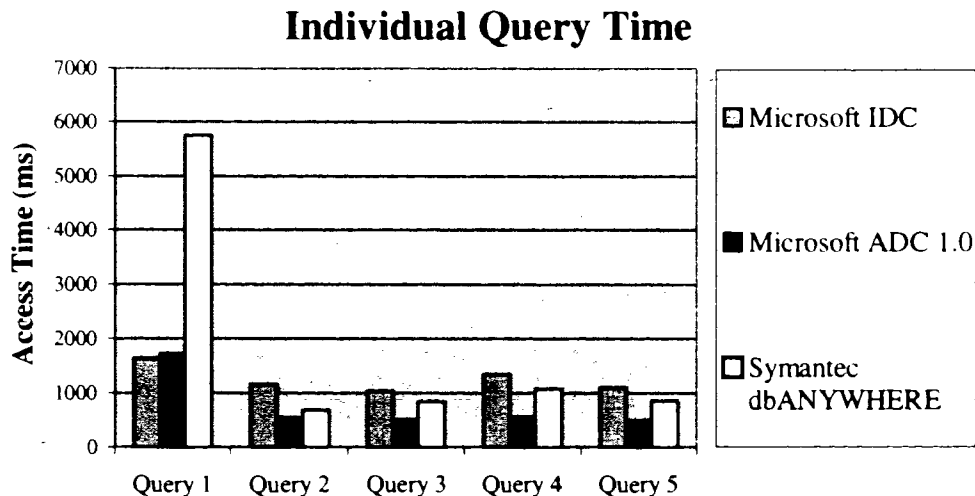


Figure 5.10: Average individual query access time

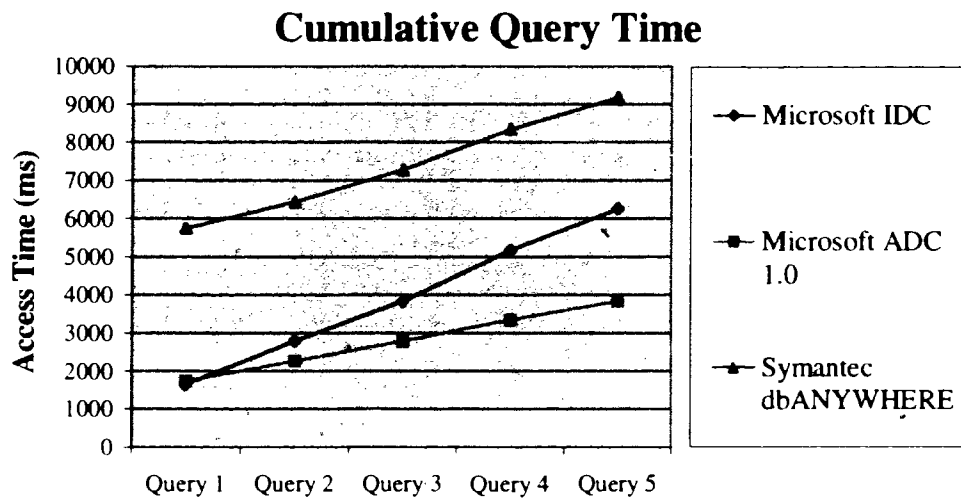


Figure 5.11: Average cumulative query access time

It can be observed from the summary of the experimental data and the column chart that there is no significant difference between the average individual time of Query 1 and subsequent queries in the results for MS IDC. On the other hand, very large differences can be observed in the results of the MS ADC and the JDBC queries using Symantec dbANYWHERE, especially in the JDBC approach. This is, in fact, a reasonable observation since the MS IDC is a completely stateless approach. As a result, subsequent query operations in applications developed under MS IDC do not take advantage of previous ones. The MS ADC approach saves some processing time in subsequent query operations since information retrieved from the first query operation will be cached using a sophisticated client caching mechanism in order to keep the time of any subsequent query operations to a minimum. As mentioned earlier, JDBC using Symantec dbANYWHERE is a completely state-based approach with the connection kept open once the connection has been established for the first query operation. Hence, tremendous performance gains result for subsequent query operations.

Perhaps more information can be concluded from the linear regression models of different sets of the experimental data. Therefore, a simple linear regression analysis for the cumulative query execution time against the number of executed queries was carried out to further analyze the experimental data. However, similar execution times for each query need to be assumed in order to come up with a meaningful analytical result.

Although this restriction is not very realistic in most database applications, the data gathered in our experiment indeed satisfies this requirement and is useful in determining certain characteristics of the various approaches. Results of the three regressions are given below. Note that the dependent variable  $y$  represents the cumulative query execution time while the independent variable  $x$  represents the number of queries to be performed.

Microsoft IDC:  $y = 454.4 + 1161.6x$

Microsoft ADC:  $y = 1205.8 + 527x$

JDBC (Symantec dbANYWHERE):  $y = 4769 + 875.8x$

In each of the regression equations, the  $y$ -intercept represents the overhead (e.g. for initialization or establishing the connection) required in the first query operation while the  $slope$  represents the actual time required for each query operation, including the first one. The  $y$ -intercept in the regression equation of the MS IDC is the smallest, and that of the MS ADC is the next, while that of the JDBC approach is the largest. This result agrees with the nature of each approach and the observations made in the previous discussion. The overhead (judging from the  $y$ -intercept of the corresponding regression equation) of the JDBC approach is surprisingly large probably due to the immaturity of the TCP/IP protocol used for JDBC connection. Another possible explanation is that much more processing is involved in a state-based protocol than a stateless one for storing vital information of the connection and this form of processing has to be performed in the slower Java byte-code format.

On the other hand, the slope in the regression equation of the MS IDC is the largest since nearly the same processing time is required in all query operations due to the stateless nature of the MS IDC and not much benefit can be gained from previous operations. Although the actual time spent in each query in the JDBC approach is less than that of the MS IDC (judging from the slope of the regression equations), it is in fact more than that of the MS ADC. A possible reason is that the time spent in communicating with the remote server and displaying the query results executed as Java intermediate byte-code is longer than that executed as native code in the MS ADC.

Moreover, it is possible that the client caching mechanism employed by the MS ADC is as efficient as a state-based database access technology.

It is clear from the above discussion that very large performance gains can result from on-line Web database access technologies through the use of either a persistent database connection or sophisticated client caching mechanism. Although considerable overhead is unavoidable in the first query operation, the performance of on-line approaches do outperform traditional ones and are recommended for Web database access in most situations. This observation is particularly true in applications that are expected to have long database sessions with a huge number of query requests.

## **Chapter 6**

# **Summary, Conclusion, and Future Work**

Advanced features and associated technologies for global information retrieval are being continuously developed to provide better ways of implementing sophisticated Web-based applications that link to live data. This thesis has presented the characteristics of both traditional (e.g. CGI, proprietary APIs) and emerging Web database access technologies (e.g. Java applets with JDBC and Microsoft ADC). While the challenges and ambitions of developing Web database applications have been identified throughout the thesis, it is clear that there is still much work to be done before Web-based data-driven applications become comparable to traditional client/server database applications in terms of quality, robustness, and scalability. It is hoped that this thesis will serve as a basis for further investigation or study into the development of better Web database systems. This chapter will summarize the important points that have been addressed throughout the thesis and suggest some ideas for possible future work.

### **6.1 Summary**

Different from traditional ones, the newer Web database access technologies feature a wide range of effective and efficient capabilities. The major strength of the newer approaches is that they promote a high degree of interaction between the user and the underlying applications through the use of local executable content and persistence database service. This leads to the employment of more flexible design mechanisms in various aspects of the application. This design flexibility has been fully demonstrated by a prototypical application. The approach also has a performance advantage achieved by the use of either a state-oriented database protocol in the case of Java with JDBC, or client-side caching in the case of the Microsoft ADC. With the ever-growing resources available in global information systems, the use of powerful Web publishing techniques and efficient database access technologies becomes an essential part in the development of sophisticated Web database application. The following advantages for Web database access are generally characterized by the recent approach.

1. **High-level and intuitive user interfaces:** The technologies provide a wide range of graphical user interface possibilities that are not possible using only HTML and gateway programming. The availability of these interface options also gives developers an opportunity to construct intuitive user interfaces for Web applications which are similar to those of the familiar Windows environment.
2. **High degree of interactivity:** Highly interactive Web applications can now be developed since local execution in the Web client is made possible. Moreover, the fact that most activities can be confined to local sites implies reductions of network bandwidth consumption, better resource utilization, and quick response to user's requests.
3. **Performance enhancement:** State or session information no longer needs to be managed by programmers. Instead, the newer on-line approaches have the ability to handle these situations through the use of either a sophisticated built-in client caching mechanism or a state-based database connection. These kinds of handlers can reduce or even eliminate the startup cost for multiple requests by the same client – hence, long database sessions can be processed efficiently.

4. **High capability:** The new advanced Web programming technologies are designed specifically to extend the Web in many ways. Although it is sometimes possible to use certain traditional approaches to achieve similar effects, the functionality of traditional approaches will not be as powerful and flexible as that of the newer ones, especially for complex tasks.

On the other hand, it is important to note that certain effects and tradeoffs result from the use of these newer technologies:

1. **Additional computing resources:** As some of the tasks will be shifted from the server to the client sites, extra resources such as disk space, memory, and CPU speed are required in the client machines. However, this seems not to be such a great issue with the low cost of today's computer hardware.
2. **Interoperability:** Since the technologies under discussion are still emerging, they might not be well supported in certain Web browsers, Web servers, and operating environments. The maturity and popularity of these technologies will, however, ensure their acceptance in the Internet community.
3. **Security:** Some users might disable executable content due to the possibility of security holes and the potential risk of harming the user's file system. Moreover, some such approaches might not be able to pass through certain corporate firewalls. However, the specification of most techniques will be continuously improved, standardized, and adapted to the construction of Web-based systems and such restrictions may not need to be employed as rigorously in the future.
4. **Long download time:** Considerably long download time is necessary for distributing fairly complicated executable content. The improvement in compression and versioning techniques, as well as the popularity of higher bandwidth access lines through ISDN and cable modems, will help in mitigating this issue to a certain extent.

## 6.2 Conclusion

The architecture of various Web database access techniques has been briefly described and relevant experiments have also been performed to compare their performance. However, the adoption of a particular technology is not only based on the capability and efficiency of that technology, but also highly depends on the particular situation of usage. Issues such as the target users, server setup, and the nature of information also play important roles in coming up with an appropriate conclusion. Being the focus of the thesis, the effectiveness of user interface and interactivity, as well as the performance of remote database access are the three main issues in evaluating Web database access technologies. With the use of our prototypical Web application, the flexibility and effectiveness of two of the emerging techniques, Java and ActiveX approaches, have already been demonstrated. It is clear that these emerging approaches are capable of developing Web applications with sophisticated graphical user interface and effective client-side interaction. As a result, these technologies are very suitable in developing applications for which effective user interface and interactivity are desired.

Regarding performance issues, the performance of any Web database access technology indeed varies greatly depending on the combination of the Web server, the operating system, and the server hardware being used. According to our experiments, the Microsoft Advanced Data Connector (ADC) running on Microsoft-based operating environment and Intel-based hardware turned in best performance. This observation is true in most performance scores of both the single-operation and multiple-operations experiment. It means that the ADC approach is very efficient in both regular and long database sessions. However, it is also important to note that when reviewing our experimental results, keep in mind that they reflect our specific experimental conditions and are not comparable to the results of other tests with different experimental settings.

### **6.3 Future Work**

This thesis presents the general consideration for developing interactive Web database applications and shows that huge differences can be observed by deploying various techniques. In fact, many of the technologies discussed are very new. For



example, the JDBC specification was only released to the public within the last year. Microsoft ADC is an even more recently available technology which was made available to system developers only in the first half of this year. It will be possible to perform additional studies when these technologies become more mature. For example, it would be very useful to discover exactly which components in the overall architecture consume extra overhead. Such investigation and experimentation can help developers deploy the most effective technology among various alternatives and help researchers improve the performance of technologies by making appropriate revisions to reduce overheads. The following is a list of suggestions for possible future work in this area.

1. **Multiple-user experiments:** Several single-user experiments were performed and their results have been presented. However, it is possible that the performance of certain technologies depends on the load of the server since the communication protocol and supporting components of a particular technology might be optimized for light or heavy loads, or a compromise between the two. Therefore, it will be useful to test the effectiveness of different approaches with varying numbers of clients.
2. **Other performance measures:** In fact, issues such as optimization, availability, and resource allocation are important in evaluating database-related tools. However, our experiments were only limited to the evaluation of transaction response time. In order to further investigate the efficiency of various technologies, research can be carried out to study the effect of Web database access using different technologies on other performance measures including transaction availability and system cost.
3. **Custom-built technology:** Once the benefits and tradeoffs of various technology approaches have been identified, there is no reason why a completely new Web database access configuration cannot be developed. Such a new approach should possess the advantages of various techniques and be well suited for general use. Moreover, the newly developed approach should adhere to existing de facto and open standards such as Java with JDBC, the ActiveX component model, or other workable specifications in order to increase usability.

# Appendix

## A Listing of Experimental Results: Chapter 4

This appendix contains a complete listing of all experimental data obtained from the experiments described in chapter 4.

Table A.1: Query 1 on MS Access 7.0 using Symantec dbANYWHERE

Test #	1	2	3	4	5
Connection (ms)	3460	3180	2700	2800	2800
Execution 1 (ms)	3860	4280	3130	3630	3520
Execution 2 (ms)	1270	1310	1650	1380	1420
Execution 3 (ms)	1490	1430	1490	1540	1650
Execution 4 (ms)	1430	1430	1480	1480	1490
Execution 5 (ms)	1370	1430	1430	1540	1430
Execution 6 (ms)	1370	1420	1480	1590	1540
Execution 7 (ms)	1420	1480	1540	1480	1430
Execution 8 (ms)	1490	1370	1420	1650	1420
Execution 9 (ms)	1370	1430	1370	1540	1540
Execution 10 (ms)	1430	1430	1540	1640	1430
Execution 11 (ms)	1540	1650	1540	1590	1430

Table A.2: Query 2 on MS Access 7.0 using Symantec dbANYWHERE

Test #	1	2	3	4	5
Connection (ms)	2850	2910	3300	3350	2860
Execution 1 (ms)	7140	5820	6320	6260	5820
Execution 2 (ms)	3080	2910	3020	2970	3130
Execution 3 (ms)	3020	3020	3070	2850	3020
Execution 4 (ms)	3020	3130	3020	2970	3020
Execution 5 (ms)	3070	3070	3020	3130	3130
Execution 6 (ms)	2960	2960	2910	2970	3020
Execution 7 (ms)	3130	3020	3070	2970	3020
Execution 8 (ms)	3190	3180	3180	3070	3130
Execution 9 (ms)	2910	2960	3080	2970	3190
Execution 10 (ms)	3070	3080	2970	2910	3070
Execution 11 (ms)	2970	3020	3020	3080	2910

Table A.3: Query 1 on MS SQL Server 6.5 using Symantec dbANYWHERE

Test #	1	2	3	4	5
Connection (ms)	3130	3020	3130	3080	3080
Execution 1 (ms)	3790	4120	3740	3960	4010
Execution 2 (ms)	1650	1700	1710	2090	1870
Execution 3 (ms)	1820	1870	1760	1930	1760
Execution 4 (ms)	1760	1810	1970	1930	1980
Execution 5 (ms)	1810	1810	1870	1760	1810
Execution 6 (ms)	1920	1810	1820	1870	2040
Execution 7 (ms)	1930	1760	1810	1810	1920
Execution 8 (ms)	1930	1920	1920	2030	2030
Execution 9 (ms)	2040	1760	1820	1760	1920
Execution 10 (ms)	1860	1810	1920	1870	1920
Execution 11 (ms)	1820	1920	1810	1810	1870

Table A.4: Query 2 on MS SQL Server 6.5 using Symantec dbANYWHERE

Test #	1	2	3	4	5
Connection (ms)	3740	3080	3620	3130	2910
Execution 1 (ms)	7970	6150	6420	6590	6760
Execution 2 (ms)	3350	3350	3350	3300	3620
Execution 3 (ms)	3290	3350	3400	3240	3570
Execution 4 (ms)	3300	3460	3410	3350	3620
Execution 5 (ms)	3410	3520	3510	3460	3740
Execution 6 (ms)	3300	3290	3400	3620	3790
Execution 7 (ms)	3240	3300	3570	3290	3520
Execution 8 (ms)	3240	3410	3620	3400	3840
Execution 9 (ms)	3350	3570	3460	3410	3620
Execution 10 (ms)	3240	3350	3510	3400	3460
Execution 11 (ms)	3350	3520	3460	3460	3620

Table A.5: Query 1 on MS Access 7.0 using Intersolv JDBC/ODBC Bridge

Test #	1	2	3	4	5
Connection (ms)	15110	12910	13900	12410	12310
Execution 1 (ms)	1050	1370	1430	1260	1710
Execution 2 (ms)	550	490	550	550	600
Execution 3 (ms)	610	600	660	660	610
Execution 4 (ms)	820	770	710	720	820
Execution 5 (ms)	770	600	1210	600	600
Execution 6 (ms)	770	720	770	660	610
Execution 7 (ms)	710	770	940	660	660
Execution 8 (ms)	500	490	490	440	500
Execution 9 (ms)	600	490	550	490	500
Execution 10 (ms)	550	550	500	500	500
Execution 11 (ms)	660	550	550	550	600

Table A.6: Query 2 on MS Access 7.0 using Intersolv JDBC/ODBC Bridge

Test #	1	2	3	4	5
Connection (ms)	12360	13780	12530	12800	12530
Execution 1 (ms)	2690	1870	2230	2030	2080
Execution 2 (ms)	1040	880	930	940	880
Execution 3 (ms)	1490	1040	1540	1430	1480
Execution 4 (ms)	880	870	940	880	880
Execution 5 (ms)	1040	1050	990	930	940
Execution 6 (ms)	990	990	1050	1040	1100
Execution 7 (ms)	1040	1050	1040	1050	1050
Execution 8 (ms)	1150	1100	1150	1160	1150
Execution 9 (ms)	1270	1310	1150	1210	1150
Execution 10 (ms)	1160	1210	1150	1270	1210
Execution 11 (ms)	1210	1200	1210	1320	1260

Table A.7: Query 1 on MS SQL Server 6.5 using Intersolv JDBC/ODBC Bridge

Test #	1	2	3	4	5
Connection (ms)	14280	14010	14510	13950	13670
Execution 1 (ms)	880	900	940	880	990
Execution 2 (ms)	660	720	940	660	820
Execution 3 (ms)	440	710	830	550	990
Execution 4 (ms)	390	710	830	660	660
Execution 5 (ms)	440	770	650	490	660
Execution 6 (ms)	550	550	550	490	770
Execution 7 (ms)	660	830	610	550	610
Execution 8 (ms)	610	660	770	550	660
Execution 9 (ms)	610	660	600	550	660
Execution 10 (ms)	710	710	720	600	550
Execution 11 (ms)	610	770	770	720	770

Table A.8: Query 2 on MS SQL Server 6.5 using Intersolv JDBC/ODBC Bridge

Test #	1	2	3	4	5
Connection (ms)	13840	13120	13240	14060	13730
Execution 1 (ms)	1380	1520	1260	1460	1430
Execution 2 (ms)	940	990	930	990	870
Execution 3 (ms)	940	1050	940	1210	880
Execution 4 (ms)	930	1040	930	990	880
Execution 5 (ms)	990	990	980	1100	940
Execution 6 (ms)	1040	1050	1200	1480	1150
Execution 7 (ms)	1160	1210	1210	1270	1270
Execution 8 (ms)	1150	1100	1160	1260	1260
Execution 9 (ms)	1210	1210	1270	1480	1370
Execution 10 (ms)	1320	1260	1420	1480	1210
Execution 11 (ms)	1480	1260	1370	1490	1260

Table A.9: Query 1 on MS Access 7.0 using MS RDO 2.0

Test #	1	2	3	4	5
Connection (ms)	7040	6310	6260	7250	6100
Execution 1 (ms)	940	990	1160	1020	1040
Execution 2 (ms)	110	50	160	60	110
Execution 3 (ms)	60	60	110	110	50
Execution 4 (ms)	50	60	50	170	160
Execution 5 (ms)	50	60	50	50	50
Execution 6 (ms)	110	50	60	50	60
Execution 7 (ms)	50	50	60	60	50
Execution 8 (ms)	60	50	60	60	50
Execution 9 (ms)	60	60	60	50	60
Execution 10 (ms)	60	50	110	110	50
Execution 11 (ms)	50	60	110	50	50

Table A.10: Query 2 on MS Access 7.0 using MS RDO 2.0

Test #	1	2	3	4	5
Connection (ms)	6060	5820	6200	6480	6540
Execution 1 (ms)	1430	1540	1590	1420	1420
Execution 2 (ms)	330	220	220	270	280
Execution 3 (ms)	270	280	220	270	220
Execution 4 (ms)	280	280	220	270	270
Execution 5 (ms)	220	210	280	220	270
Execution 6 (ms)	270	280	220	280	220
Execution 7 (ms)	220	270	220	220	270
Execution 8 (ms)	220	280	280	280	220
Execution 9 (ms)	220	270	270	270	320
Execution 10 (ms)	270	220	220	270	280
Execution 11 (ms)	220	280	220	270	220

Table A.11: Query 1 on MS SQL Server 6.5 using MS RDO 2.0

Test #	1	2	3	4	5
Connection (ms)	6090	5220	5270	5220	5660
Execution 1 (ms)	490	380	490	500	550
Execution 2 (ms)	110	60	50	50	60
Execution 3 (ms)	50	60	50	50	60
Execution 4 (ms)	60	60	110	50	50
Execution 5 (ms)	60	60	60	60	60
Execution 6 (ms)	60	110	60	110	110
Execution 7 (ms)	60	110	60	50	60
Execution 8 (ms)	50	60	60	50	60
Execution 9 (ms)	50	60	60	50	50
Execution 10 (ms)	60	110	60	60	110
Execution 11 (ms)	60	60	50	110	60

Table A.12: Query 2 on MS SQL Server 6.5 using MS RDO 2.0

Test #	1	2	3	4	5
Connection (ms)	5990	5600	5220	5820	5390
Execution 1 (ms)	880	770	830	900	760
Execution 2 (ms)	390	440	440	390	390
Execution 3 (ms)	380	440	390	390	440
Execution 4 (ms)	440	440	440	440	390
Execution 5 (ms)	380	380	440	440	390
Execution 6 (ms)	380	440	440	440	440
Execution 7 (ms)	390	380	440	380	440
Execution 8 (ms)	390	390	440	380	380
Execution 9 (ms)	390	380	440	380	390
Execution 10 (ms)	380	440	380	380	390
Execution 11 (ms)	380	390	380	380	390

Table A.13: Query 1 on MS Access 7.0 using MS ADC 1.0

Test #	1	2	3	4	5
Connection (ms)	110	110	110	110	50
Execution 1 (ms)	2370	1210	1100	1100	1050
Execution 2 (ms)	270	280	270	330	280
Execution 3 (ms)	220	280	270	270	280
Execution 4 (ms)	220	220	280	270	270
Execution 5 (ms)	220	280	280	220	280
Execution 6 (ms)	280	220	270	220	280
Execution 7 (ms)	220	270	270	280	280
Execution 8 (ms)	270	220	280	270	220
Execution 9 (ms)	270	280	220	270	330
Execution 10 (ms)	280	280	280	270	280
Execution 11 (ms)	270	220	280	220	270



Table A.14: Query 2 on MS Access 7.0 using MS ADC 1.0

Test #	1	2	3	4	5
Connection (ms)	110	110	170	110	110
Execution 1 (ms)	6090	5650	5000	4500	4940
Execution 2 (ms)	3130	3350	3070	3020	3130
Execution 3 (ms)	3030	3080	3020	3030	3020
Execution 4 (ms)	3030	3080	3080	3030	3020
Execution 5 (ms)	3020	4180	3070	3020	3080
Execution 6 (ms)	4110	3080	3020	3950	3020
Execution 7 (ms)	3020	3130	3130	3070	4010
Execution 8 (ms)	3020	3070	3020	3020	3020
Execution 9 (ms)	3020	3020	2970	2920	3020
Execution 10 (ms)	4120	4000	3070	2970	3020
Execution 11 (ms)	3020	3070	4120	4120	2910

Table A.15: Query 1 on MS SQL Server 6.5 using MS ADC 1.0

Test #	1	2	3	4	5
Connection (ms)	110	170	110	110	110
Execution 1 (ms)	1370	1090	1040	1510	1750
Execution 2 (ms)	330	330	330	440	270
Execution 3 (ms)	280	270	220	330	270
Execution 4 (ms)	220	210	270	330	280
Execution 5 (ms)	220	280	280	280	220
Execution 6 (ms)	270	220	220	320	280
Execution 7 (ms)	270	220	280	390	270
Execution 8 (ms)	280	270	280	330	220
Execution 9 (ms)	220	270	220	270	270
Execution 10 (ms)	270	220	220	280	270
Execution 11 (ms)	280	280	330	270	270

Table A.16: Query 2 on MS SQL Server 6.5 using MS ADC 1.0

<b>Test #</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Connection (ms)	170	220	110	110	50
Execution 1 (ms)	4670	4720	4780	4610	6480
Execution 2 (ms)	4780	3730	4060	3740	3900
Execution 3 (ms)	3680	3680	3680	3950	3840
Execution 4 (ms)	3620	3680	3680	3680	3840
Execution 5 (ms)	3630	3740	3680	3630	3840
Execution 6 (ms)	3680	3680	3630	3620	3850
Execution 7 (ms)	3680	3680	3680	3620	3850
Execution 8 (ms)	4770	3680	3570	3730	4510
Execution 9 (ms)	3620	4170	3620	3680	3850
Execution 10 (ms)	3630	3680	3730	3630	3850
Execution 11 (ms)	3620	3630	3630	3570	3840

## B Listing of Experimental Results: Chapter 5

This appendix contains a complete listing of all experimental data obtained from the experiments described in chapter 5.

Table B.1: Querying MS Access 7.0 using MS IDC

Test #	1	2	3	4	5
Query 1 (ms)	1590	1600	1720	1650	1650
Query 2 (ms)	1320	1040	1040	1320	1050
Query 3 (ms)	930	990	1100	930	1200
Query 4 (ms)	1210	1370	1650	1210	1270
Query 5 (ms)	1150	1260	930	1040	1100

Table B.2: Querying MS Access 7.0 using MS ADC 1.0

Test #	1	2	3	4	5
Query 1 (ms)	1510	1810	1700	1810	1810
Query 2 (ms)	610	550	490	500	550
Query 3 (ms)	550	490	500	490	500
Query 4 (ms)	550	600	550	550	550
Query 5 (ms)	500	500	550	440	490

Table B.3: Querying MS Access 7.0 using Symantec dbANYWHERE

Test #	1	2	3	4	5
Query 1 (ms)	6100	5380	5650	6150	5490
Query 2 (ms)	660	720	660	710	660
Query 3 (ms)	830	880	820	830	820
Query 4 (ms)	1100	1040	1100	1040	1050
Query 5 (ms)	760	820	820	1050	770

# Bibliography

- [Bel94] David Belson. *The Network Nation Revisited*, 1994. Available from <http://www.stevens-tech.edu/~dbelson/thesis/thesis.html>.
- [Ber93] Edward V. Berard. *Essays on Object-Oriented Software Engineering (Vol. 1)*. New Jersey: Prentice-Hall, Inc., 1993.
- [BM95] Ken Bergmann, Microsoft Developer Network Technology Group. A High-Level Look at Microsoft Internet Information Server, 1995. Available from <http://www.microsoft.com/workshop/admin/iis/iisovw.htm>.
- [CM97] Peter Coffee and Mike Moeller. Special Report: Java and Active Platform. In *ZD Internet Magazine*, 8(2):122-33, August 1997.
- [DH96] John Deep and Peter Holfelder. *Developing CGI Applications with Perl*, pp. 9-43, 89-100. John Wiley & Sons, Inc., 1996.
- [Dut96] William Dutcher. PC WEEK: Interactivating your Web site, 1996. Available from <http://www.pcweek.com/@network/0930/30cgi.html>.
- [Haz96] Deva Hazarika, Vice President, Product Development, Moai Technologies, Inc. Developing and Deploying Interactive Applications on the Internet, March 1996. Available from <http://www.microsoft.com/workshop/prog/prog-gen/mspaper.htm>.
- [Hol95] Greg Holden. *Publishing on the World Wide Web*. Hayden Books, 1995.
- [II97] White Paper: Deploying Java and JDBC – Four Types of Java JDBC Solutions, Intersolv Incorporation, 1997. Available from <http://www.intersolv.com/products/dd-wp-jdbc-solution.htm>.

- [Kra97] Ralf Kramer. Databases on the Web: Technologies for Federation Architectures and Case Studies. In *ACM SIGMOD*, 1997.
- [Lin96] David S. Linthicum. Linking Web Servers with Live Data. In *PC Magazine*, 15(15):178-79, September 1996.
- [Lip96] Robert P. Lipschutz. Web Servers. In *PC Magazine*, 15(15):167-204, September 1996.
- [Mcg96] Michael McGee. Web Pages: A Programmer's Perspective, June 1996. Available from <http://www.microsoft.com/workshop/prog/prog-gen/webpage.htm>.
- [MDE95] The Component Object Model Specification, Microsoft Corporation and Digital Equipment Corporation, October 1995. Available from <http://www.microsoft.com/oledev/olecom/title.htm>.
- [MC96] Publishing Information and applications, Microsoft Corporation, 1996. Available from [http://www.microsoft.com/iis/usingiis/resources/iis2docs/08\\_iis.htm](http://www.microsoft.com/iis/usingiis/resources/iis2docs/08_iis.htm).
- [MC97a] About Active Server Pages, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/iis/learnaboutiis/activeserver/about.htm>.
- [MC97b] ADC Web Page, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/data/adc/default.htm>.
- [MC97c] Cabinets (CAB), Microsoft Corporation, March 1997. Available from <http://www.microsoft.com/workshop/prog/cab/default.htm>.
- [MC97d] Internet Database Connector, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/sql/inet/inetdevstrat2.htm>.
- [MC97e] Internet/Database Technology Roadmap: Advanced Database Connector, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/sql/inet/inetdevstrat5.htm>.
- [MC97f] Internet Information Server 3.0, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/iis/default.asp>.
- [MC97g] Internet Server API Overview, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/win32dev/apiext/isapimrg.htm>.
- [MC97h] JScript Web Page, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/jscript>.

- [MC97i] Using Remote Data Objects and the Remote Data Control, Microsoft Corporation, 1997. Available from <http://premium.microsoft.com/msdn/library/devprods/vb/vb50docs/F1/D7/SD874.htm>.
- [MC97j] VBScript Web Page, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/vbscript>.
- [MC97k] Welcome to Open Database Connectivity, Microsoft Corporation, 1997. Available from <http://www.microsoft.com/odbc/default.htm>.
- [Ncs94] The Common Gateway Interface, NCSA HTTPD Development Team, 1994. Available from <http://hoohoo.ncsa.uiuc.edu/cgi/>.
- [NCC96] The Server-Application Function and Netscape Server API, Netscape Communications Corporation, 1996. Available from [http://www9.netscape.com/newsref/std/server\\_api.html](http://www9.netscape.com/newsref/std/server_api.html).
- [NCC97] Netscape LiveWire and Netscape LiveWire Pro, Netscape Communications Corporation, 1997. Available from [http://www.netscape.com/comprod/announce/dst\\_live.html](http://www.netscape.com/comprod/announce/dst_live.html).
- [Nor96] Ken North. PC Week Labs July 3, 1996: ODBC extends reach to servers and Web, 1996. Available from <http://www.pcweek.com/reviews/0701/01odbc.html>.
- [SN96] Pratik Patel and Karl Moss. *Java Database Programming with JDBC*. Coriolis Group Books, 1996.
- [Pim96] Ashish Pimplapure. *Virtual Groups: A Web Based Electronic Conferencing System for Online Education*. M.Sc. Thesis, Simon Fraser University, 1996.
- [Rau97] Stephen Rauch. Manage Data from Myraid Sources with the Universal Data Access Interfaces – Microsoft Systems Journal, Sept 1997. Available from <http://www.microsoft.com/msj/0997/universaldata.htm>, Sept 1997.
- [SN96] *Java Unleashed*. Sams.net Publishing, 1996.
- [SFU97] Virtual-U Research Project, Simon Fraser University, 1997. Available from <http://virtual-u.cs.sfu.ca>.
- [SH96] Performance Benchmark Tests, Shiloh Consulting and Haynes & Company, 1996. Available from <http://www.microsoft.com/InfoServ/haynes1.htm>.
- [SM96a] The JDBC database access API, Sun Microsystems, 1996. Available from <http://splash.javasoft.com/jdbc/index.html>.

- [SM96b] JDBC Drivers, Sun Microsystems, 1996. Available from <http://splash.javasoft.com/jdbc/jdbc.drivers.html>.
- [SM97] Java Universe Overview, March, Sun Microsystems, 1997. Available from <http://www.sun.com/tech/access/JavaUniverseOverview.html>.
- [Sul94] Gary C. Sullo. *Object Engineering - Designing Large-Scale Object-Oriented Systems*. New York: John Wiley & Sons, Inc., 1994.
- [TL96] TeleLearning Research Network, TL-RN, 1996. Available from [http://www.telelearn.ca/telelearn/p\\_access/overview.html](http://www.telelearn.ca/telelearn/p_access/overview.html).
- [Tur96] Shannon R. Turlington. *Exploring ActiveX*. Ventana Communications Group, Inc., 1996.
- [W3C97a] HTTP - Hypertext Transfer Protocol Overview, World Wide Web Consortium, 1997. Available from <http://www.w3.org/Protocols/Overview.html>.
- [W3C97b] W3C Activity: Hypertext Markup Language (HTML). World Wide Web Consortium, 1997. Available from <http://www.w3.org/MarkUp/Activity>.
- [Woo95] D. R. Woolley. Conference on the Web, 1995. Available from <http://freenet.msp.mn.us/~drwool/webcon2.html>.
- [Zai97] Osmar R. Zaiane. *Where Web Applications Meet Databases*. Theme 3 Workshop on Web Technology, TeleLearning NCE - Montreal, May 25 1997.