A Comparison of Web-based Technologies to Serve Images from an Oracle9*i* Database

THESIS

Submitted in fulfilment of
the requirements for the
degree of

MASTER OF SCIENCE

In the Department of
Computer Science

Rhodes University

By

Dylan Swales

January 2004

KEYWORDS

Multimedia, Intranet, Internet, World Wide Web, Active Server Pages, Active Server Pages .NET, Java Server Pages, Java Servlets, Oracle9*i*, Internet Information Services, Apache Tomcat, Database drivers, Database Providers, Response time performance, Error handling, Tracing, Caching, Ease of programming.

ABSTRACT

The nature of Internet and Intranet Web applications has changed from a static content-distribution medium into an interactive, dynamic medium, often used to serve multimedia from back-end object-relational databases to Web-enabled clients. Consequently, developers need to make an informed technological choice for developing software that supports a Web-based application for distributing multimedia over networks. This decision is based on several factors. Among the factors are ease of programming, richness of features, scalability, and performance.

The research focuses on these key factors when distributing images from an Oracle9*i* database using Java Servlets, JSP, ASP, and ASP.NET as the server-side development technologies. Prototype applications are developed and tested within each technology: one for single image serving and the other for multiple image serving. A matrix of recommendations is provided to distinguish which technology, or combination of technologies, provides the best performance and development platform for image serving within the studied environment.

ACKNOWLEDGEMENTS

I would like to start by thanking my supervisors Professor Dave Sewry and Alfredo Terzoli for the incredible amount of time and effort they have dedicated to helping me with this project. Not only have they helped me in this project but have played a major role in my growth and maturity as an academic researcher.

Thanks to the staff and postgraduate students of the Computer Science Department. They are an incredible group of people whom I strongly admire. You have all made my time at Rhodes University an unforgettable and enjoyable experience.

I would also like to thank Telkom, the NRF, and the Distributed Multimedia Centre of Excellence (CoE) at Rhodes University for their financial support.

Finally a warm word of thanks to my parents and to all my very special friends for their support, especially during the writing of this thesis. This would have not been possible without you!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY OF ACRONYMS

ADO – ActiveX Data Objects

ADO.NET - ActiveX Data Objects .NET

API – Application Programming Interface

ASP – Active Server Pages

ASP.NET – Active Server Pages .NET

BLOB – Binary Large Object

CGI – Common Gateway Interface

CLR – Common Language Runtime

COM – Component Object Model

CPU – Central Processing Unit

DAO – Data Access Object

DBMS – Database Management System

DLL – Dynamic Link Library

DNS – Domain Name Server

DSN – Data Source Name

EJB – Entity JavaBean

FSO – File System Object

HTML – Hyper Text Mark-up Language

HTTP – Hyper Text Transfer Protocol

IDE – Integrated Development Environment

IE – Internet Explorer

IIS – Internet Information Services

IP – Internet Protocol

J2EE – Java 2 Enterprise Edition

J2SDK – Java 2 Software Development Kit

JDBC – Java Database Connectivity

JSP – Java Server Pages

LAN – Local Area Network

OCI – Oracle Call Interface

ODBC – Open Database Connectivity

ODP.NET – Oracle Data Provider .NET

OHS – Oracle HTTP Server

RDO – Remote Data Object

SQL – Structured Query Language

TTC – Two-task Common

UI – User Interface

URL – Unified Resource Locator

VB – Visual Basic

WWW – World Wide Web

XML - Extensible Mark-up Language

Chapter 1

INTRODUCTION

"When you are a Bear of very Little Brain, and you think of Things,

you sometimes find that a Thing which seemed very Thingish inside you is quite different

when it gets out into the open and has other people looking at it."

A.A. Milne: The House at Pooh Corner

This chapter discusses the motivation for the study and briefly introduces the research methodology and several questions that provided direction in the study. An overview of the dissertation concludes the chapter.

1.1 Introduction and Motivation

The nature of Internet and Intranet applications is changing. Initially, HTML (Hypertext Mark-up Language) was used to create static Web applications. The content of such applications was established at the time of creation and, as a result, all client requests to that Web application were presented with exactly the same data without any customisation.

Over time, the World Wide Web (WWW) transformed from a static content-distribution medium to an interactive, dynamic one. Content on the Web is now typically dynamically generated and often used to serve multimedia (images, audio, and video), and other complex data, from back-end databases to Web-enabled clients.

The demand for media intensive Web applications has motivated the merge between Web and database technologies into a form that can be defined as *Data-driven multimedia Web applications*. Many websites are now powered in the background by object-relational databases such as Oracle9*i*.

This merge has resulted both in an appreciation of the value of multimedia and a realization of the challenges in serving multimedia to Web-enabled clients. From the developer's perspective, those challenges are motivated by the need to make a technological choice for developing software that supports a Web-based application for serving multimedia over networks.

Some of the most popular server-side Web technologies include CGI (Common Gateway Interface), Active Server Pages (ASP), Active Server Pages .NET (ASP.NET), Java Server Pages, and Java Servlets. The decision to select a particular technology is based on several factors. One of the most important factors is performance: end users expect Web-based applications to deliver content as quickly and efficiently as possible.

Yet as important as it is, performance should not be the sole reason for selecting a Web technology. Important questions to be asked include:

- Is it possible that one technology scales better than another and provides a more robust architecture for the delivered application?
- Is it easier to program and deploy a Web application in one technology than another?
- Does the technology provide an environment that delivers a rich set of features such as enhanced error handling, tracing, and caching?

With these questions in mind, a study was initiated to explore Web technologies used to serve multimedia content from an object-relational database to browser enabled clients. The project focused on a single multimedia type, namely images, and the selected server-side Web technologies were Active Server Pages (ASP), Active Server Pages .NET (ASP.NET), Java Server Pages (JSP), and Java Servlets. The object-relational database selected for the study was Oracle9*i*.

1.2 Research Methodology

To conduct this study, fully functional three-tier client-server architectures were set up for each of the selected Web technologies. With the three-tier client-server architectures in place, the various database drivers/providers allowing access to the Oracle9*i* database through each Web technology, were identified and categorised according to their ability to support the retrieval of images.

Once the various drivers/providers were selected, prototype applications were developed for each Web technology. The prototype applications were developed to access and serve images from the Oracle9*i* database to the client machine's Web browser. During application development, various features relating to each Web technology and environment were documented, including caching mechanisms, error handling and tracing, and ease of programming. In addition, the Integrated Development Environments (IDEs) used during application development were analysed.

The architectures were deployed on an isolated LAN at Rhodes University. Performance tests were then conducted to measure the response time taken to serve images from the Oracle9*i* to the client machine's Web browser. The intention of the performance tests was two-fold. Firstly, to determine which selected driver/provider generated the fastest response time within each Web technology and, secondly, to determine which technology performed best when serving single and when serving multiple images.

Based on the performance results as well as caching abilities, error handling, tracing, and ease of programming, recommendations could finally be made on which technology, or combination of technologies, to utilize when serving images from an Oracle9*i* database via the Web.

1.3 Document Overview

Chapter 2 provides background information on data-driven multimedia Web applications. It explains the architecture and operation of a three-tier client-server multimedia Web application, its components and the mechanisms it provides to facilitate media serving across an Intranet.

Chapter 3 introduces and discusses the tools available and the competencies needed to access Oracle9*i* using ASP, ASP.NET, JSP, and Servlet applications. A diagrammatic representation of available data access technologies is provided. The data access technologies include various driver/providers combined with server-side objects that permit interaction between ASP, ASP.NET, JSP, and Servlet applications and an Oracle9*i* database. Once the data access technologies have been identified, they are further analysed to determine which components directly support image serving from an Oracle9*i* database.

Chapter 4 details the development of ASP, ASP.NET, JSP and Servlet prototype applications when serving images from an Oracle9*i* database to a client's browser. The steps taken during the development of the applications are described, focusing on the benefits and limitations encountered within each application environment. Particular emphasis is placed on technology specific options when developing robust and scalable image serving applications.

In addition, this chapter describes the IDEs used for application development and details specific architectural features of each Web technology, such as caching mechanisms, error handling and ease of programming.

Chapter 5 describes of the test environment and methodologies used to conduct the performance comparisons. Various environment variables, server-side set-up options, and timing methodologies are discussed to present the reader with an in-depth understanding of the techniques used to conduct the performance comparisons made between ASP, ASP.NET JSP, and Servlets.

Chapter 6 diagrammatically presents the response times obtained when using ASP, ASP.NET, JSP, and Servlet technologies for image serving from an Oracle9*i* database. The performance results are based on the single and multiple image serving scenarios described in chapter 5.

The results are firstly grouped according to the technology used, focusing within each technology on differences due to the various providers/drivers identified in chapter 4. Then, a combination of the results is presented, comparing the four technologies against each other, for single and multiple image serving.

Chapter 7 discusses the response times presented in chapter 6 and details the author's recommendations for the development of data-driven Web applications that serve images from an Oracle9*i* database. The discussion is broken into two components. The first component solely discusses the results presented in chapter 6. It focuses on which technology (ASP, ASP.NET, JSP, or Java Servlets) performs best when serving single or multiple images from an Oracle9*i* database. The recommendations form the second part of this chapter and contain key considerations such as technology integration, ease of programming, error handling, set up, and caching.

Chapter 8 summarizes the findings and concludes with final remarks about the work done, its achievements and possible extensions.

Chapter 2

DATA-DRIVEN MULTIMEDIA WEB APPLICATIONS

"When you know a thing, to hold that you know it; and when you do not know a thing, to allow that you do not know it - this is knowledge."

Confucius (551 BC – 479 BC), The Confucian Analects

This chapter provides background information on Web applications that are driven by multimedia databases (object-relational databases). It explains the architecture and operations of a three-tier client-server multimedia Web application, its components and the mechanisms it provides to facilitate media serving across an Intranet.

2.1 Three-tier Client-server Web Architecture

Typically, cooperating applications within a Web-based architecture can be categorized as either clients or servers. The client application requests services and data from the server, and the server application responds to client requests. Figure 2.1 depicts a simple example of this client-server architecture, whereby a client makes an HTTP GET request for a Web page and the Web server responds with the HTML page back to the client, which is displayed within a Web browser.

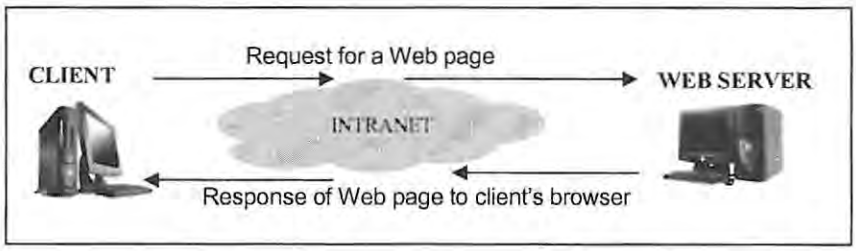

Figure 2.1 Two-tier client-server architecture

An extended version of a standard client-server model is the three-tier client-server architecture. Within this three-tier architecture, it is possible to organise distributed client-server systems for use in data-driven Web applications.

Within a three-tier data-driven multimedia Web application there are numerous communication channels between the various components. Figure 2.2 illustrates the components and information flow of this three-tier client-server architecture.

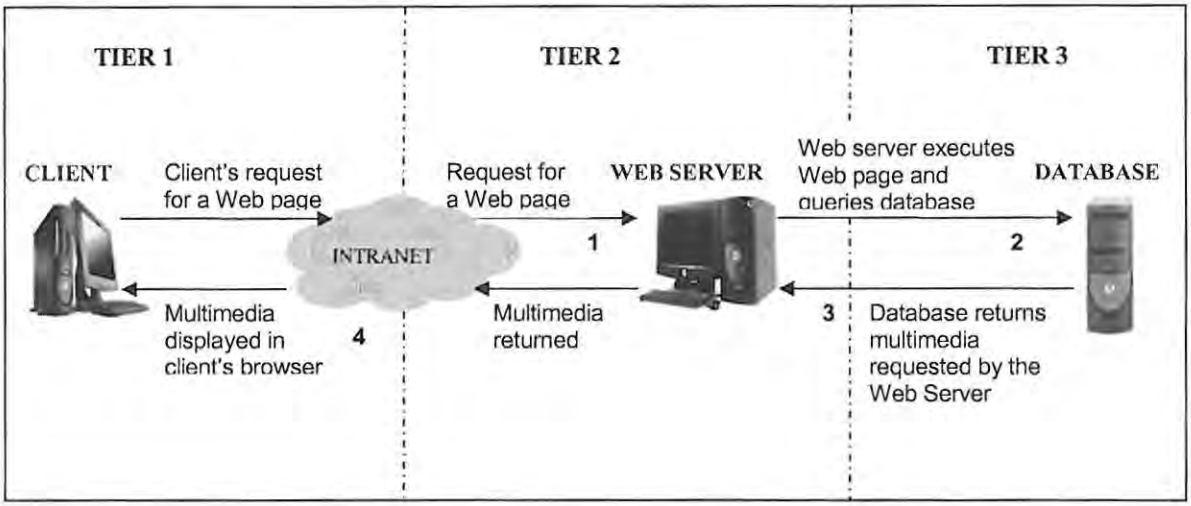

Figure 2.2 Three-tier client-server architecture

The process of multimedia (image) retrieval occurs as follows:

1. A client makes a request for a Web page by typing in the URL (Uniform Resource Locator) within the Web browser.
2. The client's browser communicates with the domain name server (DNS), located on the Intranet, to translate the server name (URL) into an IP address, which it uses to connect to the Web server. A DNS is a server that maps the numbered location of a Web server to a domain name. For example, the domain name server knows that the Web site address "http://csds.ict.ru.ac.za" points to the IP address location 146.231.123.30 on the Rhodes University LAN.
3. Following the HTTP (Hyper-text Transfer Protocol) protocol, the browser sends a GET request to the server, asking for a specific server-side page (referenced by label 1 in figure 2.2).
4. The Web server makes a connection to the object-relational database that holds the multimedia content. This is accomplished by using specific database drivers/providers and specifying connection parameters.

5. Once a connection is established, a SQL (Structured Query Language) command is used to query a specific table within the database corresponding to the relevant multimedia (referenced by label 2 in figure 2.2).

6. The multimedia is then sent back to the Web server within various objects specific to the server-side technology used (referenced by label 3 in figure 2.2).

7. The server then sends any HTML document, including the multimedia back to the client's browser where it is rendered (referenced by label 4 in figure 2.2).

Most three-tier client-server systems perform the following tasks, which correspond to three tiers, or layers, of the 3-tier model (Table 2.1): (Microsoft Corporation, 2003)

Tier	Task	Description
1	User interface and navigation	This layer comprises the entire user experience. Not only does this layer provide a graphical interface so that users can interact with the application, input data, and view the results of requests, it also manages the manipulation and formatting of data once the client receives it. In Web applications, the Web browser performs the tasks of this layer.
2	Application logic	Tier 2 is situated between the user interface and the database and is the domain of the Web application developer. The application logic tier is the middle tier, which bridges the gap between the user interface and the underlying database, hiding technical details from the users. In Web applications the Web server and server-side development technologies control the application logic.
3	Data services	Tier 3 is provided by a structured data store, which manages and provides access to the application data. In terms of Multimedia Web applications the data store is normally in the form of an object-relational database such as Oracle 9*i*.

Table 2.1 Tasks performed within a three-tier client-server application

Each component within the three-tier architecture plays an important role in the multimedia serving process. A brief description of the components and their core features and functionality follows.

2.1.1 Databases

Multimedia content has diverse and complex data types and includes a wealth of meta-data (information about the media content) that is critical to the way this data is queried, indexed, and optimized for retrieval and rendition, which is different from that of traditional relational data.

The conventional, inexpensive way to support multimedia data within a data-driven Web application is through the use of file-based speciality servers. These servers manage a single, specific media type and support the formats and the rendition of the formats associated with the data, the administration of the system, the indexing and retrieval of the data, and the creation and analysis of the meta-data associated with the data (Oracle Corporation, 2000b). These speciality servers are an appropriate solution when only a small number of users need access to this information. However, these servers are not designed for large volumes of data, not for large numbers of simultaneous users, and become extremely complex to maintain and administer when the information in Web applications includes many types of multimedia and other non-traditional data (Mauro, 2001b). Consequently, organizations have started to recognize the disadvantages of flat file systems – the lack of central management, synchronization, availability, scalability, and transaction versioning that are necessary for the dynamic nature of internet applications (Oracle Corporation, 2002a).

As a result, some database vendors have extended their traditional bounds, from storing relational data, to handling complex multimedia objects within a relational scheme. Web application developers now leverage the advanced data management services found in established relational databases for use in media-rich Web applications. One such database is Oracle 9*i*.

Oracle 9*i* is an object-relational database management system. This means that, in addition to managing relational data, it provides support for the definition of more complex object types, the data associated with those objects, as well as methods that can be performed on them. As a result, Oracle 9*i* extends its traditional bounds to include support for BLOBs (Binary Large Objects) as a basis for adding multimedia, such as digitized audio, image, and video into its database (Oracle Corporation, 2000a).

Oracle *inter*Media is a feature that enables Oracle9*i* to store, manage, and retrieve this multimedia in an integrated fashion with other enterprise information. The benefit is that relational and multimedia data remains in synchronization and allows Web applications to benefit from Oracle's advanced database technology services (Mauro, 2001a). For example, developers can provide a base set of services within their applications to insert multimedia data in existing or new database tables, perform image processing on a number of image formats, and perform conversions between image formats.

2.1.2 Web Servers

The role of a Web Server within data-driven Web applications is simply to service requests and responses between a client and other servers and is responsible for invoking server-side development technologies for Web page processing. The primary Web server within a Windows-based system, running ASP and ASP.NET applications, is IIS (Internet Information Services), whilst Apache Tomcat is popular in the JSP and Java Servlet community. The Oracle HTTP server is the Oracle-extended version of the Apache Web server running under the Oracle9*i* Application Server. JSP and Servlet applications cannot be serviced by IIS and vice-versa for containers that serve JSP and Servlets. Although the Web server is an important aspect within the three-tier architecture of data-driven Web applications, it is the functionality of the Web server in conjunction with the appropriate Web development technology that is more important and is discussed in greater detail in later chapters.

2.1.3 Clients

The Client tier consists of the most basic functions and have the least processing required within the three-tier Web architecture. Client machines are in effect "dumb terminals" with the role of requesting Web applications and waiting for the appropriate response of multimedia and/or HTML to be displayed within a Web browser. The processing of dynamic Web applications is accomplished on the Web server using a server-side development technology and the end result is data that is sent back to the client where it is formatted and displayed within the Web browser. Therefore, minimal processing power is required on the client tier, conforming to the "thick server and thin client" paradigm.

2.2 WEB DEVELOPMENT TECHNOLOGIES

There are a multitude of server-side technologies employed to produce dynamic Web-based applications. CGI (Common Gateway Interface), Java Servlets, JSP, ASP and ASP.NET are among them.

2.2.1 Server-side Development Technologies

CGI was the earliest technology used to service a dynamic HTTP request by a client. A CGI program has a main function to process the client request and generate output in a browser-viewable format, for example, HTML, JPEG etc. However, the main disadvantage of CGI is that for each incoming client request, a new process has to be created on the server. The result is that performance is greatly affected by the overhead of process creation every time a dynamic request is served (Michalas *et al* 2000).

Servlets are Java technology's answer to CGI programming. A Java Servlet is a program that runs on the Web server, acting as a middle layer between a request from a Web browser or other HTTP client and databases or applications on the HTTP server (Hall, 2001). Unlike CGI, Servlets use a single process, multi-threaded model with comparatively little overhead in handling client requests (Barthalo, 2003).

Java Server Pages (JSP), another Java-based technology, provides Web developers with a framework to create dynamic content using HTML and Java code (Hall 2001). A JSP file contains standard HTML, interspersed with Java code used to generate dynamic content. Web applications built using JSP technology are typically implemented using a translation phase that is performed once, the first time the page is called. The page is compiled into a Java Servlet class and remains in server memory, so subsequent calls to the page results in a shorter response time (Wassef, 2000).

Microsoft introduced Active Server Pages (ASP), along with its IIS 3.0, to allow developers to create Web pages that can interact with databases and other applications. Microsoft defines ASP as "an open, compile-free application environment in which you can combine HTML,

scripts, and reusable ActiveX server components to create dynamic and powerful Web-based business solutions." (Microsoft Corporation, 2001a). ActiveX is a software technology, built on the Component Object Model (COM) foundation, which allows networked components to interact with each other independently of the languages in which they are written (Microsoft Corporation, 1997). Most ASP pages are written using VBScript (a subset of Microsoft's Visual Basic programming language) or JScript, but support for languages such as Perl and Python are available through third-party vendors (Gladwin, 2000).

ASP is similar to JSP in that they both render dynamic content and separate HTML from server-side code. ASP however, has three important limitations. Firstly, ASP is currently available only with IIS, running under the Windows operating system. Secondly, ASP code is written in scripting languages, which are not suitable for complex applications that require reusable components (Hall 2001). Lastly, ASP pages need to be recompiled on every request and do not store cached, compiled pages within server memory (Mirage, 2003).

ASP, being based on the Component Object Model (COM) and Win32 API technologies, have not provided a very coherent architecture for modern distributed applications, whereas with Java 2 Enterprise Edition (J2EE), Sun produced a suite of technologies that developers could employ, beginning with Standard Edition projects and scaling up to full Enterprise JavaBeans (Ahmed *et al* 2002).

With the advent of Microsoft's .NET framework, ASP.NET has shifted from traditional scripting to a fully-fledged programming architecture. Programmers can employ Visual Basic (VB), C# and a variety of other languages found within the .NET framework. ASP.NET is now an object-oriented Web application development platform, and has seen many improvements with regard to error handling, memory management, scalability, and flexibility to name a few (Ahmed *et al* 2002).

2.2.2 Analysis of Web Development Technologies

In this dissertation, a comparison is made between four Web development technologies, namely ASP, ASP.NET, Java Servlets and JSP. The study was motivated by a desire to make a technology choice for developing software that would support Web-based multimedia services. A review of existing literature showed varying conclusions about the superiority of one technology over another.

In 2002, Esposito published an article discussing the possible issues that arise when porting Web applications from ASP/ADO to ASP.NET/ADO.NET. Esposito explains that the use of ADO objects within .NET applications is possible but two considerations should be taken into context. Firstly, the VBScript programming language used to develop ASP applications is not completely compatible with the VB.NET language used within .NET applications. Secondly, one needs to consider data binding – which is the ability to bind data directly to User Interface (UI) controls. This is not possible in ASP/ADO and shortens the programmatic task to display dynamic data (Esposito, 2002).

Microsoft released a paper in October 2001 presenting the results of a comprehensive performance benchmark of Web application development technologies based on the Nile e-commerce Application Server Benchmark. In the paper a performance comparison was conducted between an ASP.NET application (implemented using C# and ADO.NET) and an ASP application (implemented using VBScript and ADO). Both applications were based on accessing relational data from a SQLServer database. The results showed that under a user load of 750 clients accessing the Web application, the ASP application was a third slower than the ASP.NET application (Microsoft Corporation, 2001c).

A set of performance results presented by Moore in November 2002 illustrated the difference between the DataReader and DataSet objects used within ADO.NET. The results indicated that the DataReader object outperforms the DataSet object and should be utilized whenever possible (Moore, 2002). Moore further illustrated that the DataSet object is optimal when data caching is enabled or with use in XML Web Services.

In July 2002, Microsoft published benchmark data that assessed the performance gains for an ASP.NET data-driven Web application when using the .NET Framework Data Provider for Oracle instead of the Microsoft OLE DB Data Provider for Oracle. The benchmark was conducted using the Nile 3.0 e-commerce Benchmark and the primary metric used to measure performance was throughput (number of pages served per second). The results indicated that the .NET Framework Data Provider scaled significantly better than the OLE DB provider by as much as 300% (Leake, 2002).

"Sun held up a Pet Store as its blueprint for implementing best practices to build Web applications. The .NET version should be held in the same standards: It should serve as a template for people building Web applications in .NET."

-Scott Stanfield, CEO of Vertigo Software

In May of 2001, Sun Microsystems introduced the Java Pet Store demonstration as a design blueprint for customers to follow when implementing J2EE Web applications. The Pet Store demonstrated how to use the capabilities of JSP, Java Servlet, and Enterprise JavaBeans (EJB) to develop flexible, scalable, cross-platform enterprise applications (Sun Microsystems, 2003a).

In June 2001 Oracle modified the Java Pet Store application in an attempt to demonstrate the performance and scalability of its J2EE application server (Oracle9iAS) against other leading application servers. The results indicated that Oracle9iAS provided superior scalability and performance, and required half the memory resources over another unnamed application server (Oracle Corporation, 2001).

In response to this publication (November 2001), Microsoft announced it had re-implemented the Java Pet Store using ASP.NET and C#. Microsoft pronounced that the .NET version required one-third of the lines of code, provided 28 times faster average response times (for 450 concurrent users), required one-sixth of the CPU utilization, and scaled much better as the number of users increased (Microsoft Corporation, 2001b).

It was not until June 2003, that an independent organisation, The Middleware Company, was able to conduct a performance case study with the full support of the J2EE and .NET platform technology vendors, together with a selected panel of industry experts and practitioners.

The case study tested performance when hosting a typical Web application with steadily increasing user loads. The performance comparison was based on testing a J2EE application server with a JSP/Servlet implementation and a JSP/EJB implementation of the specification, plus the Microsoft .NET implementation of the specification (The Middleware Company, 2003). The results displayed an almost identical throughput using the .NET and J2EE/EJB architectures. However, without the use of EJB in the application, the J2EE architecture performed almost a third slower than the other applications. The J2EE/EJB solution fared slightly better than the .NET solution (about 2%) when using an Oracle 9i database.

2.3 Summary

This chapter provided background information on Web applications that are driven by multimedia databases (object-relational databases). It explained a typical three-tier client-server architecture used for data-driven multimedia Web applications as well as detailing components and operations within each tier of the three-tier model. The chapter also introduced and provided literature analysis on server-side Web development technologies.

CHAPTER 3

IDENTIFICATION, ANALYSIS, AND SELECTION OF TECHNOLOGIES USED TO ACCESS ORACLE9*i*

"Do not wait; the time will never be 'just right'. Start where you stand, and work with whatever tools you may have at your command, and better tools will be found as you go along."

Napoleon Hill

This chapter introduces and discusses the various tools and competencies available to access Oracle9*i* using ASP, ASP.NET, JSP, and Servlet applications. An identification and diagrammatic representation of available data access technologies is provided. The data access technologies include various driver/providers combined with server-side objects that permit interaction between ASP, ASP.NET, JSP, and Servlet applications and an Oracle9*i* database. Once the data access technologies have been identified, they are analysed to determine which components directly support image serving from an Oracle9*i* database. Finally, a combination of architectures is provided, representing the data access technologies used to develop image serving applications.

CHAPTER 3. IDENTIFICATION, ANALYSIS, AND SELECTION OF TECHNOLOGIES USED TO ACCESS ORACLE9i

The driving force behind a dynamic Web site is the ability of a Web application to interact with a database. Microsoft provides ASP and its successor, ASP.NET, for developing such applications whilst Java provides JSP and Servlets. Developers are faced with the decision of selecting not only a particular technology, but also the various database access options made available within the technologies for interacting with relational and object-relational databases. An even more important decision, with respect to the research work, is which database access technology and associated server-side options enable image serving between an Oracle9*i* database and the associated Web application. Can ASP and ASP.NET applications utilize the same data access technologies?

The most practical method of performing such a comparison is to analyse sequentially the various components that are common to Web applications as they interact with the Oracle9*i* database. From a high level perspective, this interaction is a two-step process for image serving.

- First, a connection to a database is established. This connection provides a communication link between the Web application and the Oracle9*i* database and is accomplished using vendor-specific database drivers/providers.

- Next, the Web application communicates with the database about the data to be retrieved. The communication is accomplished by using SQL statements to query specific records within the database whereupon data is retrieved from the database using technology specific server controls and objects.

The following sub-sections include a detailed analysis of the various components used in this two-step process of image serving within ASP, ASP.NET, JSP and Servlet applications. Once the various components and technology-specific database access technologies have been identified, they are further analysed to determine which options directly support image serving from an Oracle9*i* database. Emphasis is placed on database drivers/providers and server-side objects available within each technology.

3.1 ASP Database Access

In the past, the only way to interface with a database from any application was to use a low-level API (Application Processing Interface) for each vendor's database. Although this approach provided a quick and efficient means to access data, it often posed a problem when applications must access multiple databases running on different platforms, necessitating each application to have a different API version to support each database (Michalas *et al* 2000). Later, a universal API came into existence through ODBC (Open DataBase Connectivity) and many database vendors conformed to this standard and developed ODBC-compliant drivers for their databases. ODBC technology provides a common API for accessing heterogeneous databases. The API provides application developers with the ability to access different database management systems (DBMSs) through common code, using SQL as the standard for accessing data. Thus, a developer can build and distribute a client/server application without targeting a specific DBMS. Figure 3.1 illustrates a standard ODBC architecture (Lee, 2002).

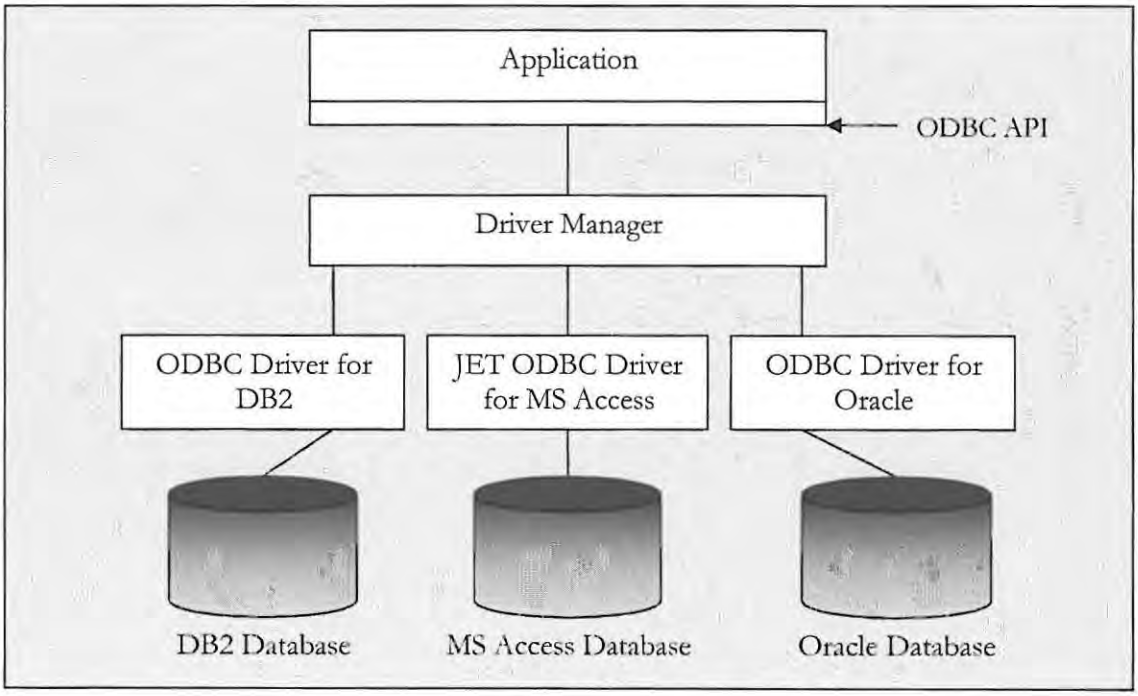

Figure 3.1 ODBC Architecture

As Figure 3.1 illustrates, the driver manager provides the intermediate link between the application and the databases. The ODBC interface contains a set of functions that the drivers of each DBMS implement.

ODBC is a low level interface, which means that Visual Basic (VB) programmers, including ASP developers using VBScript, cannot access the API directly, as it was designed for the C and C^{++} type language. VB programmers typically access the ODBC API through Data Access Objects (DAO). DAO are designed specifically for the Microsoft Jet engine, the database engine for the Microsoft Access database. DAO can also connect to other databases such as Oracle, using the Jet engine to translate calls between DAO and ODBC. Figure 3.2 illustrates the use of DAO with a Microsoft Access database and an Oracle database (Lee 2002).

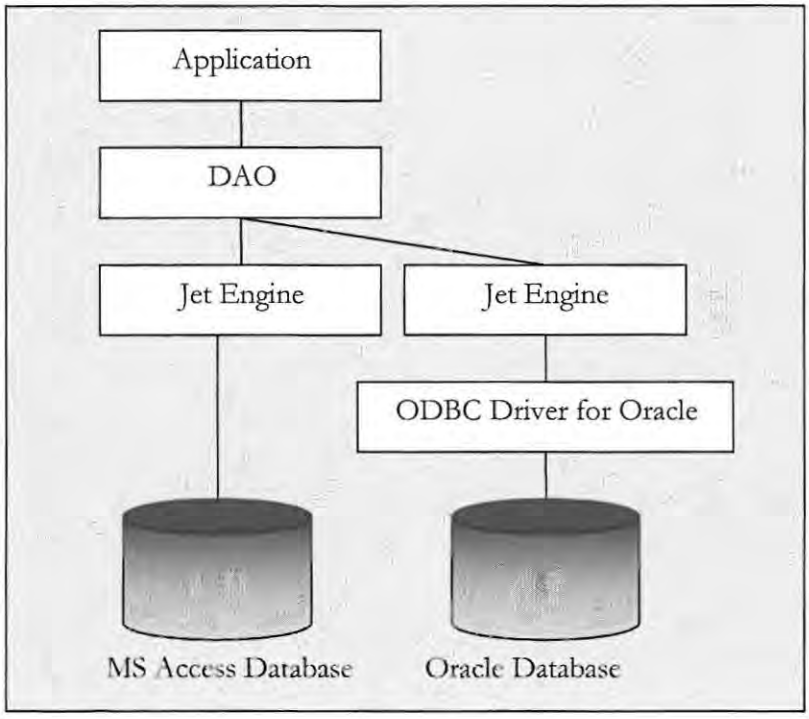

Figure 3.2 Using DAO to Access Databases

This extra translation step between DAO and the ODBC driver results in a slower connection between the application and the Oracle database (Lee 2002). To overcome this limitation,

Microsoft introduced Remote Data Objects (RDO) which accesses the ODBC API without the extra translation step through the Jet engine. Figure 3.3 illustrates the use of RDO to access an Oracle Database.

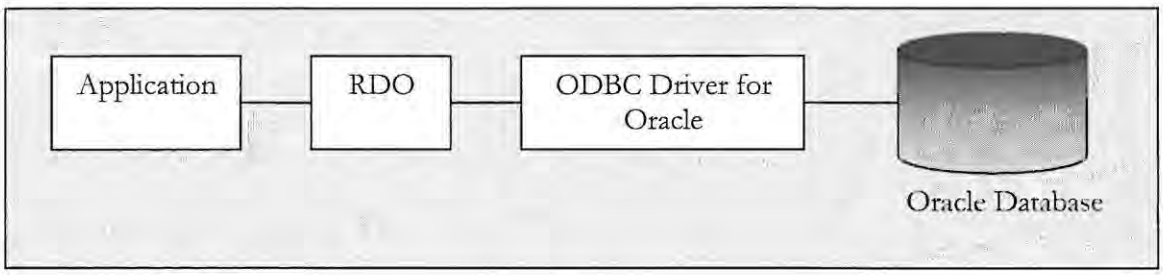

Figure 3.3 Using RDO to access an Oracle Database

Since the release of ODBC, Microsoft introduced Object Linking and Embedding Database (OLE DB). OLE DB is a COM-based data access object that builds on the success of ODBC by providing an open standard for accessing heterogeneous data. Whereas ODBC was created to access a relational data source, OLE DB is designed to include many other data stores, including relational, non-relational (such as email and directories services), and unstructured data sources. OLE DB includes a bridge to ODBC to enable continued support for ODBC drivers as ODBC is the most optimized architecture for accessing relational data stores (Lee 2002).

However, OLE DB provides binding for C and C^{++} programmers and since VB does not provide pointer data types it cannot make direct calls to OLE DB using RDO. Hence, another data access model was introduced, called ActiveX Data Objects (ADO), which contains several built-in objects for accessing data from data stores.

Three primary server-side objects within ADO are used to retrieve data from an associated database, namely the *Connection*, *Command*, and *RecordSet* objects. The Connection object establishes a connection to the database using a database driver, such as ODBC or OLE DB, with associated parameter details. The Command object is used to query the database whilst the RecordSet object provides sequential, connected data access to the specified table in the

database. Figure 3.4 illustrates the many paths through which ASP applications can connect to a database using ADO and an associated database driver for an Oracle9*i* database.

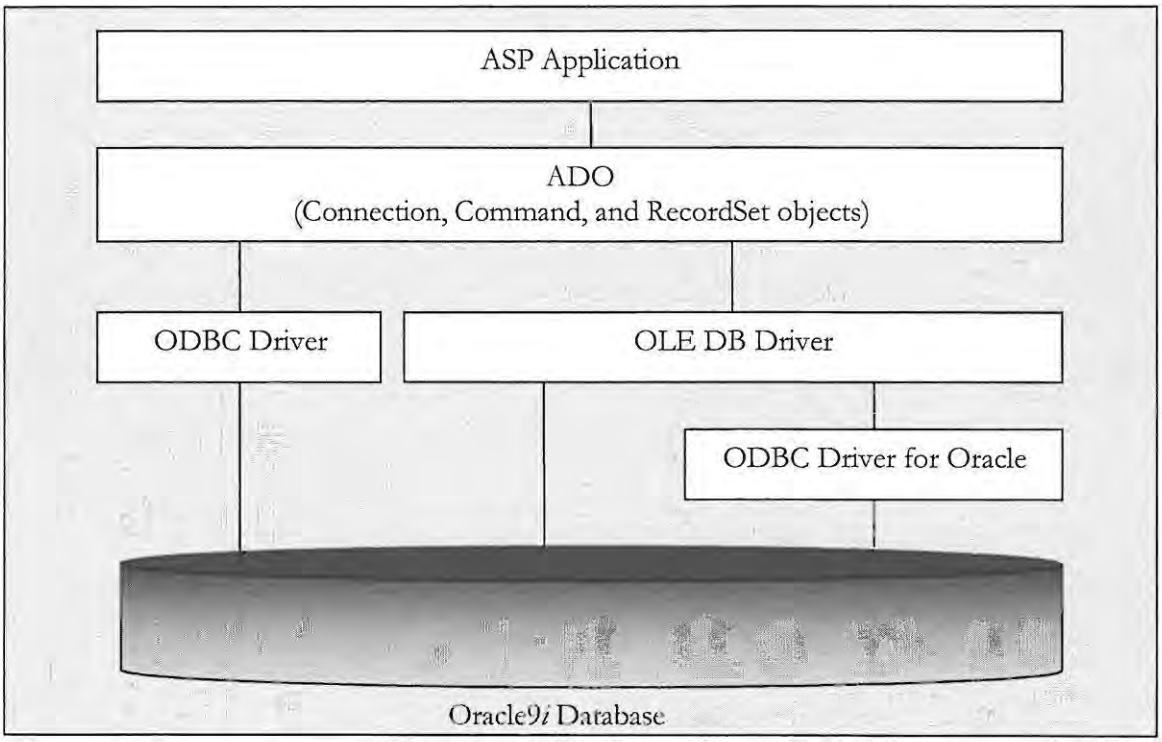

Figure 3.4 Various connection routes an ASP application can take with ADO

This architecture identifies the most recent and supported technologies for developing ASP data-driven Web applications. The question remains as to which data access technology, depicted in figure 3.4, provides the components necessary for image serving from the Oracle9*i* database. This is further complicated as vendors provide customised ODBC or OLE DB drivers for an Oracle database. For example, both Oracle and Microsoft provide ODBC drivers for accessing an Oracle9*i* database. However, the Oracle ODBC driver might provide multimedia serving capabilities whilst the Microsoft ODBC driver may not. A discussion on data access technology selection, based on image serving support for an Oracle9*i* database, can be found in section 3.4.

3.2 ASP.NET Database Access

When Microsoft started developing the .NET Framework, they decided to redesign the data access model instead of extending the ADO platform. ADO was found to have three major pitfalls. Firstly, ADO was designed primarily for connected data access based on COM marshalling which consumes much needed resources on the server (Lee 2002). COM marshalling is the process of converting data types from a database to the ADO object (Mader, 1996). Secondly, there was no coherent integration with Extensible Mark-up Language (XML) which provided the basis for XML Web services. Lastly, ADO needed to be compatible with the base class library type system found within the .NET framework which was not consistent with the current ADO model (Lee 2002). Consequently, Microsoft developed ADO.NET which provides data access and manipulation capabilities based on disconnected data, and uses XML as the universal transmission format in place of COM marshalling.

An essential component of ADO.NET is the .NET data provider which implements the ADO.NET API for database access. The ADO.NET data provider provides some of the same objects as ADO, for example the Connection and Command objects, but provides two new objects (in place of the RecordSet object) for data access, namely the *DataReader* and *DataSet* object.

The DataReader object provides a connected read-only/forward only view of the data which is similar to the RecordSet object found within ADO.

The DataSet object provides a disconnected model representing a cache of data. Data originating from multiple sources (database, XML file, code, user input) can be entered into the DataSet, each stored as an instance of a *DataTable* object, where it can be manipulated or converted before sent as output to a browser, form, file, or database. However, a *DataAdapter* object is needed for the retrieval and saving of data between the DataSet object and the specified source data. The DataAdapter object uses the DataReader object to populate the DataTable instance. As an optional component, a *DataView* object can be utilised to provide a

custom view of a DataTable instance within the DataSet and several *list-bound* controls can be used to display the data.

Figure 3.5 illustrates the various options available to communicate with an Oracle database using the DataSet and DataReader objects found within ADO.NET as well as the RecordSet object found within ADO.

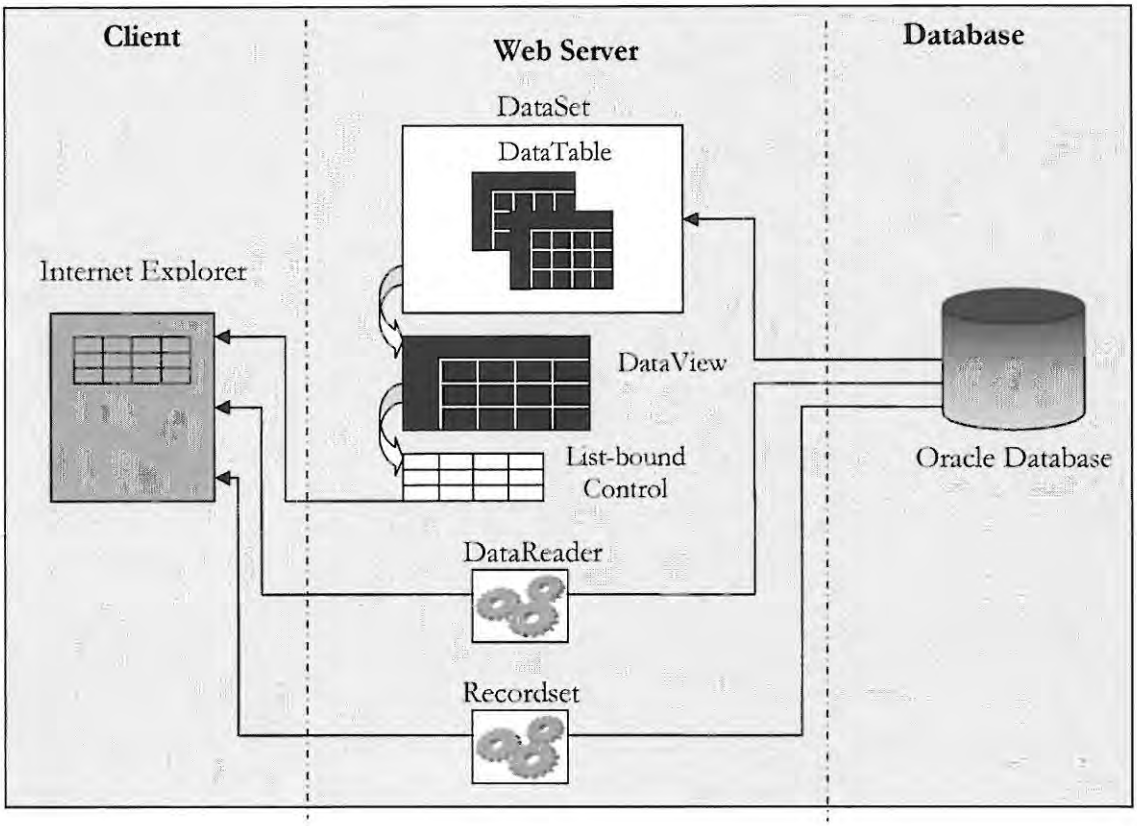

Figure 3.5 ASP/ASP.NET objects used to communicate with an Oracle database (Adapted from Microsoft Corporation (Microsoft Corporation, 2002)).

The .NET Framework provides the OLE DB .NET and ODBC .NET data providers for communicating with non-SQL databases such as Oracle. The .NET Framework data provider for OLE DB communicates with an OLE DB data source through an OLE DB driver. One important aspect of this approach is that the OLE DB driver can be identical to those used within ADO. For example, the OLE DB driver within ADO, illustrated in figure 3.4, can be used when the .NET Framework provider for OLE DB is utilized for data access within

ADO.NET. This option demonstrates database access technology integration between ASP and ASP.NET applications. The .NET Framework Data Provider for ODBC provides a similar architecture to the .NET Framework Data Provider for OLE DB; for example, it calls into the ODBC driver for the data source. Both approaches use COM marshalling for data access.

Another option is to use a direct Oracle data provider which enables data access to Oracle data sources through the Oracle client connectivity software, Oracle Call Interface (OCI). OCI is an API that allows developers to create applications that use the native procedures or function calls of a third-generation language to access an Oracle database server and control all phases of SQL statement execution (Oracle Corporation, 2003)). This approach bypasses any ODBC or OLE DB driver but the OCI software must to be installed on the server using the Oracle data provider. Figure 3.6 illustrates the various paths an ASP.NET application can take to communicate with an Oracle database.

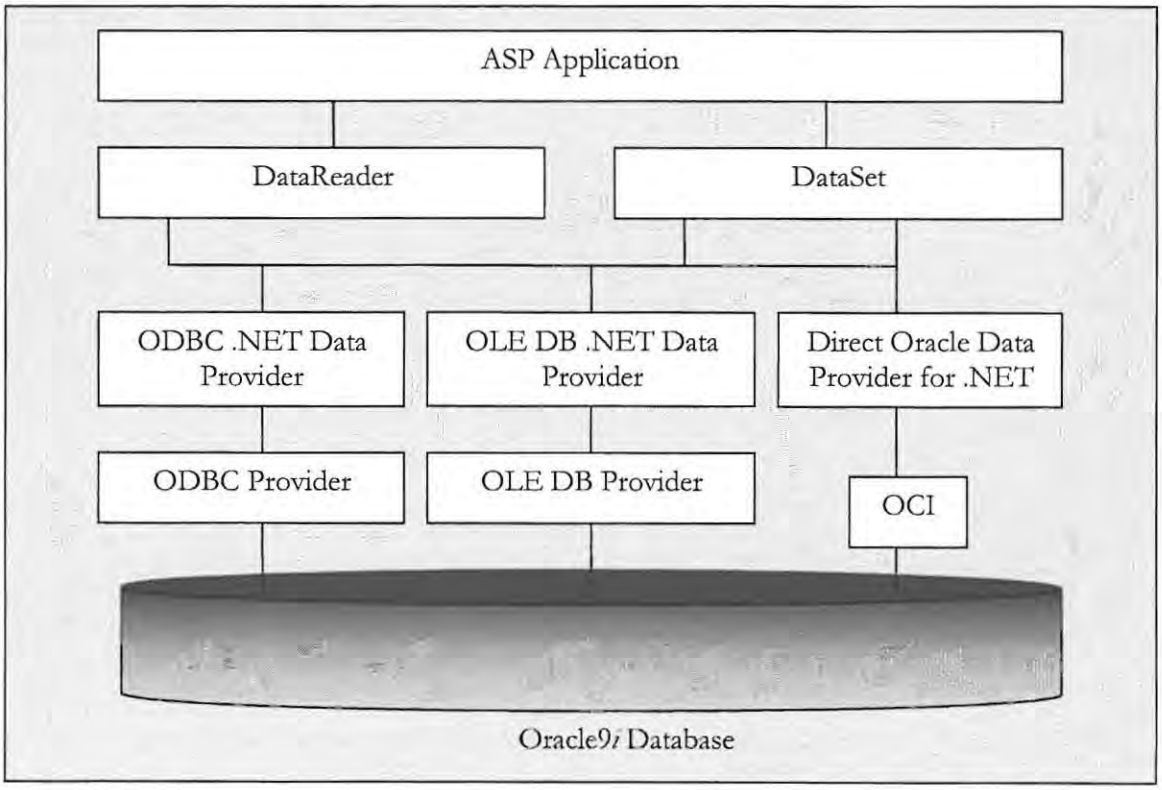

Figure 3.6 Various connection routes a ASP.NET application can take with ADO.NET

Figure 3.6 represents the most recent and supported data access technologies and components for developing ASP.NET Web applications. This combination of architectures is used as the basis for identifying which providers/drivers support image serving from an Oracle9*i* database. This is further complicated by vendors developing customized providers for use in ASP.NET applications. Additionally, an appropriate selection of the DataReader or DataSet objects must be established with a possible selection of a list-bound control used in conjunction with the DataSet object (refer to figure 3.5).

3.3 JSP and Servlet Database Access

Java Database Connectivity (JDBC) is a Java API providing an industry standard for database-independent connectivity between the Java programming language, which includes JSP and Servlets, and a wide range of data sources. The JDBC API consists of a set of packages, containing classes and interfaces, written in Java, that provide developers with the ability to connect to a database, issue SQL statements and process the results.

To use the JDBC API with a particular database, a JDBC technology-based driver is needed to mediate between the JDBC API and the DBMS. For example, the `java.sql` package, found within the JDBC API, contains the `DriverManager` class whose primary function is to connect a Java application to the correct JDBC technology-based driver. The `java.sql` package also provides interfaces for the *Connection*, *PreparedStatement*, and *ResultSet* objects. The Connection object establishes a connection to the database, the PreparedStatement object represents a precompiled SQL statement, and the ResultSet object represents a table of data which is generated by executing the SQL query within the PreparedStatement object. These objects are analogous to the Connection, Command, and RecordSet objects found within ADO.

CHAPTER 3. IDENTIFICATION, ANALYSIS, AND SELECTION OF TECHNOLOGIES USED TO ACCESS ORACLE9i

JDBC technology-based drivers fit into one of four categories (White *et al* 2002):

Type I. *JDBC-ODBC Bridge plus ODBC Driver*: The Java Software Bridge provides JDBC access via ODBC drivers. An installation of J2SDK 1.4 includes a JDBC-ODBC Bridge driver. The ODBC driver, and in many cases database client code, must be installed on the machine that uses this driver.

Type II. *Native-API partly Java Technology-enabled Driver*: The *native-API partly Java technology-enabled driver* converts JDBC calls into calls on the API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, database-specific client code is required to be installed on the machine that uses this driver.

Type III. *Pure Java technology-enabled Driver for Database Middleware*: This driver translates JDBC API calls into a DBMS-independent middleware protocol which is then translated to a DBMS protocol by the middleware server. The specific protocol used depends on the vendor and is the most flexible JDBC API alternative. To support Internet access, the middleware protocol must handle the additional requirements that the Web imposes, such as security, access through firewalls, *etc.*

Type IV. *Direct-to-Database Pure Java technology-enabled Driver*: This kind of driver converts JDBC calls directly into the network protocol used by DBMSs, providing the most practical solution for Intranet access. Since many of these protocols are proprietary, the database vendors themselves are the primary source. Several that are now available include Oracle, Sybase, Informix, IBM DB2, and Microsoft SQLServer.

Figure 3.7, below, illustrates how a JSP or Servlet application interacts with an Oracle9*i* database through the JDBC API and the four technology-based drivers.

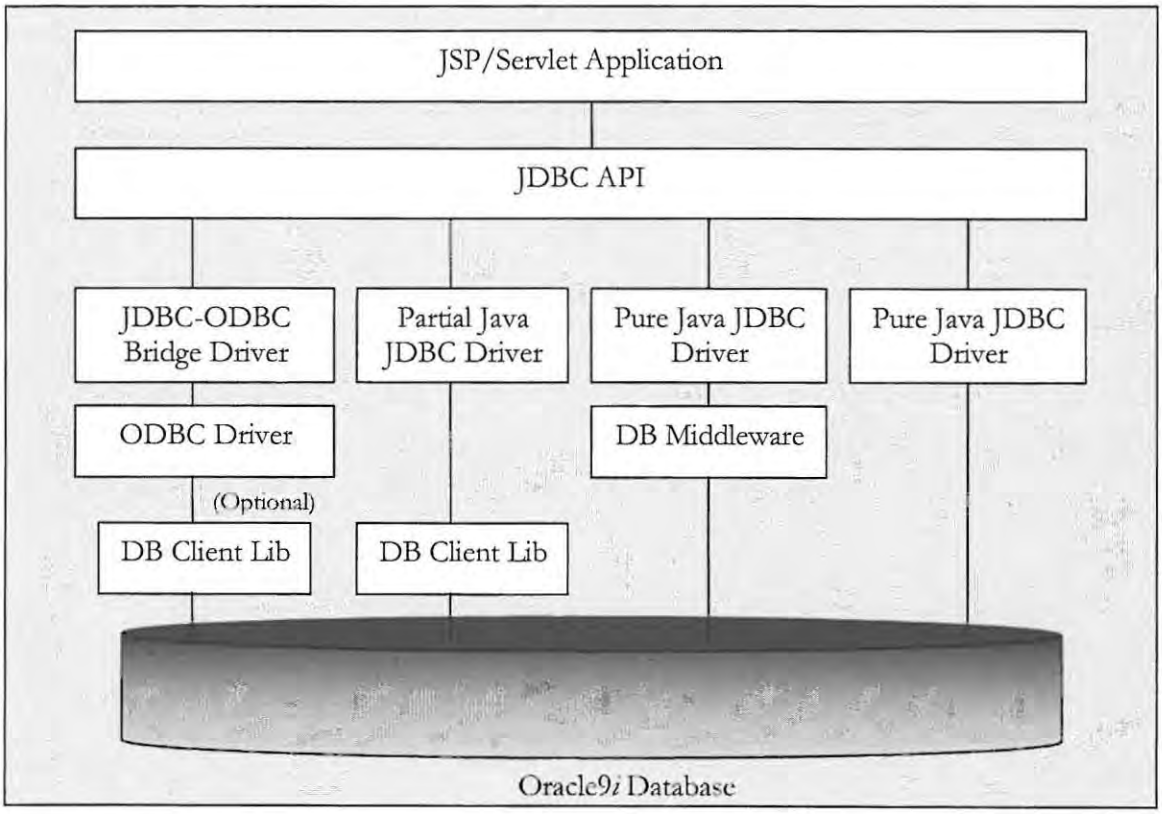

Figure 3.7 JDBC technology-based drivers

The architecture illustrated in figure 3.7 identifies the most recent and supported data access technologies and components for JSP and Servlet database access. Section 3.5 provides a detailed discussion of the JDBC technology-based drivers supporting image serving from an Oracle9*i* database.

3.4 Analysis of various drivers/providers supporting image serving from an Oracle9*i* database using ASP and ASP.NET.

The following method was used to determine which path and associated database access technologies, illustrated in figures 3.4 and 3.6, support image serving between the Oracle9*i* database and the ASP or ASP.NET application:

1. The various vendor-specific database drivers/providers currently available for accessing an Oracle9*i* database using ASP and ASP.NET in conjunction with ADO and ADO.NET were identified.
2. ASP and ASP.NET applications were then analysed to determine which database drivers/providers could connect to the Oracle9*i* database and access meta-data associated with an image.
3. Each successful application was then extended to observe if it supported the serving of multimedia, with specific reference to the Oracle ORDImage data type for images.

Table 3.1 summarises the results of steps 1 to 3 above.

Technology	Database Driver/Provider	Supports Meta data	Supports Multimedia (Images)
ASP and ADO	**Oracle ODBC Driver**	√	√
ASP and ADO	**Microsoft ODBC Driver**	√	
ASP and ADO	**Oracle OLE DB Driver**	√	√
ASP and ADO	**Microsoft OLE DB Driver**	√	
ASP.NET and ADO.NET	**.NET Framework Data Provider for ODBC with Microsoft ODBC Provider**	√	
ASP.NET and ADO.NET	**.NET Framework Data Provider for ODBC with Oracle ODBC Driver**	√	√
ASP.NET and ADO.NET	**.NET Framework Data Provider for OLE DB with Microsoft OLE DB Driver**	√	
ASP.NET and ADO.NET	**.NET Framework Data Provider for OLE DB with Oracle OLE DB**	√	√

ASP.NET and ADO.NET	**Microsoft .NET Framework Data Provider for Oracle**	✓	✓

Table 3.1 Providers supporting image serving within ASP and ASP.NET.

As illustrated in table 3.1, all drivers/providers allow for the connection and access of meta-data associated with images within the Oracle9*i* database. Both the Oracle ODBC and OLE DB providers permit image access and retrieval within ASP/ADO, whilst the Microsoft ODBC and OLE DB providers do not.

Within the .NET environment, the Microsoft .NET Framework providers for ODBC and OLE DB permit access and retrieval of images from the Oracle 9*i* database. However, this approach is only possible if the .NET Framework providers for ODBC and OLE DB work in conjunction with the Oracle-supplied drivers for ODBC and OLE DB. Additionally, the Microsoft .NET Framework data provider for Oracle allows for the access and retrieval of images from the Oracle9*i* database.

According to Oracle, an additional provider, the Oracle Data Provider for .NET (ODP.NET) provides the access and retrieval of multimedia and meta-data associated with multimedia from the Oracle9*i* database. However, ODP.NET was not utilized for the research reported in this dissertation as the Oracle Client (Version 1), used for previous application performance comparisons does not support ODP.NET.

Figure 3.8 illustrates a combination of the database drivers/providers that support multimedia serving from the Oracle9*i* database with associated data access technologies. This architecture provides the basis for the performance tests of ASP and ASP.NET, and of the DataSet and DataReader objects within ASP.NET.

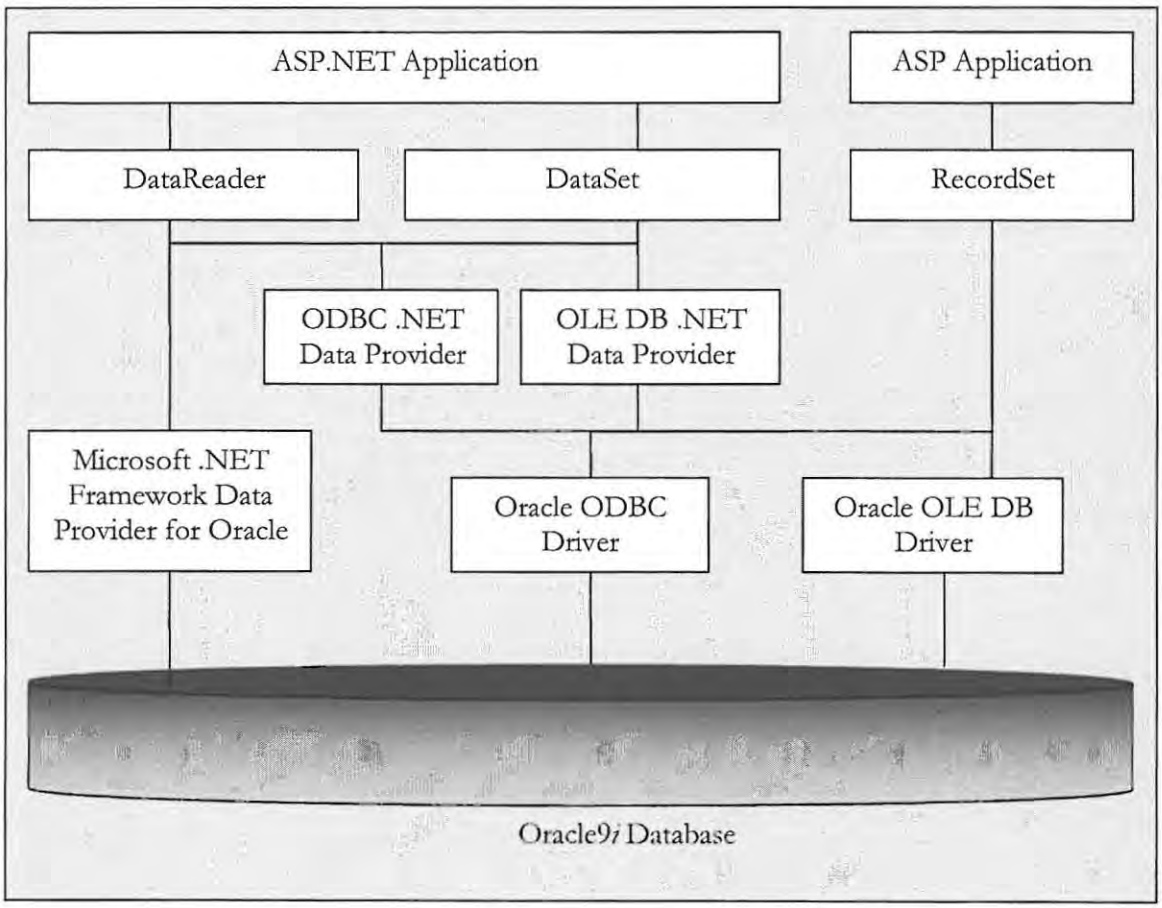

Figure 3.8 ASP and ASP.NET server objects and drivers/providers supporting image access and serving from Oracle 9i.

As illustrated in Figure 3.8, the RecordSet object provides a single method to access the Oracle9*i* database through either the Oracle ODBC or OLE DB provider. The ASP.NET environment provides the DataReader and DataSet objects as two separate methods to access images within Oracle9*i*. Furthermore, the DataReader and DataSet objects can each use three separate data access providers, namely the Microsoft .NET Framework Data provider for Oracle, the ODBC .NET data provider in conjunction with the Oracle ODBC driver, and the OLE DB .NET data provider in conjunction with the Oracle OLE DB driver. This provides a total of six methods in ASP.NET and two in ASP for image access and serving from the Oracle9*i* database using a combination of five driver/provider options. Sections 6.1 and 6.2

provide the performance results and a detailed discussion comparing these eight approaches used in ASP and ASP.NET Web to access images in the Oracle9*i*.

3.5 Analysis of various drivers supporting image serving from an Oracle9*i* database using JSP and Servlets.

This section introduces the Oracle JDBC drivers, their architecture and core functionality, and identifies the appropriate drivers supporting image serving using JSP and Servlet applications.

The JDBC standard was defined by Sun Microsystems, allowing individual providers to implement and extend the standard with their own JDBC drivers. Consequently, Oracle has developed customized JDBC drivers supporting the standard JDBC API. In addition, Oracle JDBC drivers have extensions to support Oracle-specific data types and to enhance the performance of Java applications. Oracle extensions to JDBC are captured in the package `oracle.jdbc` and `oracle.sql`. Both packages contain classes and interfaces that specify the Oracle extensions in a manner similar to the way the classes and interfaces in `java.sql` specify the public JDBC API. As the public JDBC API and Oracle extensions are available within all Oracle JDBC drivers it was decided that the research should focus on image serving from an Oracle9*i* database using JSP and Servlets in conjunction with the Oracle JDBC drivers.

It should be noted that other vendor-specific JDBC drivers are available for the Oracle 9*i* database. Complete lists of the drivers are provided on the Sun Microsystems Web site available at http://servlet.java.sun.com/products/jdbc/drivers (Sun Microsystems, 2003b).

Oracle provides the following JDBC drivers supporting image serving from an Oracle9*i* database (Oracle Corporation, 2001b). Please refer to section 3.3 for JDBC technology-based driver types.

Thin Driver: The Oracle JDBC Thin driver is a 100% pure Java, Type IV JDBC technology-based driver type. It targets Oracle JDBC applets, but can also be used for JSP and Servlet applications. Because it is written entirely in Java, this driver is platform-independent and does not require any additional Oracle software. The Thin driver communicates with the database using TTC (Two-task common), a protocol developed by Oracle to access the Oracle DBMS.

OCI Driver: The JDBC OCI driver is a Type II JDBC technology-based driver type and requires an Oracle client installation; it is therefore Oracle platform-specific. The OCI driver converts JDBC invocations to the Oracle OCI which are then sent to the Oracle database server. The OCI driver communicates with the server using the TTC protocol.

Oracle also provides the Internal Thin and Internal OCI Drivers which offer the same functionality as the OCI and Thin Drivers illustrated above, but run inside the Oracle database. The internal drivers were therefore excluded from the research as they do not function within the middle-tier of the three-tier test environment.

Figure 3.9, below, illustrates the driver-database architecture for the JDBC Thin and OCI drivers.

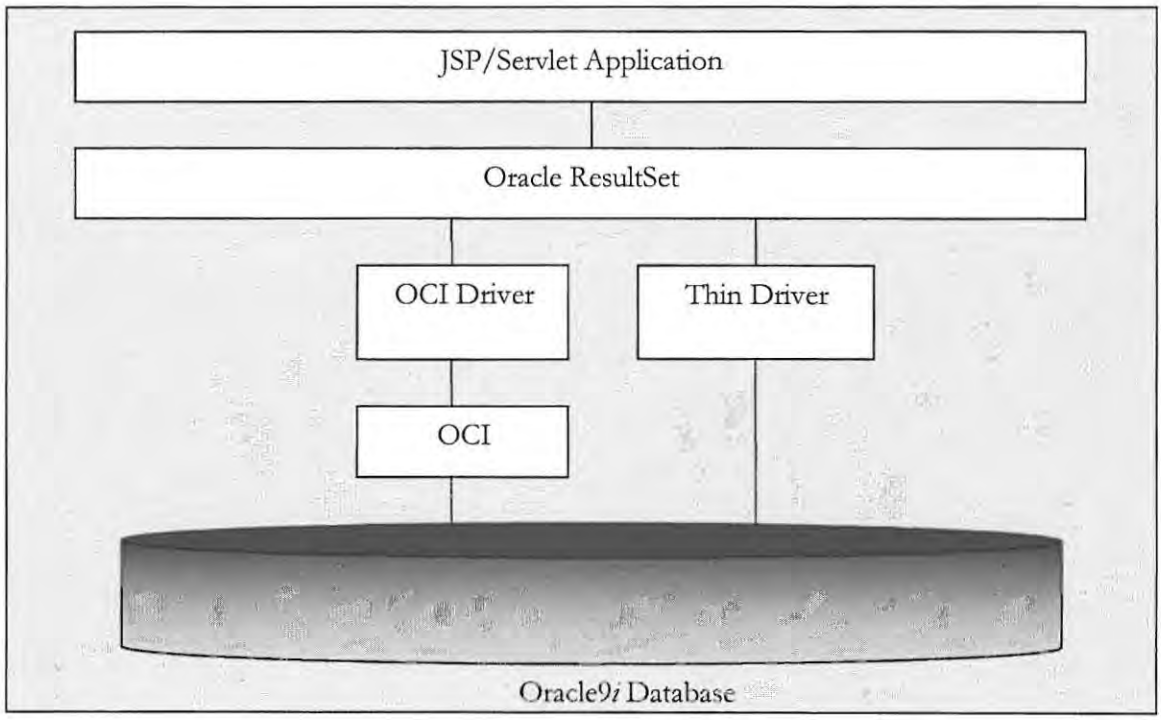

Figure 3.9 Oracle JDBC drivers supporting image access and serving from Oracle 9i.

As illustrated in Figure 3.9, the Java API and Oracle extensions provide two approaches to serve images from an Oracle9*i* database using the Oracle Thin and OCI Drivers. Section 6.3 provides the performance results and a detailed discussion comparing these two approaches used in the JSP and Servlet applications to access images in the Oracle9*i* database.

3.6 Summary

This chapter detailed an analysis of the various components used for the process of serving images within ASP, ASP.NET, JSP and Servlet applications. The various components and technology-specific database access technologies had been identified and then analysed to determine which options directly supported image serving from an Oracle9*i* database.

The data access technologies identified as directly supporting image serving from an Oracle9*i* database are as follows.

ASP The RecordSet object provides a single method to access the Oracle9*i* database through either the Oracle ODBC or OLE DB provider.

ASP.NET The DataReader and DataSet objects provide two separate approaches to access images within the Oracle9*i* database. Each object can use three separate data access providers, namely the Microsoft .NET Framework Data provider for Oracle, the ODBC .NET data provider in conjunction with the Oracle ODBC driver, and the OLE DB .NET data provider in conjunction with the Oracle OLE DB driver.

JSP and Servlets The Java API and Oracle extensions to the Java API provide two approaches to serve images from an Oracle9*i* database using the Oracle Thin and OCI drivers.

CHAPTER 4

DEVELOPMENT OF IMAGE SERVING APPLICATIONS

"There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

C.A.R Hoare

This chapter details the author's development of ASP, ASP.NET, JSP and Servlet prototype applications when serving images from an Oracle9*i* database to a client's browser. The steps taken during the development of the applications are described, focusing on the benefits and limitations encountered within each application environment. Particular emphasis is placed on technology-specific options considered when developing robust and scalable image serving applications.

In addition, this chapter describes the IDEs used for application development and details specific architectural features of each Web technology, such as caching mechanisms, error handling and ease of programming.

ASP, ASP.NET, JSP and Servlet applications follow a similar pattern when images are served from an Oracle9*i* database to a client's Web browser. The sequence of events can be divided into 4 stages.

1. Establish and open a connection to the Oracle9*i* database.
2. Query the database for single or multiple images and populate a server-side object with the images from the query.
3. Send the images to the client's Web browser.
4. Close the database connection and associated server-side objects.

The implementation of these stages in all the above mentioned Web technologies is investigated.

4.1 ASP Image Serving

As illustrated in figure 3.8, both the Oracle ODBC and OLE DB drivers allow for the access and retrieval of images from the Oracle9*i* database through ADO. Both drivers require installation on the physical machine running IIS. The four stages of serving images using an ASP application are described below.

- A connection to the Oracle9i database is established. To connect to an Oracle9*i* database, the ADO Connection object is instantiated and the `open` method is called.

  During application development a choice exists between a System DSN (Data Source Name) and System DSN-less connection when using the Oracle ODBC driver. A System DSN is a file containing connection information and is created through the ODBC Data Source Administrator within Windows XP. A DSN-less connection provides identical information, but instead of creating a DSN through the ODBC Data Source Administrator, the information is passed, as parameter details, to the ADO Connection object when it is opened.

  Microsoft acknowledges that the System DSN is slower than the DSN-less as it requires a registry lookup to determine the connection string's parameter details (Meier, 2000). Consequently, the DSN-less connection was selected to be used

in ASP applications when using the Oracle ODBC driver. The code segment below illustrates how the DSN-less connection was used during the development of the ASP prototype application.

```
Set swalesConnection = server.createObject("adodb.connection")
swalesConnection.open "Provider=OraODBC.Oracle;Data
Source=media_athena.ict.ru.ac.za;User
ID=mmedia;PASSWORD=mmedia"
```

It should be noted that the Oracle OLE DB driver does not use a System DSN and System DSN-less connection. The connection information is passed to the Connection object when the `open` method is called.

- The RecordSet object is instantiated after a connection to the Oracle9*i* database is established. Using the Connection object, a select SQL statement is executed which populates the RecordSet object with the associated data from the Oracle9*i*. All drivers specified in table 3.1 provide methods which allow the extraction of meta-data information associated with the image. For example, the following SQL statement can be executed to obtain meta-data information.

```
Set swalesRecordSet = dylanConnection.execute("SELECT
t.IMAGE.getmimetype() as MIMETYPE, t.IMAGE.getcontentLength()
as CONTENTLENGTH, t.IMAGE.getheight() as HEIGHT,
t.IMAGE.getwidth() as WIDTH FROM  LARGEPHOTOS t WHERE ID = 1")
```

To retrieve an image from the Oracle9*i* database, the Oracle ODBC and OLE DB drivers provide a `getContent()` method, which returns the image's data as a BLOB (Binary Large Object).

```
Set swalesRecordSet = dylanConnection.execute("SELECT
t.IMAGE.getContent() as CONTENT FROM  LARGEPHOTOS t WHERE ID =
1")
```

This method proved to be a deciding factor for selecting the Oracle ODBC and OLE DB drivers rather than the Microsoft ODBC and OLE DB drivers. The ASP runtime engine supports the BLOB data type, but not the Oracle ORDImage data type. When using the Microsoft ODBC and OLE DB drivers, without using the `getContent()` method, an error is generated, stating that the ASP runtime engine does not support the associated ORDImage data type. Consequently, the Microsoft ODBC and OLE DB drivers do not provide methods to retrieve an image.

- The ADO Response object is used to send the image (BLOB). The Response object provides a `BinaryWrite()` method which enables binary data to be written to the HTTP output stream. The following code segment illustrates the use of the BinaryWrite() method to output multiple images from the RecordSet object.

```
Do While Not myRecordSet.EOF
  CONTENT = myRecordSet ("CONTENT")
  Response.BinaryWrite( CONTENT )
myRS1.MoveNext
Loop
```

ASP applications use internal server buffers when writing data to the client. The default installation of IIS 5.0 enables buffering within all ASP applications, meaning that no data is sent to the client's browser until the entire ASP application has finished executing. If the `BinaryWrite()` method outputs data that exceeds the default maximum buffer size of 4 MB, the ASP error 251,

"*response exceeds the buffer limit*" is generated. This is naturally a problem when serving an image greater than 4 MB. It is therefore recommended to disable buffering or increase the maximum buffer size. A buffer size of 6 MB was used in this research for all ASP applications, as the largest image to be served was approximately 5 MB. As an aside, buffering was also disabled without any performance difference between the two approaches being recognised.

- Once the image is served to the client, the RecordSet object and Connection object must be closed explicitly to free the associated system resources.

Section 6.1 provides the performance data when using the Oracle ODBC and OLE DB drivers for image serving.

4.2 The ASP environment

The following observations about the ASP environment were noted by the author during application development and should be taken into consideration when developing ASP applications.

4.2.1 Caching mechanisms

As mentioned in section 2.2.1, an ASP file contains a mixture of scripting code and HTML. On activation, the ASP file is first interpreted, line by line, before it is executed by the ASP scripting engine. During execution, it is a mandatory task, within the research applications, to generate dynamic content, including the retrieval of images from the Oracle9*i* database, before the data is sent to the client's Web browser. ASP does not provide the ability to cache dynamic content which again includes any images that have been retrieved from Oracle9*i*.

The caching mechanisms of ASP reside within IIS. IIS contains two caches for ASP applications, the *File Template Cache* and the *Scripting Engine Cache*.

The *File Template Cache* contains a specified number of interpreted ASP files. By default, IIS keeps 256 files in the cache (Microsoft Corporation, 2004a). When IIS receives a request for an ASP file, it looks in the File Template Cache. If the file is not in the cache, IIS reads the file from disk, interprets the file, and places it in the File Template Cache.

The *Scripting Engine Cache* contains a specified number of ASP Scripting Engine instances. When IIS receives a request for an ASP file, it can access the script engine from the cache instead of starting a new instance of the script engine (Microsoft Corporation, 2004b). IIS keeps 30 instances of the script engine in the cache.

The number of files and script engine instances in the cache can be changed. Careful consideration must be taken not to make the cache too large as it will eventually consume too large an amount of memory, thereby decreasing the performance of other processes on the server machine. The caching options provided by IIS appear primitive, in that no dynamic content can be cached in ASP applications. Section 4.4.1 discusses the enhancements to the caching mechanisms in the ASP.NET environment, where the ability to cache dynamic content has become available.

4.2.2 Error handling and Tracing

In order to provide robustness to any Web application, there is a need to handle errors that may be generated during application execution. An error can occur for a variety of reasons and by default, when an error occurs within ASP, the ASP file stops executing and information on the error is sent to the client's browser. ASP provides two mechanisms to handle such errors.

The first is to write a line of code `On Error Resume Next` within each ASP file where one would expect an error to be generated. This line of code forces ASP to skip the generated error and to continue executing the script. At this point, the ASP *Err* object can be used to test if an error has occurred. This approach was used to handle errors in ASP applications, but it was found that large amounts of code needed to be

placed within the ASP files where an error can occur, making it extremely complex to follow and debug.

ASP also includes the *ASPError* object. Whenever an ASP error occurs, IIS generates an HTTP *500;100* error. The HTTP *500;100* error is a low-level error that occurs within IIS and, by default, is parsed by a server-side page that determines the error information to be sent to the client's Web browser. The default page that handles the HTTP 500;100 error was changed to point to a customized ASP error page. This customised ASP page was then used to access the ASPError object which provides specific information relating to an error, including the error number, the file in which the error occurred, and the line and column number where the error occurred.

Both approaches in ASP provide unstructured mechanisms to handle errors. The term "unstructured" is used mainly because either an error is skipped or information relating to the error is determined. The error could not be handled in a structured, error specific manner. For example, if a request is made for an image that does not exist within the Oracle9*i* database, the runtime engine generates an exception error. The error could not be handled appropriately by resubmitting another SQL query for a different image. Either the error is skipped or more detailed information about the error is provided to the client. Section 4.4.2 discusses the changes that have occurred in ASP for handling errors in a structured manner within ASP.NET.

During application development, there is often a need to monitor the values of various variables and functions. This process is termed tracing and is an important aspect of debugging applications. Due to its interpretive behaviour, ASP does not provide any tracing mechanisms and forces developers to write output variables during program execution. For example, a common technique in ASP was used, during which the `Response.Write()` method is used to output the value of variables during program execution. This process provided an ad-hoc manner to conditionally debug applications. Section 4.4.2 describes the enhancements made with regard to tracing applications within ASP.NET.

4.2.3 Ease of programming

An ASP file contains a mixture of HTML and scripting code and it is not uncommon for multiple ASP files to contain identical scripting code, which makes it difficult to follow and maintain across many ASP files within an application. For example, when connecting to an Oracle9*i* database, the ADO connection object is instantiated and opened. A change to the connection string requires the same change across multiple files.

ASP provides the ability to insert the content of one ASP file into another ASP file before it is interpreted, using the `#include` directive. However, the `#include` directive does not reduce the amount of code to be interpreted and it is still difficult to keep track of code changes across multiple ASP files. Essentially, ASP does not provide object-oriented capabilities such as the instantiation of classes to be used across multiple files. Section 4.4.3 discusses the changes to this model made available in ASP.NET applications where a developer not only can separate HTML from scripting code, but fully use the object orientated paradigm.

4.2.4 Observations on development tools

The IDE used for ASP application development was Macromedia Dreamweaver UltraDev Version 4.0. Dreamweaver supports visual programming for ASP and JSP applications. The following capabilities and features of Dreamweaver were identified when serving images from an Oracle9*i* database using ASP applications.

- A connection to the Oracle9*i* database could be established using either a System DSN or System DSN-less connection string for the Oracle ODBC driver. Additionally, the establishment of a connection could be dynamically tested within the IDE, without having to run the application through IIS. This provided a convenient and efficient mechanism for testing ODBC or OLE DB drivers without having to deploy the application.

- A RecordSet object could be created. A dialog box is provided which displays all the tables within the Oracle9*i* database and a SQL statement can be dynamically produced to query data. However, Dreamweaver does not support certain Oracle *inter*Media datatypes such as ORDImage. When attempting to select an image from Oracle9*i*, an error was displayed within the IDE, specifying that the datatype was invalid. An identical message is displayed when querying meta-data associated with the image.

- Oracle provides a plug-in for Dreamweaver UltraDev 4.0 with the capability to insert multimedia content into the ASP application. The plug-in is installed via the Dreamweaver Extension Manager and requires the Oracle ODBC or OLE DB drivers to be installed on the physical machine. The plug-in supports all *inter*Media object types and provides access to the meta-data associated with an image.

An observation about Dreamweaver, as well as the Oracle plug-in, is that the code is automatically generated by the visual programming processor, which is difficult to trace and includes features that are not always necessary. This makes it difficult to modify the code once it has been generated, although it does provide an ideal environment for developers who are not familiar with ASP programming techniques.

It should be noted that Version 1.0.0 of the Oracle *inter* Media Extension supports Dreamweaver UltraDev version 4 on Windows 2000 operating system and is not currently supported with the latest Dreamweaver version, namely Dreamweaver MX.

4.3 ASP.NET Image Serving

As illustrated in figure 3.8, the ODBC, OLE DB, and Microsoft .NET Framework providers allow for the access and retrieval of images from the Oracle9i database through the DataReader and DataSet objects within ADO.NET.

An installation of the .NET Framework is required to run any ASP.NET application. The .NET Framework has two main components: the *Common Language Runtime* (CLR) and the .NET Framework *Class Library*.

The CLR is the foundation of the .NET Framework, managing code at execution time whilst providing core services such as garbage collection, thread management etc (Platt, 2001). The *Class Library*, the other main component of the .NET Framework, is hierarchically grouped into namespaces, which includes classes, interfaces, and value types that provide access to system functionality (Platt 2001). For example, *System.Data.OleDb* represents the *OleDb* type used for the .NET Framework data provider for OLE DB.

The .NET Framework provider for OLE DB is included within the .NET Framework (Version 1). The ODBC and Microsoft .NET Framework providers are add-on components to the .NET Framework, which required separate installations. In addition, the ODBC and OLE DB providers require the Oracle ODBC and OLE DB drivers to be installed on the same machine.

The approach used by the author to serve images from an Oracle9*i* database, using the ADO.NET DataReader object is described below:

- A connection is established to the Oracle9*i* database. When using the ODBC Framework provider, again the research had the choice of using a DSN or DSN-less connection string, referencing the Oracle ODBC drivers. As with ASP connections, best performance is obtained with a DSN-less connection string and was therefore used throughout the research when developing ASP.NET applications. The Microsoft .NET Framework and OLE DB providers provide connection information when the Connection objects `open` method is called. The code segment below illustrates how a connection was established using the Microsoft .NET Framework provider. The other providers follow a similar method to establish a connection to the Oracle9*i* database.

```
Dim swalesConnection As OracleConnection
Dim connectionString As String
connectionString = "Data Source=media_athena.ict.ru.ac.za;User
ID=mmedia;Password=mmedia;"
swalesConnection = New OracleConnection(connectionString)
swalesConnection.Open()
```

- The DataReader object is instantiated once the connection is established. The Command object, which holds the SQL query, is executed to populate the DataReader with the image as a BLOB data type. The SQL statement used to query the database remains the same as ASP, supporting the `getContent()` method and all other methods for meta-data extraction.

- The `BinaryWrite()` method, defined in the HttpResponse class, is used to output the BLOB as binary characters to the HTTP output stream. The System.Web.HttpResponse class encapsulates HTTP response information from an ASP.NET operation. The code segment below illustrates how the `BinaryWrite()` method is used to output the images from the DataReader object.

```
Dim swalesCommand As OracleCommand = New OracleCommand("SELECT
t.IMAGE.getContent() as CONTENT FROM LARGEPHOTOS t  WHERE id
=1", swalesConnection)
Dim swalesdr As OracleDataReader
      swalesdr = myCommand.ExecuteReader
While swalesdr.Read()
      Response.BinaryWrite(dr.GetValue(0))
End While
```

There seemed to be no limitation with the amount of data that could be sent to the output stream when using the `BinaryWrite()` method in ASP.NET. When using the DataReader object to retrieve a large BLOB, it is recommended to pass `CommandBehavior.SequentialAccess` to the

`ExecuteReader` method call: Because the default behaviour of the DataReader is to load an entire row into memory with each read, large amounts of memory can be consumed by a single BLOB. `SequentialAccess` sets the behavior of the DataReader to load only the data requested (for example, by means of the `GetBytes` method, which controls the number of bytes to read). During application testing, there seemed to be no significant performance increase when using `SequentialAccess`. This could be due to the fact that the images were never larger than 5MB.

- Once the image is served to the client, the DataReader and Connection objects must be closed explicitly to free the associated system resources.

The approach used by the author to serve images from an Oracle9*i* database, using the ADO.NET DataSet object is described as follows. Note that the presence of the DataSet object introduces an extra step in the standard 4 step process for image serving applications.

- A connection is established to the Oracle9*i* database.

- An instance of a DataAdapter object is created once the connection is established. The constructor for the DataAdapter takes two parameters. The SQL query and a reference to the instantiated Connection object. The code segment below demonstrates how the DataAdapter object was used during application development. The DataAdapter object is filled with a single image from the Oracle9*i* database.

```
Dim cmdImages As OleDbDataAdapter
        cmdImages = New OleDbDataAdapter _
        ("SELECT t.IMAGE.getcontent() as CONTENT FROM
LARGEPHOTOS t WHERE id =1", swalesConnection)
```

- An instance of the DataSet object is then created and the DataAdapter's `Fill` method is used to populate the DataSet with the results from the SQL query. The Fill method takes two parameters: a DataSet instance and a string. The DataSet instance represents the DataSet to be filled, and the string identifies the DataTable to be created inside the DataSet. The code segment below demonstrates how we created a DataTable instance, called images, within the DataSet.

```
Dim swalesds As DataSet = New DataSet()
cmdImages.Fill(swalesds, "images")
```

- The image is sent to the client using varying approaches. As mentioned in section 3.2, ASP.NET provides the DataView object and several list-bound controls to display data held in the DataSet. The DataView object can be used to present a subset of data from a DataTable. For example, if we wanted to display the first 10 records of a DataTable, the DataView object can be used to filter the DataTable, before it is displayed.

  It is also possible to bind a list-bound control, such as the *DataGrid* or *DataRepeater*, directly to a DataTable, without using a DataView. For example, when a DataTable is directly bound to a DataGrid, ASP.NET generates an HTML output table, filled with data from the DataTable

  However, during application development, it was established that the direct binding of an image, stored as a BLOB within a DataTable, to a list-bound control was not possible. This was because the list-bound control does not naturally understand that the BLOB is to be displayed as an image. The detection of an occurrence of a BLOB, before the DataTable was bound, was necessary. Once the BLOB was detected, an image tag could be placed within the list-bound control to display the image. However, the image tag required a

separate request for the specified image from the database, because it was not possible to use the image (BLOB) already in the DataTable. As a result, the direct binding of a DataTable, containing a BLOB, requires two requests for the identical image from the database. List-bound controls are therefore not recommended for direct data binding to DataTables for image serving.

As an alternative approach to prepare to send the image, an attempt was made to access the image at a specified coordinate within the DataTable. Each DataTable has a *DataRow* and *DataColumn*. To access an image BLOB value in a DataTable, the DataRow and DataColumn was specified, after which it could be served using the `BinaryWrite()` method. The code segment below illustrates how we accessed the BLOB image within the DataTable instance, called images.

```
Dim tmp() As Byte
Dim dt As DataTable
Dim dr As DataRow
Dim dc As DataColumn
    dt = ds.Tables("images")
      dc = dt.Columns(0)
      dr = dt.Rows(0)
      tmp = (dr(dc))
    Response.BinaryWrite(tmp)
```

This method was identified as the most optimal approach to access an image (BLOB) within a DataSet.

- The connection to the database can be closed after the data has been sent to the client's browser or once the DataSet object has been populated with results from the database. This demonstrated the disconnected model of the DataSet object. While garbage collection eventually cleans up objects and therefore releases connections and other managed resources, garbage collection only occurs when it is needed and handled implicitly by the .NET Framework (Platt

2001). It is a developer's responsibility to make sure any expensive resources are explicitly released. It should be noted that garbage collection is not offered in ASP.

4.4 The ASP.NET environment

The following observations were noted by the author and should be taken into consideration when developing ASP.NET applications.

4.4.1 Caching mechanisms

As mentioned in section 4.2.1, ASP does not provide any native dynamic data caching capabilities, but relies on the *File Template Cache* and the *Scripting Engine Cache* contained within IIS. ASP.NET provides two mechanisms for caching and improved performance, namely the ASP.NET execution model and intrinsic caching capabilities.

Unlike ASP, ASP.NET files are not interpreted line by line and then executed. When an ASP.NET file is requested for the first time it, is interpreted and then compiled, by a *Common Language Runtime* (CLR) compliant compiler, into an intermediate language, stored within an assembly cache. The ASP.NET runtime engine then executes the file and sends HTTP output to the browser. Subsequent calls to the same ASP file are directly accessed within the assembly cache and executed, effectively bypassing the interpretation and compilation of the file. This process provides a great improvement over traditional ASP, where IIS cached interpreted ASP files, but never compiled and cached the file in server memory.

ASP.NET has also made available three different types of intrinsic caching mechanisms, which, when appropriately used, can increase the overall performance of ASP.NET applications. These cashing mechanisms are *output*, *fragment*, and *data* caching (Ahmed *et al* 2002).

Output caching provides the capability to cache response content generated from the execution of ASP files. An image requested from an Oracle9*i* database would therefore be included within the cache, eliminating the time required for database interaction, on a successive, identical request. This method provides the greatest performance increase, but can only be used when the requested image is not expected to change within a particular time frame. During application deployment a dramatic increase in the performance of ASP.NET applications was experienced when using output caching, but it is recommended to be used only when the content of the database query is not expected to change over short periods of time.

Fragment caching allows for the caching of portions (fragments) of response content generated from the execution of an ASP.NET file. Fragment caching is implemented by separating user controls out of the main ASP.NET file, and assigning caching parameters to each user control. For example, section 4.3 illustrated that to insert dynamically an image into a DataGrid list-bound control one needs two separate requests for the image from the Oracle9*i* database. However, it is possible to fragment cache the request for the image, allowing for the dynamic generation of the DataGrid, effectively reducing database interaction to a single request.

Data caching, provides the ability to cache individual objects. This method makes for an easy-to-use temporary data storage area while conserving server resources by releasing memory as cached objects expire.

A major consideration in planning a caching strategy within ASP.NET, is the appropriate utilization of server resources. For example, the research used output caching for an ASP.NET file that served a single image of 5MB. The ASP.NET file and 5MB image are stored in server memory. If 100 identical ASP.NET files used output caching for different images of the same size, more than 500MB of server memory would be consumed, leaving minimal space for other server processes. Consequently, there is a trade-off when using the caching mechanisms described above, in that for every item cached less memory is available for other uses. The

author recommends extensive testing of various caching mechanisms to be completed within all ASP.NET applications before deployment.

4.4.2 Error Handling and tracing

Unlike the unstructured approach for handling errors in ASP, ASP.NET supports structured error handling, using the *try-catch-finally* construct. The *try-catch-finally* construct allows to effectively "catch" different forms of errors and respond to them appropriately. For example, when a request for an image that did not exist in the test Oracle9*i* database was made, an `InvalidOperationException` error was thrown by the ASP.NET runtime engine. It was possible to catch this exception and handle it appropriately by resubmitting a request for a different image from the database without the need to display error information to the client. In addition, multiple runtime exceptions could be caught and handled within the try-catch-finally construct.

ASP.NET also includes a Trace class which enabled the viewing of diagnostic information about a single request for an ASP.NET file. The `Trace.Write()` method could be used to output diagnostic information relating to variables and function calls during page execution. ASP.NET appends a series of diagnostic information tables immediately following the output of the application data, which includes custom information relating to variables and function calls. The trace information can also be written to a log file instead of being sent as output to a browser, and when disabled, no trace information is sent to the client, eliminating the need to remove statements from an application when it is deployed to production.

4.4.3 Ease of programming

Many features have been introduced within ASP.NET to decrease the time taken to deploy Web applications and ease the complexity of application development. Through multiple experiences of application development, the following features were

identified which improve the ease of programming within ASP.NET applications, as opposed to ASP.

- Unlike ASP, which supports only scripting languages such as VBScript or Jscript, ASP.NET supports more than 25 .NET languages. This provides flexibility to a variety of programmers who are familiar with concepts and programming techniques used in a specific language, ranging from Java (which is closely matched by C#) to Visual Basic.

- Another feature of ASP.NET allowed for the elimination of interspersing HTML and code within a single file. In ASP.NET, two separate files were created, one which defines the user interface, and the other for server-side programming logic (called a code-behind class). In addition, the .NET Framework containes thousands of classes we could employ within the code-behind file or create new classes to be instantiated within the file. This object-oriented programming feature eases application development, eliminating the need to use the `#include` directive used in ASP.

4.4.4 Observations on development tools

The IDE used for developing ASP.NET applications was Visual Studio .NET. Visual Studio .NET, a visual programming tool, provides the following capabilities with specific reference to image serving from an Oracle9*i* database:

- A connection to the Oracle9*i* database could be established, using the .NET Framework provider for OLE DB. Visual Studio .NET did not provide the ability to create a database connection using the .NET Framework provider for ODBC and the Microsoft .NET Framework provider for Oracle.

- An ADO.NET DataReader and DataSet object could be created and it was possible to automatically populate each object with relational data from an Oracle9*i* database. On the other hand, Visual Studio .NET did not support the

Oracle ORDImage data type and therefore could not populate images into ADO.NET objects. In addition, the direct data binding of a list-bound control to a DataSet was not possible.

Key features of Visual Studio .NET that provided ease of development and deployment for image serving included:

- **Predictive text:** When referencing the class libraries of ASP.NET, Visual Studio .NET provides a predictive text approach when referencing types or classes when writing code.

- **Debugging:** Visual Studio .NET contains a rich set of debugging tools, which were utilized to debug ASP.NET applications. *Breakpoints* could be placed into specified areas of code, enabling the trace and flow of the ASP.NET application during run-time. The Visual Studio debugger provides different kinds of *breakpoints*, which enabled the trace of variables within general code or within function calls. Once a breakpoint was reached, it was possible to continue with application execution whilst examining the values and variables in a separate window.

  This feature dramatically increases the efficiency of program development in ASP.NET applications as there was no longer a need to write responses out to the browser when tracing applications.

4.5 JSP and Servlet Image Serving

JSP and Servlet technologies use similar principles for developing applications. Servlets provide a component-based, platform independent mechanism for building Web applications (Prasad *et al* 2000). They are essentially Java classes that extend Web Server functionality by handling requests from a client's browser and generating an appropriate response. A Servlet takes an HTTP request from a browser

(HttpServletRequest), performs any database interaction, generates dynamic HTML content, and then returns an HTTP response (HttpServletResponse) to the client's browser.

The HttpServletResponse interface defines methods that allow developers to retrieve certain types of output streams. To send character data, a developer must use the `PrintWriter` returned by the `getWriter` method. To send binary, a developer must use the ServletOutputStream, or an extended version of it, returned by the getOutputStream. What is important to note about Servlets is that they run inside a container (i.e. Apache Tomcat), providing access to the entire family of Java API's including the extended JDBC API provided by Oracle.

The JSP technology, which abstracts Servlets to a higher level, is an open, freely available specification developed by Sun Microsystems, providing an alternative to Servlets (Mahmoud, 2003). JSP's are a direct extension of Java Servlets (although they are directly interoperable), and provide a way to separate content generation from content presentation. For example, during Servlet construction a developer is responsible for generating HTML and dynamic content using Java code, whereas JSP technology allows for the separation of HTML and dynamic content. JSP is similar to ASP where the Web file contains a mixture of HTML and server-side code.

It should be noted that developers have access to certain implicit objects that are available for use in JSP files, without being declared first. For example, the response object is implicitly created and references the same HttpServletResponse explicitly created within Servlets. The JspWriter is another implicitly created object which writes textual data to the HTTP output stream. This object will be discussed later as it causes a runtime exception when using other output streams to write binary data.

Because of the similarity of JSP and Servlet applications, the process of image serving will be discussed together. Separate reference will be made to JSP or Servlet application where appropriate.

As illustrated in figure 3.9, the Oracle OCI and Thin JDBC drivers allow for the access and retrieval of images from the Oracle9*i* database using JSP and Servlet applications.

To execute Servlet or JSP applications, the Oracle JDBC classes must be deployed to a specified directory of the Web server.

The steps used by the author to serve images from an Oracle9*i* database, using the Oracle JDBC driver and ResultSet object are described below.

- A connection to the Oracle9*i* database is established. This was accomplished by registering the JDBC driver with the DriverManager class. The connection string specifies the driver type to be used when making the connection (JDBC OCI Driver or JDBC Thin Driver). The code extract below shows how a connection was established using the JDBC Thin driver. To reference a JDBC OCI database connection, the keyword *thin*, is replaced with *oci*.

```
String JDBC_CONNECT_STRING ="jdbc:oracle:thin:@  (DESCRIPTION
=(ADDRESS_LIST =(ADDRESS = (PROTOCOL = TCP)(HOST =
146.231.121.147)(PORT = 1521)))(CONNECT_DATA = (SID =
media)(SERVER = DEDICATED))) ";
String JDBC_USER_NAME = "mmedia";
String JDBC_PASSWORD  = "mmedia";

Connection swalesConn;
DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver() );
swalesConn = DriverManager.getConnection( JDBC_CONNECT_STRING,
JDBC_USER_NAME, JDBC_PASSWORD );
```

The above code specifies details corresponding to a particular instance of the Oracle9*i* database connection. One such detail is the host address of the machine running Oracle9*i*. The host address can be specified as a domain name (e.g. athena.ict.ru.ac.za) or corresponding IP address (e.g. 146.231.121.147). The host address should not be used during application testing when the server machine is isolated from the external network. This is due to Apache Tomcat

requiring a lookup on the DNS (Domain Name Server) to map the domain name to the corresponding IP address.

- An Oracle ResultSet object is then instantiated and populated with the relevant images by calling the `executeQuery` method of the PreparedStatement object. The PreparedStatement object holds the SQL statement used to query the database. The code segment below illustrates how an image from the Oracle9*i* database was placed into the Oracle ResultSet object.

```
OraclePreparedStatement stmt;
OracleResultSet swalesRSet;
stmt =
(OraclePreparedStatement)swalesConn.prepareStatement("select
IMAGE from smallphotos where id =1" );
swalesRSet = (OracleResultSet)stmt.executeQuery();
```

The Oracle extensions to the JDBC API provide support for Oracle *inter*Media data types. Consequently, when using the SQL query it was possible to directly select an image as an ORDImage data type to be populated within the Oracle ResultSet object.

Once the ResultSet is populated, its content was served to the client using various techniques.

The first technique used during application development was the incorporation of the Oracle *inter*Media Java classes, supplied by Oracle for use in JSP and Servlet applications. The classes are placed as a package within the shared libraries folder of Apache Tomcat and can be utilized within any JSP or Servlet application. The Oracle *inter*Media Java classes are available in two versions, one for Servlet applications (OrdHttpResponseHandler class), and the other for JSP applications (OrdHttpJspResponseHandler class).

The OrdHttpResponseHandler class facilitates the serving of multimedia data from an Oracle9*i* database to the client's browser in Servlet applications. The class could be instantiated within any Servlet application and provided a `sendResponse` method for sending the image/s to the HTTP Servlet binary output stream. The class uses a default buffer size of 32768 bytes which could be manually changed. It should be noted that there was no significant increase in performance when the size of the buffer was increased. The following code illustrates how the OrdHttpResponseHandler class was used when serving multiple images from an Oracle9*i* database in Servlet applications.

```
OrdHttpResponseHandler dylanhandler = new
OrdHttpResponseHandler( request, response );
OrdImage image;
        while ( swalesRSet.next() )
        {

         image = (OrdImage)swalesRSet.getCustomDatum(1,
OrdImage.getFactory());
         swalesHandler.sendResponse("image/jpeg",
img.getContentLength( ),img.getContent( ), img.getUpdateTime(
));

        }
```

The OrdHttpJspResponseHandler class provides the same functionality and methods as the OrdHttpResponseHandler class, but is designed for use in JSP applications. When instantiating the class within a JSP file it must be included as a JavaBean. JavaBeans is a portable, platform-independent software component model, which enable developers to write reusable components (DeSoto, 1997). A more detailed discussion on the use of JavaBeans can be found in section 4.6.3. The code segment below illustrates how the OrdHttpJspResponseHandler class was used when serving multiple images from an Oracle9*i* database using JSP applications.

```
<jsp:useBean id="swalesHandler" scope="page"
class="oracle.ord.im.OrdHttpJspResponseHandler"/>
...
swalesHandler.setPageContext(pageContext);
while ( swalesRSet.next() )
        {
image = (OrdImage) swalesRSet.getCustomDatum( "image",
OrdImage.getFactory() );
swalesHandler.sendResponse("image/jpeg",
image.getContentLength( ), image.getContent( ),
image.getUpdateTime( ));
        }
```

As previously mentioned, the JSP specification states that the JspWriter object is implicitly created. The JspWriter sends textual data to the HTTP output stream, which means that developers need to explicitly create a binary output stream for sending non-relational data, such as images.

The `OrdHttpJspResponseHandler` class calls the `JspWriter.clear` method to clear the output buffer of the file prior to obtaining the binary output stream. During application deployment it was identified that the Apache Tomcat JSP runtime engine did not support access to the binary output stream when the JspWriter object was still in use. As a result, an IllegalStateException was thrown, stating that an output stream had already been called for the response (i.e. a JspWriter and ServletOutputStream cannot share the same response). Although the exception is thrown and has been documented by Oracle, Apache Tomcat seemed to handle the exception and opened the binary output stream to send the image (Oracle Corporation, 2002b) (Tice, 2000).

Because of the exception generated by the JSP runtime engine, it was decided to use a different approach to serve images from the Oracle9*i* database, without using the OrdHttpJspResponseHandler class. The following code represents the author's alternative to using the OrdHttpJspResponseHandler class in JSP applications.

```
response.setContentType("image/jpeg");
Blob temp;
OutputStream o = response.getOutputStream();
byte[] buf = new byte[32 * 1024];
while ( rset.next() )
{
 image = (OrdImage)rset.getCustomDatum( "image",
OrdImage.getFactory() );
  temp = image.getContent();
  buf = temp.getBytes(1,(int)temp.length()+1);
  o.write(buf);
}
o.flush();
o.close();
```

This approach obtains the output stream to send the image in an array of bytes. Apache Tomcat throws an identical IllegalStateException, but the image is still sent to the client's browser and the exception seems to be handled in the same manner when using the OrdHttpJspResponseHandler class. There also seemed to be no significant difference in performance when using the OrdHttpJspResponseHandler class over the author's code above.

- All associated server object must be explicitly closed once the images have been served to the client's browser.

4.6 JSP/Servlet environment

The following observations were noted during the development of prototype applications and should be taken into consideration when developing JSP and Servlet applications.

4.6.1 Caching mechanisms

From the findings of the research there are no inherent dynamic caching capabilities of JSP and Servlet applications. The caching mechanisms of JSP reside within the execution model which occurs as follows (reference will be made to Servlet files when appropriate).

- When a JSP file is requested, it is translated and then compiled into a Servlet class file (by the JSP engine). At this point, compiled JSP files and Servlets are identical and are handled in the same manner.

- The Servlet runtime engine then loads and executes the Servlet class file. During execution, the Servlet class communicates with Oracle9*i* using the Oracle JDBC drivers for SQL operations.

- Finally, dynamically generated content is output back to the client's browser. The next time the JSP file is requested, the Servlet engine executes the already-loaded Servlet, unless the JSP file has changed in which case it is automatically recompiled into a Servlet and executed.

From the process described above, JSP and Servlet files are executed in the same manner. Essentially, the only difference is the first compilation step required to change a JSP file into a Servlet class. Chapter 6 details the performance results of the JSP and Servlet applications to determine if there is any significant difference between the executed applications.

4.6.2 Error Handling and tracing

JSP and Servlet applications provide a structured approach to handle exceptions when a run-time error is generated. Both JSP and Servlet applications provide the try-catch-finally construct to handle multiple exceptions. It should be noted that the try-catch-finally construct was first adopted for use in Java and was only later introduced into .NET.

Another approach that was utilized in JSP applications, was to set the *errorPage* parameter of the JSP *page* directive to the name of a separate JSP file, which displayed a message to the user when an exception occurred. From this file, it was possible to access the exception object to retrieve information about the exception that was generated. For example, it was possible to inform the client that the selected image was

no longer in the database (for a specific select SQL query), and that he/she could resubmit a request for a different image.

Apache Tomcat also prints out valuable information from the stack holding the trace. When an exception occurs, the error message together with the relevant stack segment can be displayed within the client's browser or saved to a file.

4.6.3 Ease of Programming

As with ASP files, JSP files contain a mixture of HTML and Java, which can become difficult to follow and reuse if no clear separation is made between the two. The following options were used to separate page presentation from content generation in JSP applications.

- An include mechanism could be used to insert content from a specified file into a JSP document. This mechanism is similar to the `#include` directive in ASP applications. JSP, however, provides two variations: The first *include* directive includes the content of the specified file while the JSP document is being compiled into a Servlet. The second *include* directive includes the response generated after a specified JSP document is executed. Sun Microsystems recommends using the first include directive as it yields better performance but should only be used if the included file does not often change (Mahmoud 2003). It should be noted that the above recommendation was supplied by Sun Microsystems and no attempt was made during application development to use the include directive. Instead, JavaBean components were used.

- A JavaBean provided a mechanism to separate page presentation from content generation for use in JSP applications. In the context of JSP files, JavaBeans are classes that return data to a JSP file. This provides an object-oriented technique whereby we could make use of reusable components across many JSP applications. For example, when using the OrdHttpJspResponseHandler class, the class was included as a JavaBean inside the JSP file. Once the

JavaBean was included, the methods defined by the OrdHttpJspResponseHandler class could be used. This clearly demonstrates a better approach to handle complex applications when compared to ASP.

- Servlet applications seem to be more difficult to program as we were responsible for writing dynamic content and HTML using Java code. We were able to instantiate classes within Servlet applications but the greatest benefit of Servlet applications was the control we had over the application environment. For example, different types of output streams can be opened in Servlet applications depending on the data to be sent to the client (see section 4.5).

4.6.4 Observations on development tools

The IDE used for developing JSP applications was Macromedia Dreamweaver. Comments on the use of this IDE can be found in section 4.2.4.

4.7 Summary

This chapter detailed the author's development of ASP, ASP.NET, JSP and Servlet prototype applications when serving images from an Oracle9*i* database to a client's browser. The steps taken during the development of the applications were described, emphasising certain benefits and limitations that were encountered during application development. A discussion was also presented, describing certain features within each technology.

ASP.NET was identified as the only technology that provides inherent mechanisms to cache dynamic database content whilst ASP.NET, JSP, and Servlets provide more advanced features and functionality in terms of ease of programming, error handling, and tracing than ASP.

The IDEs used for application development were also detailed with respect to image serving applications.

CHAPTER 5

ENVIRONMENT AND METHODOLOGY FOR PERFORMANCE TESTING

"Every benchmark, no matter how well or how poorly designed, models a very particular usage pattern and workload. Organizations with similar workloads will find the benchmark details and results highly valuable; organizations using different application architectures will know how to read the results in context. In both cases, full disclosure is key."

Timothy Dyck

One of the objectives of this thesis is to provide a performance comparison of JSP, Servlets, ASP, and ASP.NET with regard to image serving over a network. The comparison is accomplished by developing prototype applications, testing them, and eliciting the foremost factors contributing to the final performance and suitability of the different technologies. The metric used for the performance comparison is the response time for image retrieval, that is, the amount of time taken to retrieve a certain number of images from an Oracle 9i database until they are displayed within a client's Web browser.

Figure 5.1 depicts the three-tier laboratory environment used to conduct the performance comparisons.

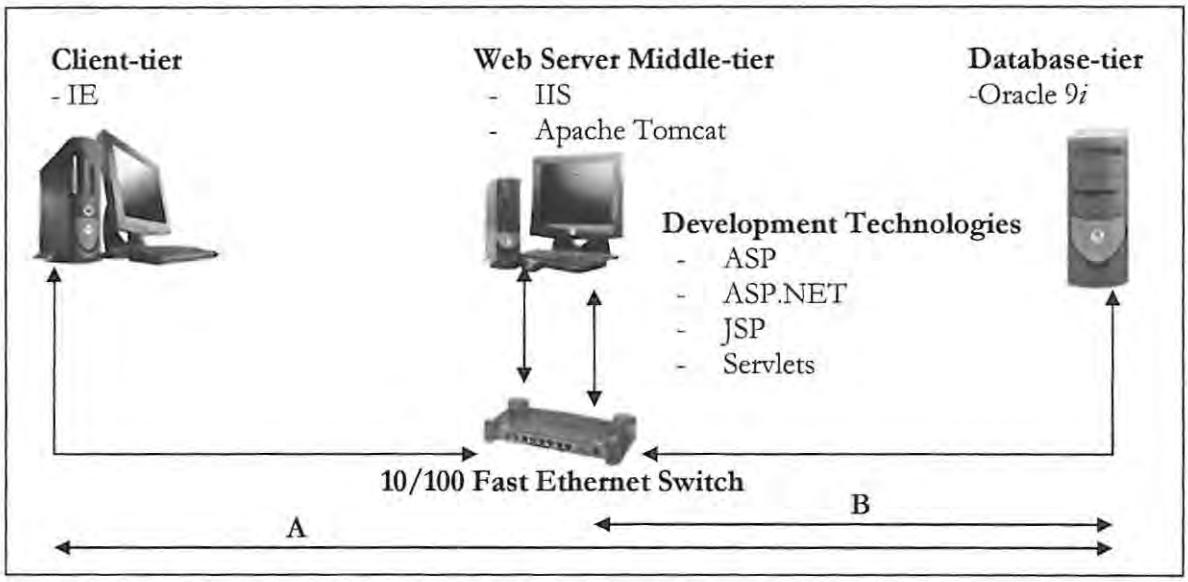

Figure 5.1 Test environment used for image serving performance tests

A brief description of the application testing methodologies is presented followed by a detailed description and discussion of each component within the laboratory environment, referenced in figure 5.1.

5.1 Application Testing Methodology

Two primary performance application tests were conducted in the research. The application tests were designed to assess a specific area of image serving from an Oracle9*i* database. Application test 1 consists of a single image serving methodology and application test 2 consists of a multiple image serving methodology.

Application test 1: Single image serving using ASP, ASP.NET, JSP, and Servlets

Application test 1 provides a performance comparison of the various Web development technologies used to serve a single image from the Oracle9*i* database. The application test consists of 4 individual tests whereby the size of the image to be served is

incremented. For example, the first application test serves a single image of approximately 50KB, followed by an increasing size of 1MB, 2.2MB, and 5MB image. Various test results depict how the size of an image affects the response time performance of the various technologies

Application test 2: Multiple Image serving using ASP, ASP.NET, JSP and Servlets

Application test 2 provides a performance comparison of the various technologies with regard to multiple image serving from the database. Again, the application test consists of 4 individual tests, but instead of increasing the size of the image, the number of images to be served from the database is increased. For example, the first application test serves a single image followed by an increasing number of 50, 100, and finally 250 images. Each image retrieved from the database is approximately the same size. Various results are presented to demonstrate how the number of images served affects response time performance.

In addition, each of the 4 tests within the 2 applications consists of 15 successive experiments to establish a mean. For example, within the single image serving application, a 50KB image is accessed and served over 15 successive requests. An example graph depicting the 15 experiments, using the ODBC driver in an ASP application is illustrated in Figure 5.2.

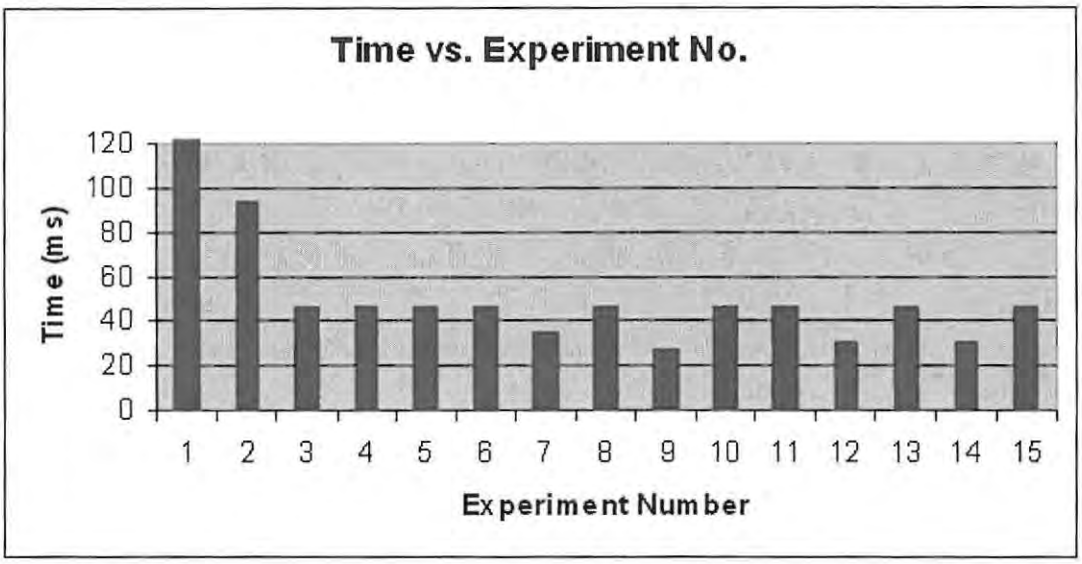

Figure 5.2 Time vs. Experiment Number – Single 50KB image served using ASP/ODBC

The mean is calculated from experiment number 3 to 15. Results from experiment 1 and 2 are not included in the calculation of the mean. Each experimental test is preceded by restarting the database and Web server machines, repeated observations indicate that the first two experiments yield inconsistent and erratic results. It is unlikely that, during normal operation, the database and Web server would be re-started sufficiently regularly for these results to be included in the analysis.

Once the mean is calculated for each of the four tests, a final graph is presented to depict the mean of each test within the application. An example graph depicting the single image serving application, using the various drivers in ASP is illustrated in Figure 5.3.

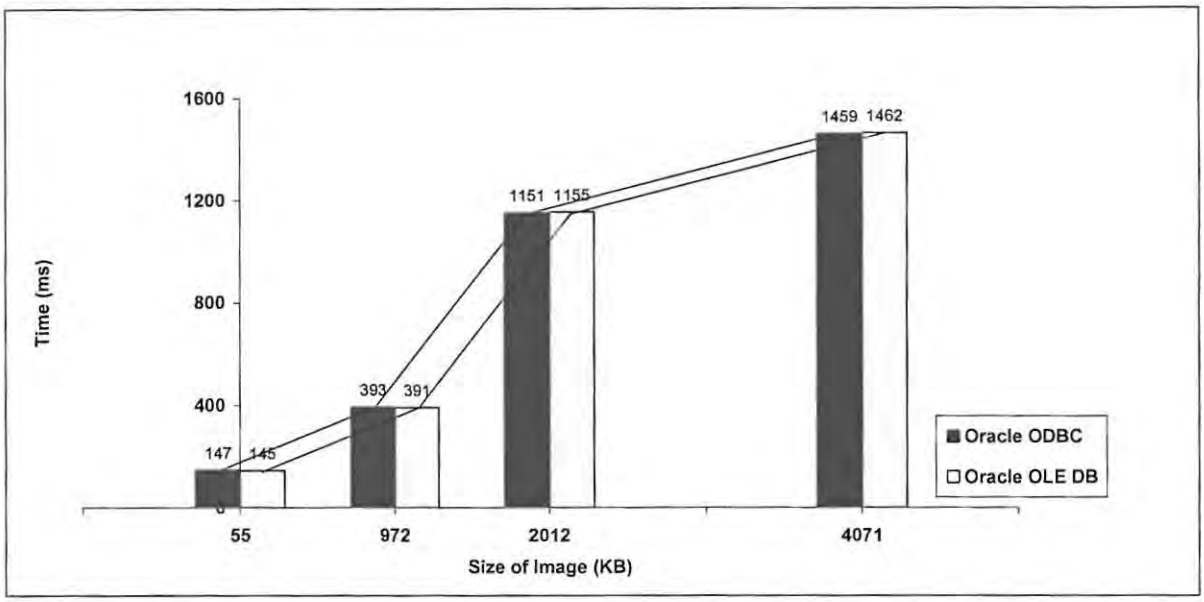

Figure 5.3 ASP single image serving results

5.2 Test Environment

With reference to figure 5.1, the performance tests were conducted in a closed environment: the database, Web servers, and client machines constituted an isolated LAN (Local Area Network). The justification for the isolated network was to eliminate any external traffic, from the Rhodes University network, that may influence response times. Consequently, all three test machines are connected to a 10/100 Ethernet switch.

The hardware and software specifications for the test machines are illustrated in Table 5.1 below:

	Database	Web Server	Client
CPU	Dual Intel PIII 800Mhz	Intel P4 1.7GHz	Intel P4 1.7GHz
Memory	1000 MB RAM	512MB RAM	512MB RAM
Operating System	Microsoft Windows 2000 Server (Version 5.0.2.1.9.5)	Microsoft Windows XP 2002	Microsoft Windows XP 2002
Software	Oracle 9*i* Release 1 (9.0.1.1.1)	.NET Framework (Version 1.0.3705)	Microsoft Internet Explorer (Version

	6.0.28)
Software	Internet Information Services (Version 5.1)
Software	Microsoft Visual Studio.NET (Version 7.0.9466)
Software	Apache Tomcat (Version 4.1.24)
Software	J2SDK (Version 1.4.1_01)

Table 5.1 Hardware and Software specifications for the test machines

The hardware and software specifications remained identical throughout application testing, with exception to the Web Servers. ASP and ASP.NET applications were run using IIS and the JSP and Servlets were run using Apache Tomcat.

5.2.1 Oracle 9*i* Database Server

Once installed and configured, the Oracle 9*i* database remained unchanged throughout application testing. Before any tests were performed, the database machine was restarted to ensure that CPU usage by other processes on the machine was at a minimum.

There are four tables within the database storing various types of images. All tables remain consistent with regard to the format, storage model, and data types. The image content together with its meta-data is stored in the database, within the local database tablespace. The image content is stored as an *inter*Media data type of ORDImage, referenced from the ORDSYS schema. The ID, or data type number, is used to uniquely identify the image within the table. The description and location, or data type varchar, is used to give a brief description of the image and its source location. The image size varies from 50KB up to 5MB. Table 5.2 below represents the general format for all the database tables.

NAME	SCHEMA	DATA TYPE	SIZE	NULLS?
ID	<none>	NUMBER	0	
DESCRIPTION	<none>	VARCHAR	40	
LOCATION	<none>	VARCHAR	40	✓
IMAGE	ORDSYS	ORDIMAGE		✓

Table 5.2 Database format for image tables

5.2.2 Client Machine

Internet Explorer (IE) was used as the default Web browser throughout application testing and had the responsibility of making a request for a Web page and displaying the returned results.

The only setting altered in the default configuration of IE was that a new version of a Web page be requested on each visit to the application. This was done to prohibit IE from caching a copy of an image or previously requested page on the local machine.

The decision to use IE as opposed to another browser, such as Netscape navigator, was determined after numerous client-side timings had been captured within IE and Netscape. The results determined that the correlation between server-side times and client-side times were more consistent in IE than that in Netscape. This behaviour was observed in ASP, ASP.NET, JSP and Servlet applications that serve single or multiple images. For example, figure 5.4 illustrates the client-side and server-side response time results for an ASP application that serves multiple images to a Netscape and IE browser. The identical ASP application was rendered over 18 successive experiments from both browsers with browser caching disabled (refer to client-side and server-side timing methodologies in chapter 5.3).

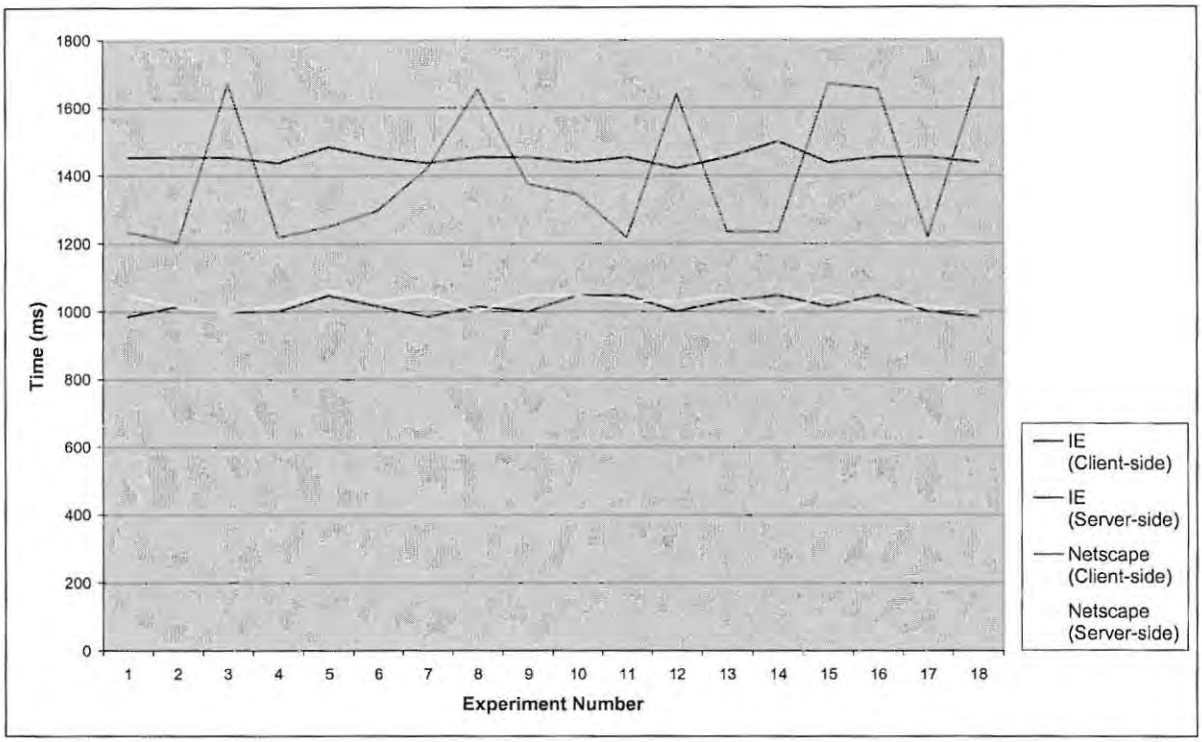

Figure 5.4 Response time results for ASP multiple image serving using IE and Netscape

The server-side response time results are consistent when the ASP application is rendered from IE and Netscape. Client-side response time results fluctuate within Netscape but remain fairly consistent within IE. The inconsistency of client-side response times captured within Netscape was the determining factor for using IE as the default browser within all application testing environments.

5.2.3 Web Server

With reference to Figure 5.1, the middle-tier component of the test environment exposes the most important and complex set-up options for application testing. What follows is a description of the various architectural options made available for application testing and associated justification for their inclusion/exclusion within the test environment.

Option #1: Deploy the Oracle9*i* Application Server and Oracle proxy plug-in for IIS

The Oracle9*i* Application Server (Oracle9*i*AS) is a J2EE-based application server written entirely in Java, providing the latest and most advanced J2EE features for developing and deploying internet applications (Oracle Corporation, 2001a). Some of the features offered within the Oracle9*i*AS are various caching services and Oracle specific containers for executing JSP and Servlet applications. However, in terms of application deployment, a question to be considered with regard to the Oracle9*i*AS, is how J2EE and Microsoft (ASP and ASP.NET) Web applications can interoperate or co-exist within this environment and would this environment provide an ideal architecture for application testing?

Figure 5.5 below depicts this architecture of the Oracle9*i*AS and Microsoft technologies interoperating within the same environment.

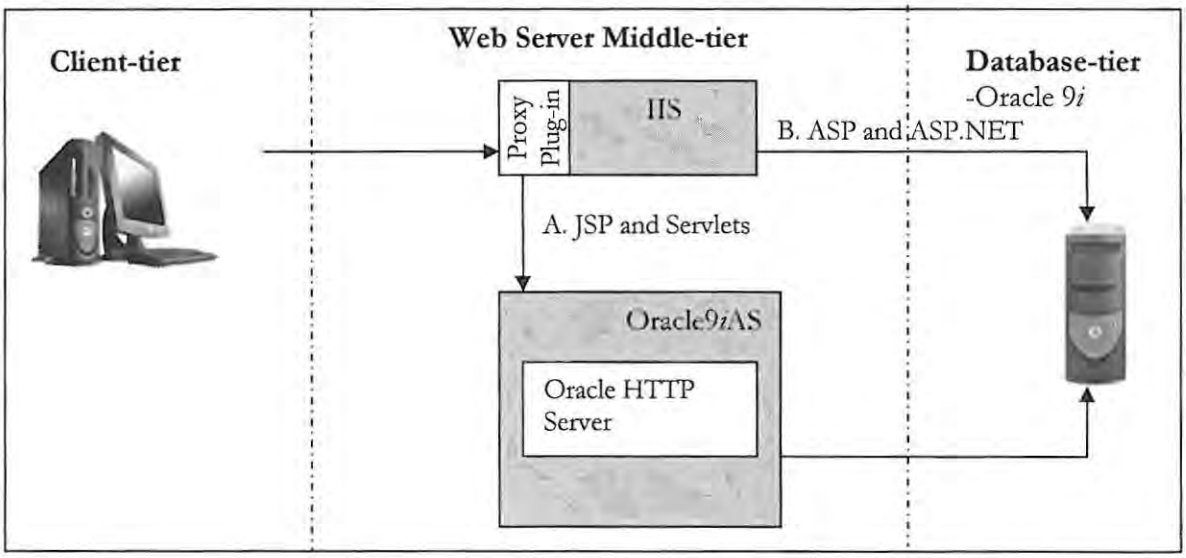

Figure 5.5 Oracle9iAS and the Oracle proxy plug-in for IIS

The Oracle9*i*AS comes with the Oracle HTTP Server (OHS) which is an extended version of the Apache Web server. In addition to the compiled Apache modules provided with OHS, Oracle has enhanced several of the standard modules as well as added Oracle-specific modules for developing and deploying Java-based Web applications (Oracle Corporation, 2002c). However, since ASP and ASP.NET applications cannot be rendered through the OHS, Oracle developed a proxy plug-in for IIS. This proxy plug-in enables reverse proxying of requests back to the OHS and into the Oracle9*i*AS for JSP and Servlet applications (label A in figure 5.5). ASP and ASP.NET applications are handled natively by IIS (label B in figure 5.5).

This architecture was not seen as ideal, for application testing in the research, for the following reasons.

- Firstly, this research concentrates on base technologies, meaning that IIS and Apache should run 'as is' without any modifications to the Web server itself.

- Secondly, the OHS is a single component within a broader, multi-functional Oracle9*i*AS that is specifically geared towards increasing the performance of Java applications within an Oracle environment. The research is not concerned about the performance of the Oracle9*i*AS but with using native ASP, ASP.NET, JSP, and Servlets to access and serve images from the Oracle9*i* database.

- Lastly, Oracle acknowledges a small performance decrease for the additional reverse proxy step when using the Oracle proxy plug-in for IIS.

Option #2: Expose both IIS and Apache Tomcat to the client browsers

In this architecture the Web servers are set up as two independent systems – one powered by IIS for ASP and ASP.NET applications, the other powered by Apache Tomcat for JSP and Servlets, with no integration between the two. Figure 5.6 below depicts this detailed middle-tier architecture.

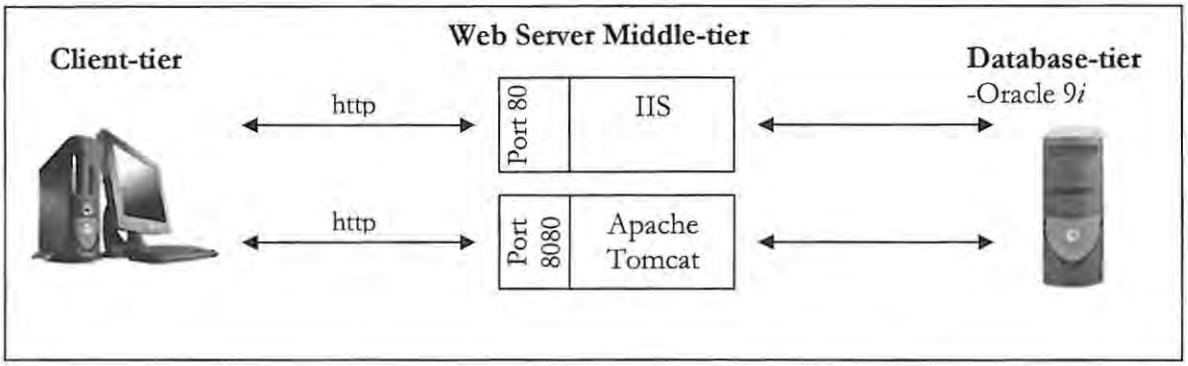

Figure 5.6 Middle-tier test environment

IIS and Apache Tomcat both serve HTTP requests by listening for browser requests at different ports. IIS is set-up to listen for HTTP requests at the default port (port 80) and Apache Tomcat is configured to listen for HTTP requests at another port (for example port 8080). As a result, the port number needs to be specified when making a HTTP request to the Apache Tomcat server and not IIS. It is possible to configure a virtual directory in IIS to eliminate the need to specify port numbers, all requests to this virtual directory get redirected to the correct port on which the Apache server is running. However, this extra step required to redirect a request to Apache Tomcat would decrease the performance of the application and should therefore not be used in the testing environment.

As a result, the architecture referenced in Figure 5.6 was selected for all application tests, providing a clear separation of the various technologies in the middle tier of the three-tier client-server architecture. The client and database tiers remained as consistent components for all application tests and both Web servers were deployed using their default installations.

5.3 Timing Method

The measurement of response time is captured on both the Web server tier and the client tier. Response times captured on the Web server tier calculate the time taken to retrieve images from the database (label A in figure 5.1). Response times captured on the client tier calculate the full image serving process, from initial client request, until all images are returned and displayed within the client's Web browser (label B in figure 5.1).

What remains consistent within all applications is the process of image serving and relevant areas where response times are captured during program execution. Figure 5.7 illustrates the process of image serving with the specific areas where response time is captured, common to all applications.

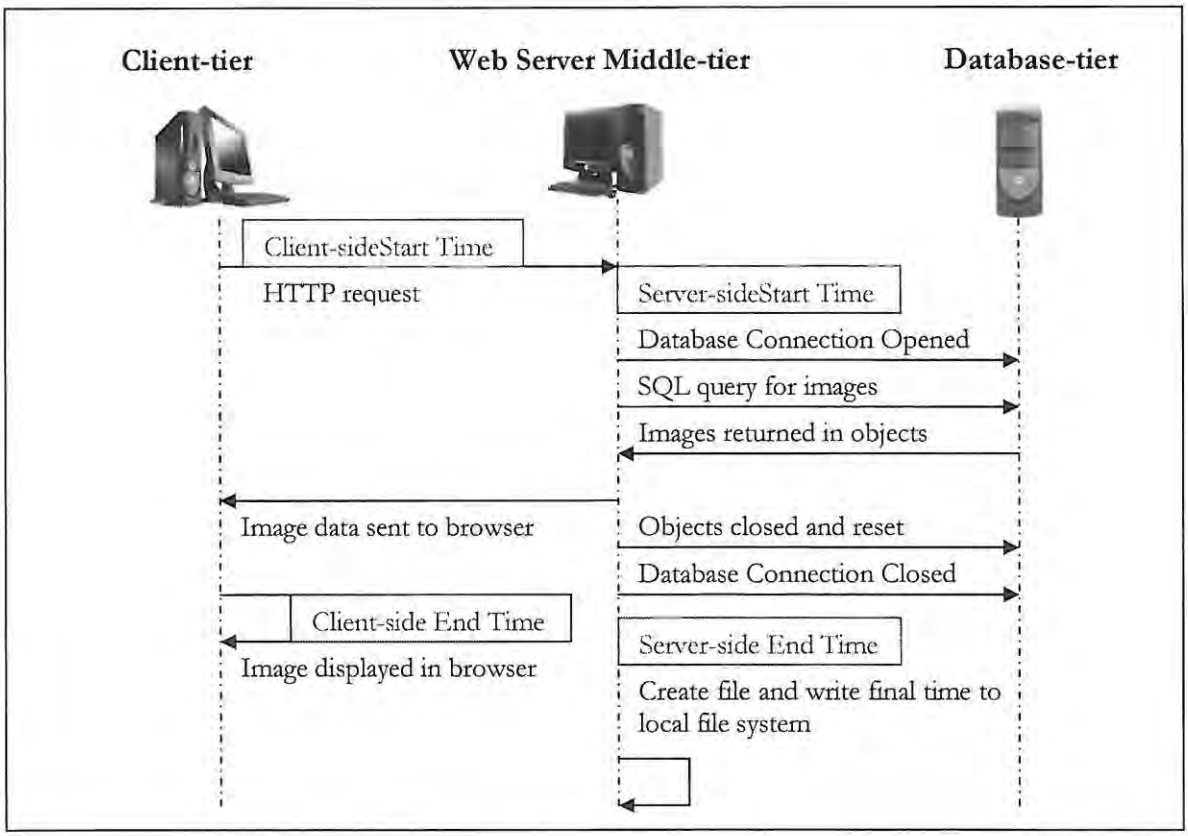

Figure 5.7 Server-side response time methodologies within the image serving process

Client-side Timing Method:

A start time is taken immediately before an HTTP request is submitted by the client. The end time is taken once the image has been successfully loaded within the browser.

Server-side Timing Method:

A start time is taken immediately before a connection is established with the database. Thereafter, a SQL command is used to query the database and once the images have returned to the server they are outputted to the client in sequential order. Finally, the end time is taken once all objects are released and the database connection is closed.

The response time is calculated by subtracting the start time from the end time and appended to a file on the servers file directory.

5.3.1 Client-side Timing

The programmatic task of capturing the response time within IE uses JavaScript to retrieve the client machine's system time as various events occur within the HTML page. Unlike server-side development technologies, JavaScript is a client-side development technology, which means that JavaScript code can be embedded within HTML where it is interpreted and executed within the Web browser, on the client's machine. The process of calculating the response time on the client machine, using JavaScript, is accomplished as follows.

As the HTML page is loaded, the JavaScript pre-built `Date.getTime()` method is used to return a current Date instance, in milliseconds. The code segment below illustrates how the current time is captured in JavaScript.

```
<SCRIPT LANGUAGE="JavaScript"><!--
  var startDate = new Date();
  var startTime = startDate.getTime();
//--></SCRIPT>
```

Thereafter, the HTML image tag is used to retrieve an image from a server-side page on the Web server. An ID number is appended to the querystring to uniquely identify the image. The retrieval of the image occurs as follows.

```
<IMG src="getPhoto.asp?id=1" onload="imageLoaded()">
```

The onload() event of the HTML image tag is used to call a JavaScript function that captures the system time once the image has been successfully loaded within the browser. The onload() event of the image tag is used because the actual retrieval of the image is processed as a separate thread within the browser and is therefore seen as

a separate request. For example, if the system time is taken before the image is retrieved, and again after, without using the onload() event procedure, the time difference is extremely minimal (a few milliseconds) as it only calculates the time taken to fire the separate thread for the image retrieval process. The final time is calculated as follows.

```
<SCRIPT LANGUAGE="JavaScript">
function imageLoaded()
{
  var endDate = new Date();
  var endTime = endDate.getTime();
  var finalTime = endTime-startTime;
  alert(finalTime);
}
</SCRIPT>
```

The response time is calculated by subtracting the start time from the end time and is presented, via a pop-up box, by the client's Web browser.

5.3.2 Server-side Timing

The task of capturing response times on the middle tier is dependant on the server-side development technology as timing methodologies for ASP, ASP.NET, JSP, and Servlet applications are unique and language specific.

The detailed process of capturing response times in ASP, ASP.NET. JSP and Servlets are detailed in the following three sections.

ASP

The Timer() method was used to capture the system time in ASP. This method was introduced with IIS version 5.0 and simply returns the milliseconds elapsed since midnight. Prior to IIS version 5.0, developers were required to use third party components to capture system times to millisecond intervals as ASP could only return values in seconds. The response time is written to the servers file directory using use

File System Object (FSO). The FSO is not a native ASP object but is instantiated from the Microsoft Scripting Runtime DLL (scrrun.dll) providing file manipulation capabilities that can be performed on the local file directory (Mitchell and Atkinson, 2000:507).

ASP.NET

.NET provides developers with various methods to capture system times. Two approaches were used in the research. One approach simply uses the `DateTime.Now.Ticks()` method which returns the number of 100-nanosecond intervals that have elapsed since 12am, January 1st, 0001. However, another approach was developed to incorporate a timing class into the ASP.NET application. This timing class provides the ability to separate the functionality of the timer from the Web application code. In addition, interval times can be captured at any position within the Web application and a final list of all recorded times can be output to a file for further examination.

JSP and Servlets

The timing methodology within JSP and Servlets are functionally identical. The java.util.date package is used, specifically the `System.currentTimeMillis()` method which returns the local machines system time in milliseconds.

The method of writing the final response time to the local file directory is to use the java.io package. For character output to a file on the local system, the PrintWriter class is utilized which is required to be wrapped in a FileWriter class. The FileWriter class has a constructor that takes the name of the file as its argument. If the PrintWriter is created from a FileWriter, the file will be accessed every time a call to the print() or println() methods is made. It is better programming to use buffering, which can be done by creating a BufferedWriter from the FileWriter, and using the BufferedWriter as the argument to the PrintWriter's constructor. For example, in

order to create a PrintWriter to write to the file result.txt the most efficient code would be.

```
FileWriter fw = new FileWriter ("result.txt");
BufferedWriter br = new BufferedWriter (fw);
PrintWriter pw = new PrintWriter (br);
```

5.4 Summary

This chapter provided a detailed description of the test environment and methodologies used to conduct the performance comparisons. Various environment variables, server-side set up options, and timing methodologies were discussed to provide the reader with an in-depth understanding of the techniques used to conduct the performance comparisons made between ASP, ASP.NET, JSP, and Servlets.

CHAPTER 6

PERFORMANCE RESULTS

"Excellence is the gradual result of always striving to do better."

Pat Riley

This chapter provides the results of performance tests of the applications described in chapter 4.

The first sets of tests provide a performance comparison of the various Web development technologies when serving a single image from the Oracle9*i* database. Each test set consists of 4 individual tests, each with an image of different size.

The second sets of tests provide a performance comparison of the various technologies when serving multiple images from the database. Again, each test set consists of 4 individual tests, but instead of increasing the size of the image, the number of images to be served from the database is increased. The first test in each set serves a single image, then 50, 100, and finally 250 images. Each image retrieved from the database is approximately the same size.

Sections 6.1 to 6.3 provide client-side timings where the various drivers/providers within each technology are tested. Client-side times are used to evaluate the complete image serving process, from the initial client request until all images have been displayed in the client's Web browser. The drivers/providers which offer the best

performance within each technology are identified and used in the comparison of the various technologies in section 6.4.

Section 6.4 provides both client-side and server-side timing results to gain a deeper understanding of the mechanisms at work within each technology.

It should also be made clear as to what sections of the timing results are areas of data transport over the network as apposed to processing in different portions of the system. The throughput obtained between the client and server machines was established to be 94.118 Mbps. This value was determined using a program called Qcheck. Qcheck is a free utility program designed to measure accurate network and performance connectivity between two machines (Ixia Corporation, 2004). For further information on this program and its operation, please visit the vendors Web site at http://www.qcheck.net. The established throughput means that the time taken to transport data between the client and server machines takes place at 94.118 Mbps. The remaining time is due to processing consumed by the client and server machines to serve single or multiple images.

6.1 ASP Performance Results

As illustrated in figure 3.8, the Oracle ODBC or Oracle OLE DB drivers provide two approaches for image serving from the Oracle9*i* database using the ADO RecordSet object. Figure 6.1 illustrates the response times for a single image serving application.

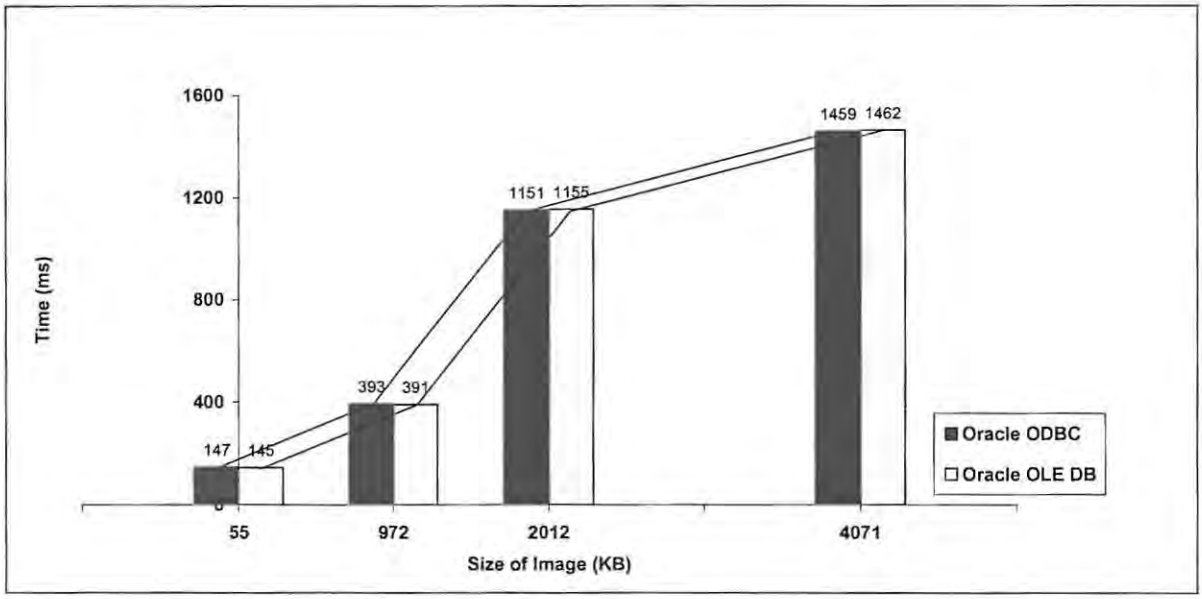

Figure 6.1 ASP single image serving results

As can be seen, the Oracle ODBC and Oracle OLE DB drivers produce almost identical results when serving a single image of varying size. The greatest relative difference in response time, of 2 ms, occurred when serving an image of 55 KB.

The ODBC and OLE DB response times do not increase linearly as the size of the image increases. As illustrated in figure 6.1, it takes approximately 392 ms to serve a 972 KB image. It was expected that a 2012 KB image would take approximately 784 ms to be served. However, the test shows that it takes approximately 1153 ms to serve a 2012 KB image. The opposite is observed when comparing the time taken to serve a 2012 KB image and a 4071 KB image, it takes proportionately less time to serve a 4071 KB image than a 2012 KB image. These findings indicate that in general the application performs more efficiently as the size of the image increases if not in a fully regular way.

When serving a small image (approximately 55 KB) the response time taken to serve the image is relatively large compared to the other three served images. This may well

suggest that the amount of processing on the server machine consumes more time than the actual transmission of image data.

The Oracle OLE DB driver, when serving single images of various size, was selected for the combined comparative results of section 6.4.

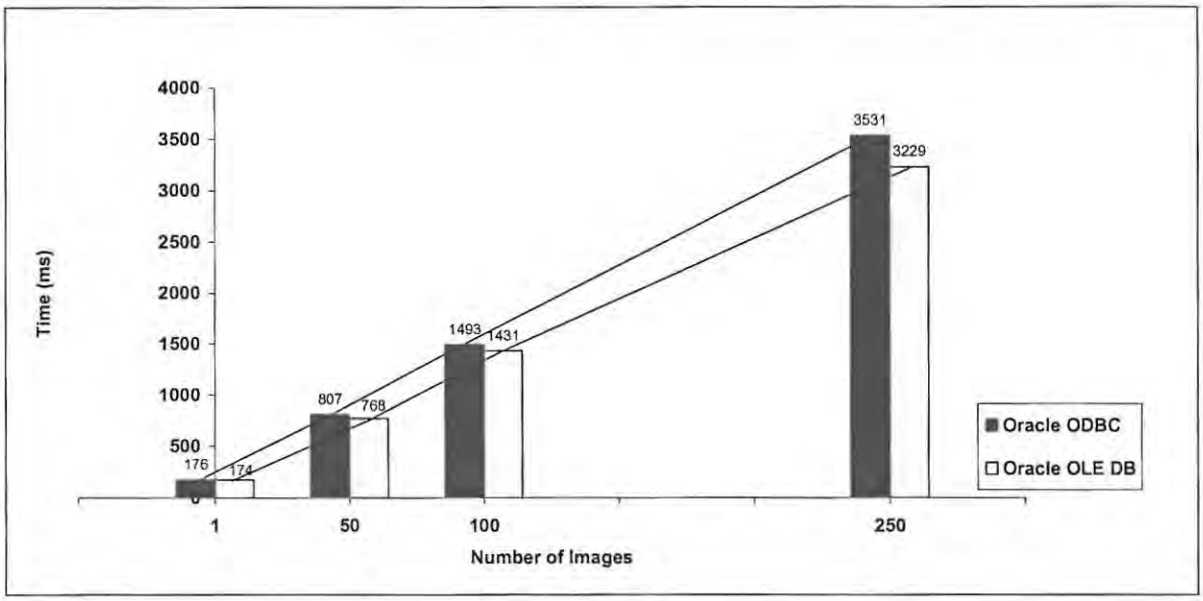

Figure 6.2 ASP multiple image serving results

Figure 6.2 illustrates the results obtained for multiple image serving from an Oracle9i database using the Oracle ODBC and Oracle OLE DB drivers. The results show that the Oracle OLE DB driver is consistently faster than the Oracle ODBC driver. The results also indicate that as the number of served images increases, the Oracle OLE DB driver performs better than the Oracle ODBC driver. Both drivers show a slightly less than linear progression, indicating that the drivers perform slightly better as the number of images to be served increases.

Because the Oracle OLE DB driver outperforms the Oracle ODBC driver when serving multiple images, of approximate same size, it was selected for the combined comparative results to be discussed in section 6.4.

6.2 ASP.NET Performance Results

As illustrated in figure 3.8, ASP.NET applications can use the ADO.NET DataReader object or the ADO.NET DataSet object to serve images from an Oracle9i database. Both objects use identical providers, namely the Microsoft .NET Framework Data Provider for Oracle, the ODBC .NET Framework Data Provider, and the OLE DB .NET Framework Data Provider. Figures 6.3 and 6.4 illustrate the single image serving results using the ADO.NET DataReader object and DataSet object respectively, in conjunction with the specified providers.

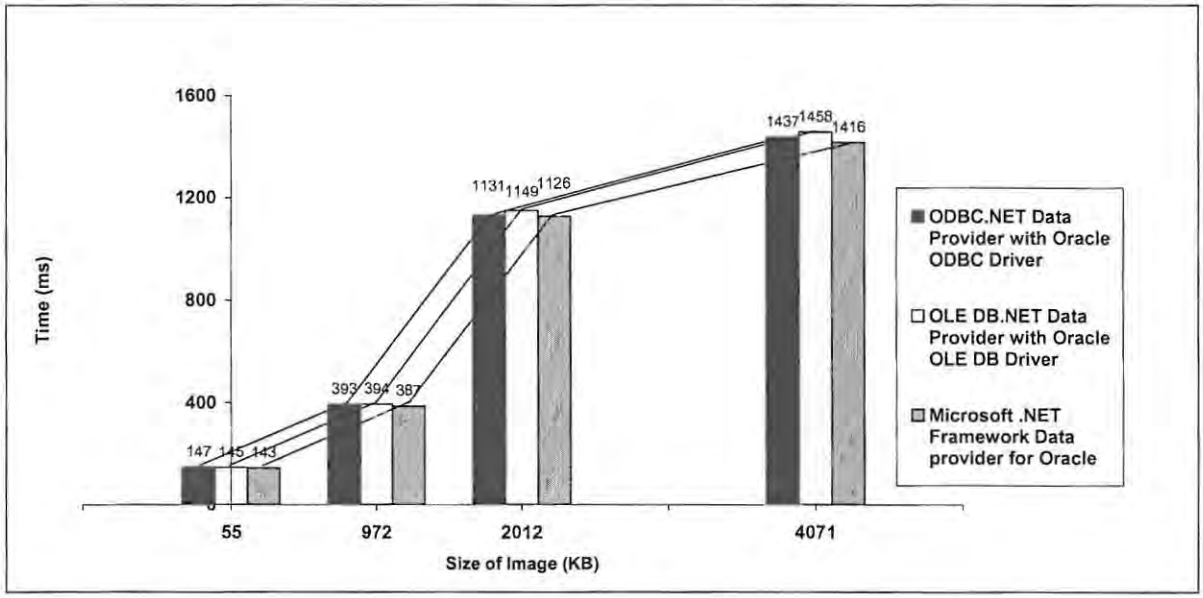

Figure 6.3 ASP.NET single image serving results (ADO.NET DataReader object)

As illustrated in figure 6.3, the OLE DB.NET Data Provider consistently produced the slowest response time across the four experiments. The Microsoft .NET Framework Data Provider was consistently the fastest of the three providers whilst the ODBC .NET Data Provider generated intermediate results. It should be noted that the Microsoft .NET Framework Data Provider did not outperform the other providers by more than 42 ms which was observed when serving an image of 4071 KB.

Again, the response times do not increase linearly as the size of the image increases, showing a very similar pattern to ASP in figure 6.1. As illustrated in figure 6.2, it takes proportionately longer time to serve a 2012 KB image when compared to a 972 KB image. The opposite is observed when comparing the response times taken to serve a 2012 KB image and a 4071 KB image. It proportionately takes less time to serve a 4071 KB image when compared to a 972 KB or 2012 KB image. When serving a small image, approximately 55 KB, the response time taken to serve the image is relatively large compared to the other results. Again, this may well suggest that the amount of processing on the server machine consumes more time than the actual transmission of data.

These findings indicate that in general the application performs more efficiently as the size of the image increases, if not in a fully regular way.

The Microsoft .NET Framework Data Provider for an ASP.NET (DataReader object) multiple image serving application was selected for the combined comparative results to be discussed in section 6.4.

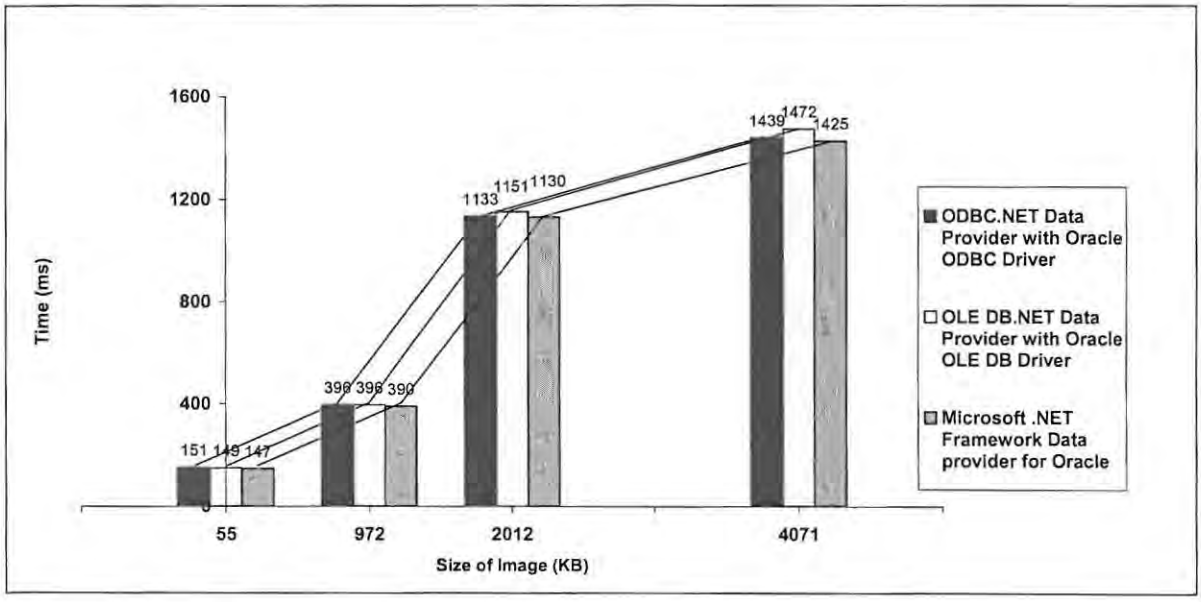

Figure 6.4 ASP.NET single image serving results (ADO.NET DataSet object)

The results in figure 6.4 illustrate that the DataSet object shows characteristics similar to those shown by the DataReader object in figure 6.3. Once again, the Microsoft .NET Framework Data Provider produced the fastest response time for single image serving. The ODBC .NET Framework Data Providers produced similar results to the OLE DB .NET Framework Data Providers, but gradually becomes faster as the size of the image is increased.

There is a high correlation between the results of the ADO.NET DataReader and DataSet objects when serving a single image, of varying size, from an Oracle9*i* database. Figures 6.3 and 6.4 illustrate similar results (that is, not more than a 14 millisecond difference (1%)) between a specified provider and the corresponding DataReader object and DataSet object. What is illustrated, although very marginally, is that the DataReader object outperforms the DataSet object for single image serving from the Oracle9*i* database. The DataSet object was therefore discarded from the combined results to be compared in section 6.4.

Figures 6.5 and 6.6 illustrate the multiple image serving results using the ADO.NET DataReader object and DataSet object in conjunction with the specified providers.

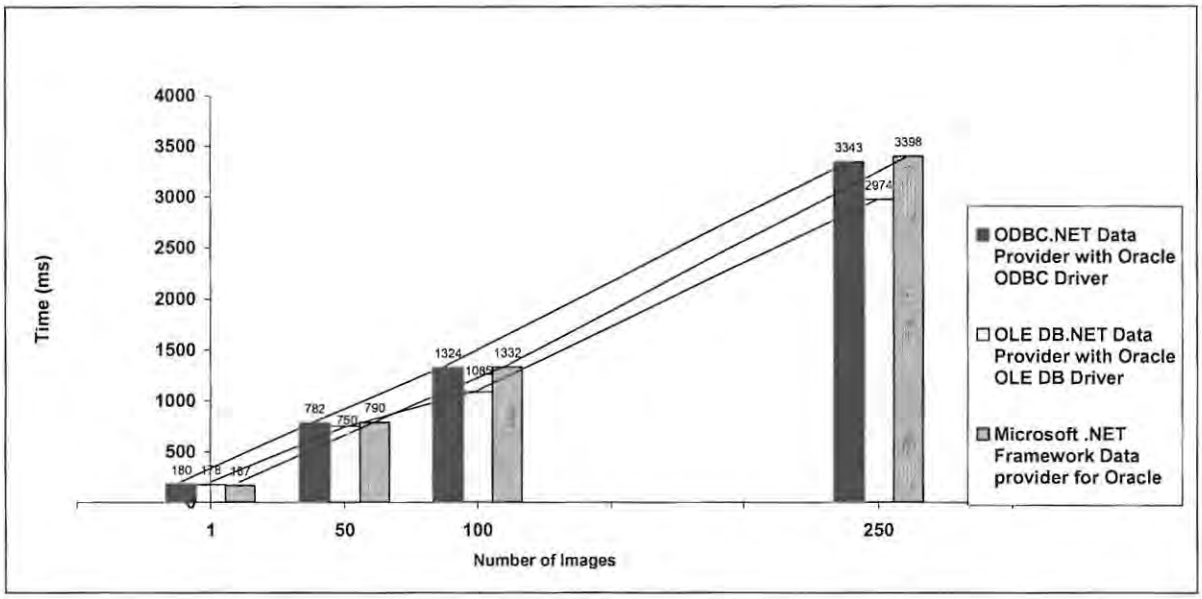

Figure 6.5 ASP.NET multiple image serving results (ADO.NET DataReader object)

Figure 6.5 illustrates that the Microsoft .NET Framework Data Provider generates the fastest response time when a single image is served, corresponding to the results obtained in figure 6.3. However, as the number of images increase, the Microsoft .NET Framework Data Provider becomes the slowest of the three providers. The OLE DB .NET Data Provider consistently generates the fastest response time as the number of images increase whilst the ODBC .NET Data Provider produces intermediate results.

From the results it is observed that all three providers illustrate a non-linear progression. After approximately 100 images have been served, a slight increase in the linear progression is observed. This suggests that the performance of ASP.NET slightly decreases as the number of images served increases.

Because the OLE DB .NET Data Provider outperforms the Microsoft .NET Framework Data Provider and ODBC .NET Data Provider when serving multiple images, of approximate same size, it was selected for the combined comparative results to be discussed in section 6.4.

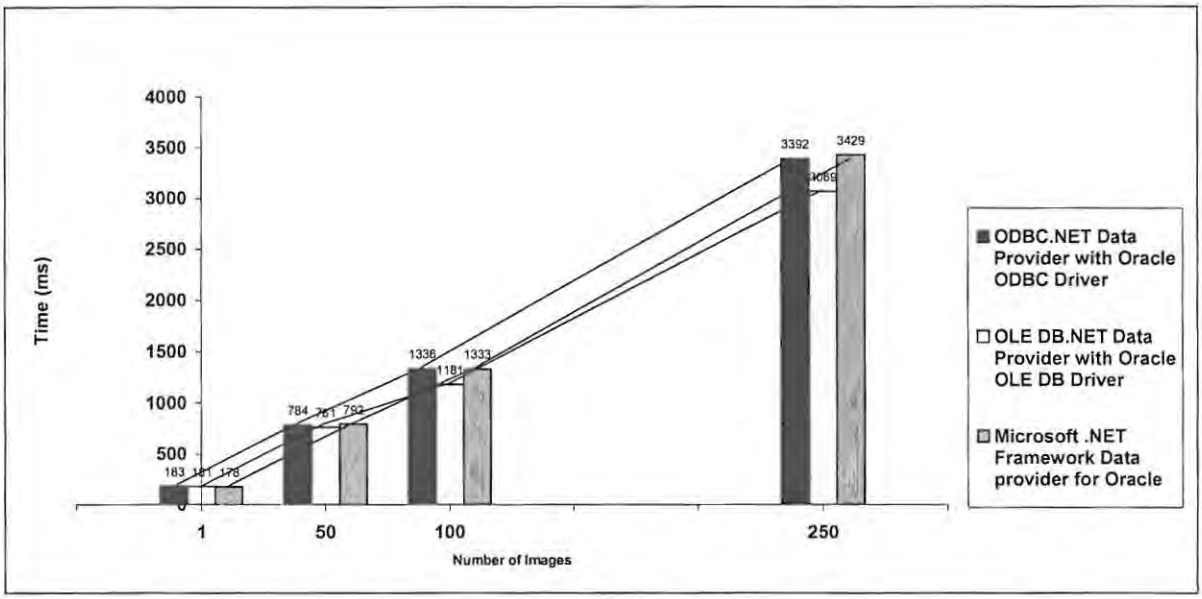

Figure 6.6 ASP.NET multiple image serving results (ADO.NET DataSet object)

Figure 6.6 again illustrates that the Microsoft .NET Framework Data Provider produces the fastest response when serving a single image from the Oracle9i database, but gradually becomes the slowest of the three providers as the number of served images increases. The OLE DB .NET Data Provider consistently outperformed the other providers as the size of the image increased whilst the ODBC .NET provider generated response times very similar to the Microsoft .NET Framework Data Provider.

Again, the results illustrated in figure 6.6, using the DataSet object, show a pattern similar to the DataReader object (refer to figure 6.5) for serving multiple images. The OLE DB .NET Data Provider produces the fastest response time when used with the DataReader and DataSet objects. The DataSet object consistently produced a slower response time, although not more than 95 milliseconds (3%), when compared to the DataReader object, for multiple image serving from an Oracle9i database. The linear progression of the DataReader object and DataSet object are also similar to the

one discussed in the case of the DataReader object above. The DataSet object was therefore discarded from the combined results to be compared in section 6.4.

6.3 JSP and Servlet Performance Results

As illustrated in figures 6.7 and 6.8, JSP and Servlet applications can use the Oracle Thin or Oracle OCI JDBC drivers to serve a single image, of varying size, from an Oracle9i database.

Figure 6.7 illustrates the single image serving results for a Servlet application using the Oracle Thin JDBC driver and Oracle OCI JDBC driver.

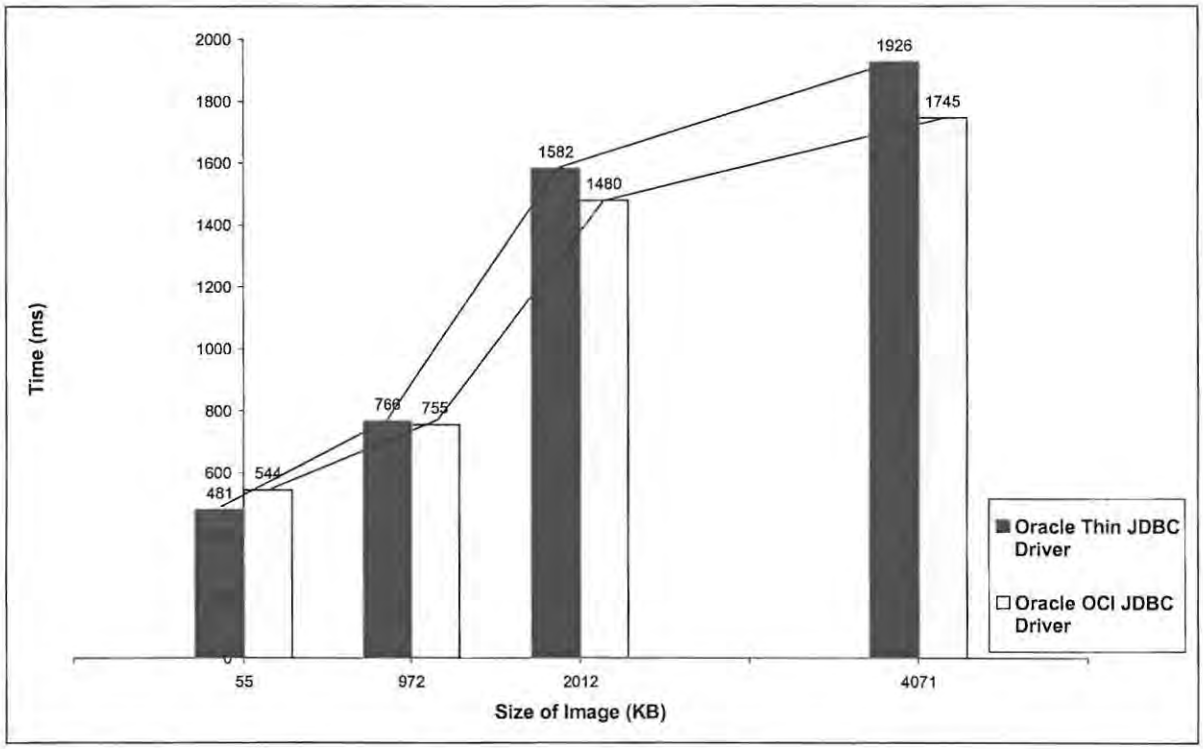

Figure 6.7 Servlet single image serving results (Oracle ResultSet object)

As one can see, the Oracle Thin JDBC driver outperforms the Oracle OCI JDBC driver when the image to be served is less than 1000 KB in size. Thereafter, the Oracle OCI JDBC driver produces a faster response time.

The results depict an approximate linear progression from serving a 972 KB image to serving a 4071 KB image. As the image is increased beyond 2012 KB, the response time is less than linear, illustrating that Servlets perform better as the size of the image is increased. When serving a small image, approximately 55 KB, the response time taken to serve the image is relatively large. This may well suggest that the amount of processing on the server machine consumes more time than the actual transmission of image data.

Figure 6.8, illustrates the single image serving results for a JSP application using the Oracle Thin JDBC driver and Oracle OCI JDBC driver.

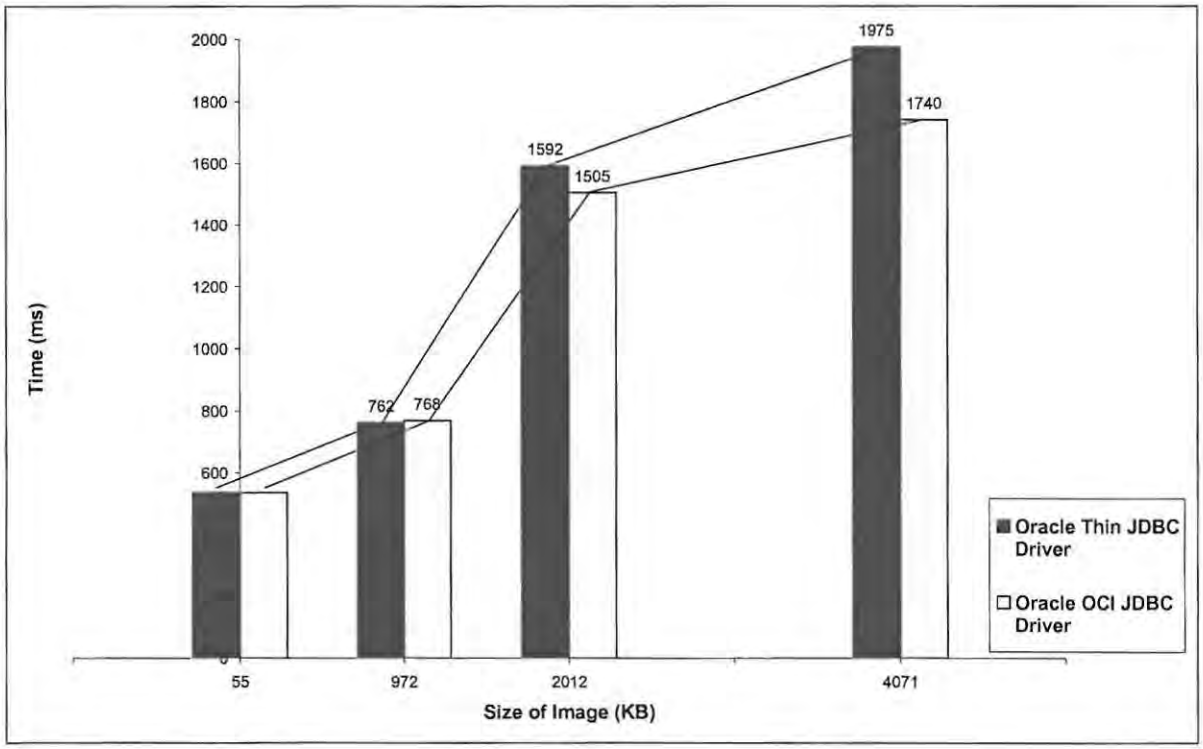

Figure 6.8 JSP single image serving results (Oracle ResultSet object)

As can be seen, the Oracle OCI JDBC Driver outperforms the Oracle Thin JDBC driver when the image served is greater than 1000 KB. There seemed to be no significant difference when serving an image less than 1000 KB.

Besides the Servlet application outperforming the JSP application when using the Oracle OCI JDBC driver to serve an image less than 1000 KB, there is no significant difference between the performance of Servlet and JSP applications when serving a single image, of varying size, from the Oracle 9i database. It was therefore decided that the JSP results be discarded from section 6.4 for the single image serving comparisons.

Figure 6.9 illustrates the multiple image serving results for a Servlet application using the Oracle Thin JDBC driver and Oracle OCI JDBC driver.

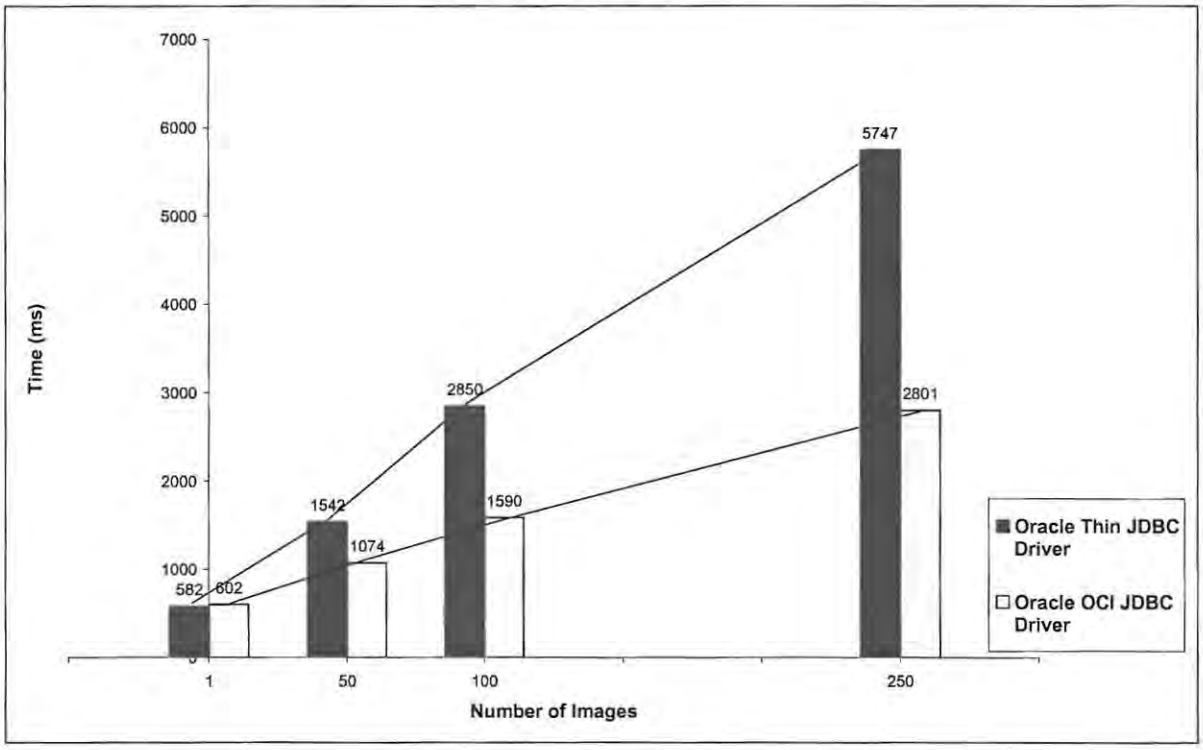

Figure 6.9 Servlet multiple image serving results (Oracle ResultSet object)

It can be seen that the Oracle OCI driver outperforms the Oracle Thin JDBC driver as the number of images to be served is increased.

Both drivers illustrate a less than linear progression as the number of served images increases. This is most apparent when serving 50 images compared to 100 images using the Oracle OCI driver.

Figure 6.10 illustrates the multiple image serving results for a JSP application using the Oracle Thin JDBC driver and Oracle OCI JDBC driver.

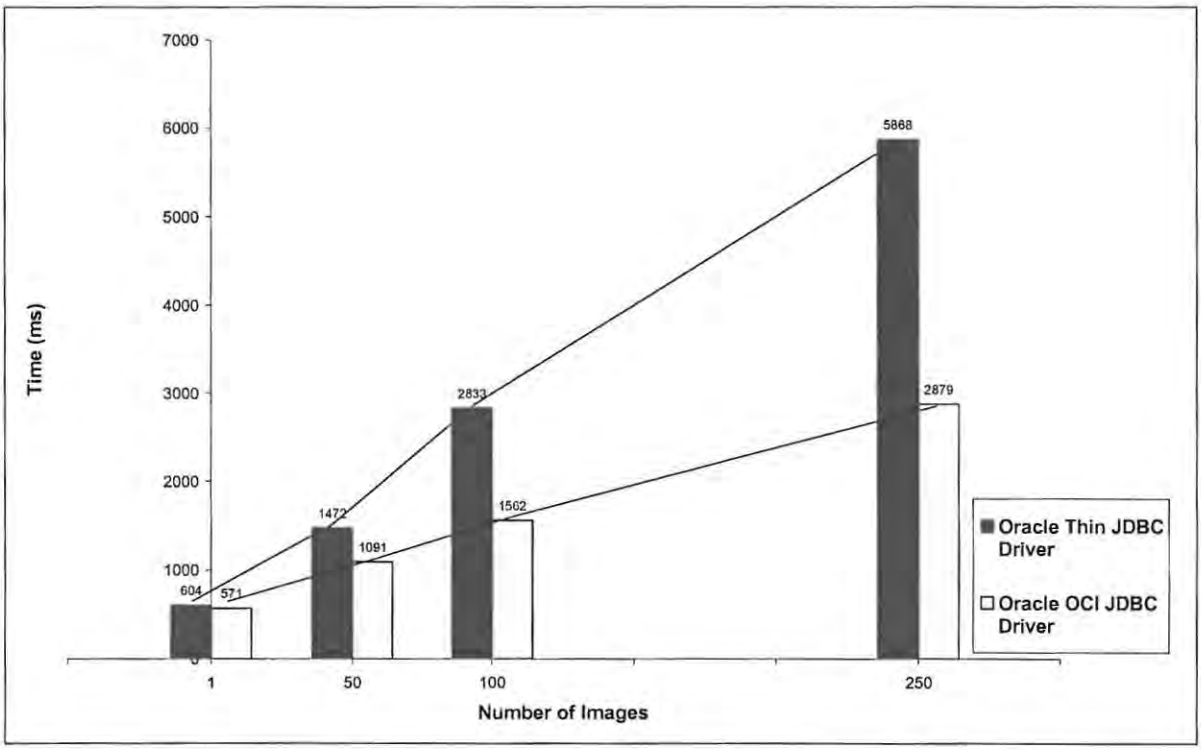

Figure 6.10 JSP multiple image serving results (Oracle ResultSet object)

JSP show response times similar to those of Servlets (figure 6.9). The Oracle OCI JDBC driver once again significantly outperforms the Oracle Thin JDBC driver when the number of images to be served is increased. There seemed to be no significant difference between Servlet and JSP applications when serving multiple images from an Oracle9*i* database and both technologies illustrated similar linear progressions. It was therefore decided that the JSP results be discarded from section 6.4 for the multiple image serving comparisons.

6.4 Combined Performance Results

This section combines and evaluates the results obtained using ASP, ASP.NET, and Servlet technologies to serve single and multiple images from an Oracle9*i* database. Table 6.1 illustrates the various technologies and associated drivers/providers that were selected for the inter-technology comparison, based on best performance results from sections 6.1 to 6.3.

JSP results were discarded from Table 6.1 because they were very similar to those for Servlets.

Technology	Performance Test	Database Driver/Provider
ASP	**Single image serving**	**Oracle OLE DB Driver**
ASP	**Multiple image serving**	**Oracle OLE DB Driver**
ASP.NET	**Single image serving**	**Microsoft .NET Framework Data Provider**
ASP.NET	**Multiple image serving**	**OLE DB.NET Framework Data Provider**
Servlet	**Single image serving**	**Oracle OCI JDBC Driver**
Servlet	**Multiple image serving**	**Oracle OCI JDBC Driver**

Table 6.1 Selected technologies and associated drivers/providers for the combined performance comparison results.

Figures 6.11 and 6.12 illustrate the client-side and server-side timing results when using ASP, ASP.NET, and Servlet technologies to serve a single image, of varying size. Server-side response time results are introduced in this section to gain a deeper understanding of the mechanisms at work within each technology. The aim is to present the reader with the differences that occur within each Web technology during the processing of Web applications.

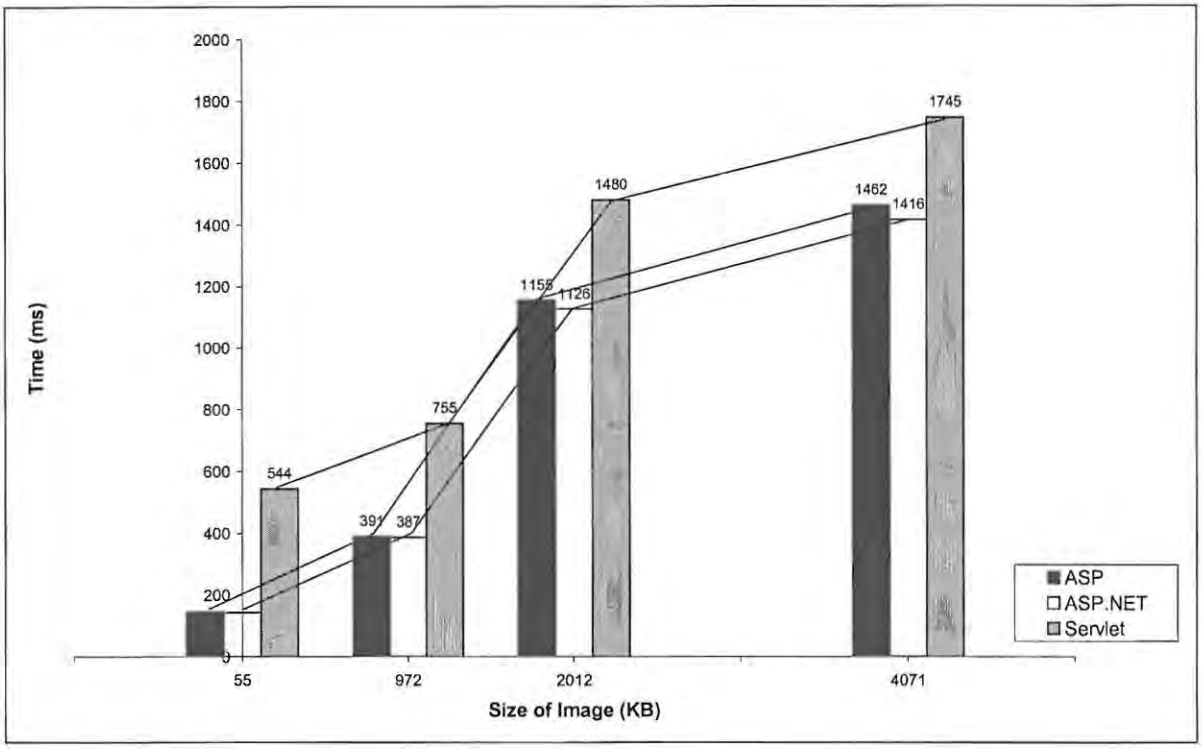

Figure 6.11 Combined single image serving results – Client-side response times

As can be seen, ASP.NET produces the fastest client-side response time when serving a single image which is greater than approximately 1000KB in size. Both ASP and ASP.NET applications are equally efficient when serving an image which is less than approximately 1000KB in size.

Servlets perform poorly against both ASP and ASP.NET when serving a single image of varying size. On average, Servlets take 350 milliseconds longer to serve an image than ASP and ASP.NET.

Although there is a significant difference in the response time results between Servlets and ASP or ASP.NET, all technologies show a similar linear progression. All technologies perform more efficiently as the size of the image is increased beyond 2012 KB. However, it takes proportionately longer to serve a 2012 KB image than a 972 KB image.

All technologies show a large response time when serving a small image (approximately 55 KB), indicating that the server processing on the machine consumes more time than the transmission of data on the network.

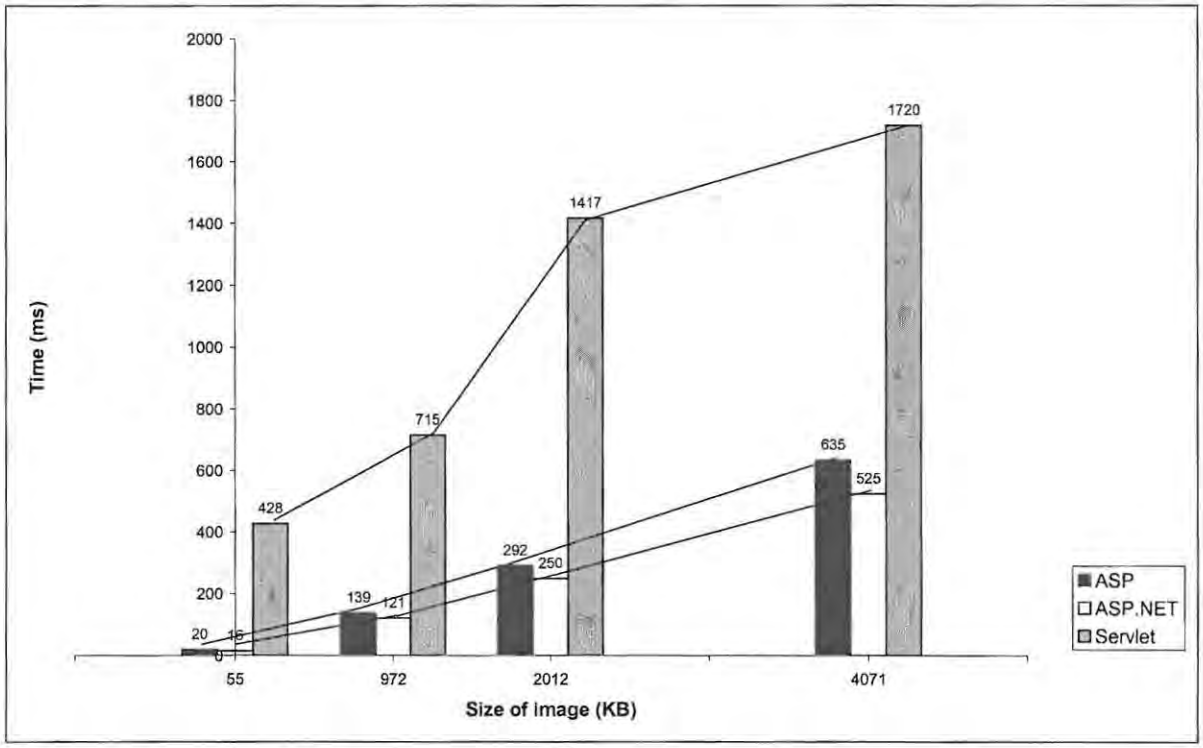

Figure 6.12 Combined single image serving results – Server-side response times

Figure 6.12 illustrates server-side response times when using Servlets, ASP and ASP.NET. The ASP.NET application outperforms the ASP application as the size of the image to be served is increased. It must be noted that there is a substantial time difference between client-side and server-side response times in both ASP and ASP.NET applications. This difference in response time ranges from approximately 125 milliseconds when serving a 55 KB image, to almost a second when serving a 4071 KB image.

The Servlet application, however, does not show a substantial time difference between the client-side and server-side response times. This difference ranges between 127 and 25 milliseconds, depending on the size of the image to be served. A

discussion on the significant difference existing between the server-side response time results in Servlet and ASP or ASP.NET application can be found in chapter 7. This discussion identifies why client-side response times are more meaningful for performance testing than server-side response times.

Figures 6.13 and 6.14 illustrate the client-side and server-side response time results when using ASP, ASP.NET, and Servlet technologies to serve multiple images from the Oracle9i database to the client's Web browser.

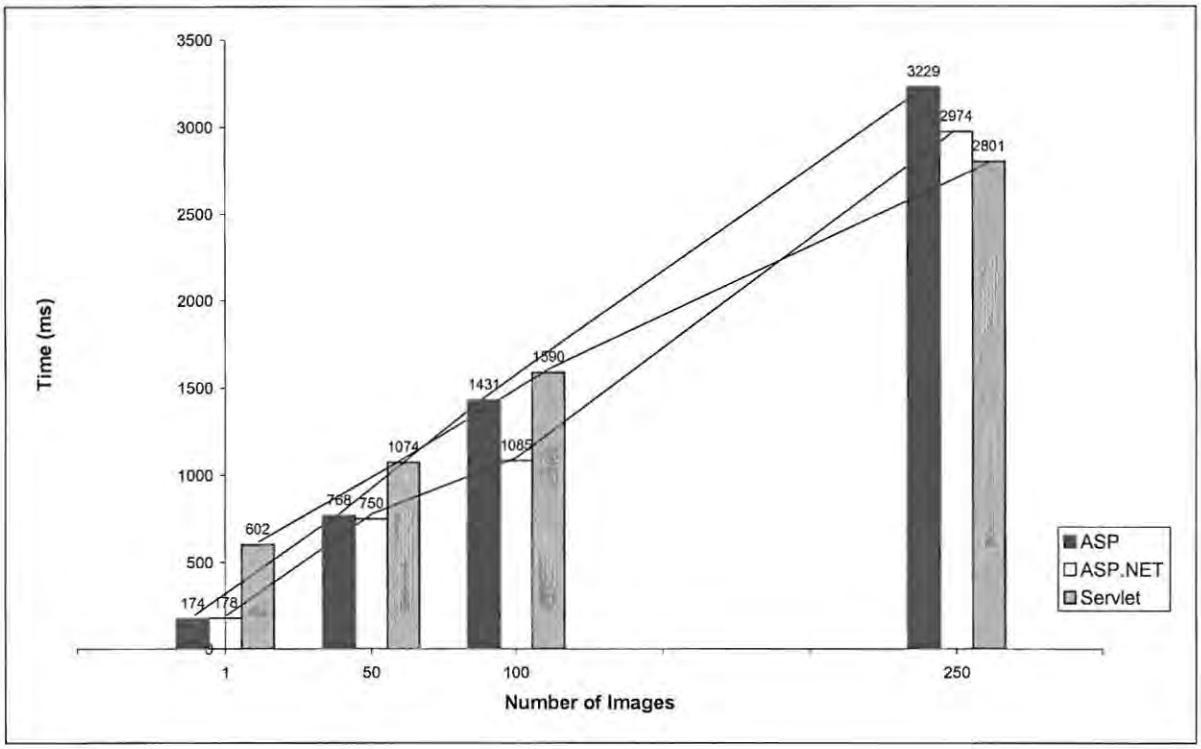

Figure 6.13 Combined multiple image serving results – Client-side response times

As can be seen, both ASP and ASP.NET applications generate almost identical client-side response times until approximately 50 images have been served. Thereafter, the ASP.NET application generates a faster response time as the number of images served increases.

The Servlet application generates a slower response time than the ASP application until approximately 70 images are being served. Thereafter the Servlet application

outperforms the ASP application. The Servlet application again generates a slower response time than the ASP.NET application until approximately 190 images are being served. Thereafter the Servlet application outperforms the ASP.NET application.

Both ASP and ASP.NET illustrate a slightly more than linear progression, indicating that the performance of the applications decreases as the number of images to be served in a single request set increases. Servlets, however, show a less than linear progression, indicating that Servlets perform more efficiently as the number of images to be served increases.

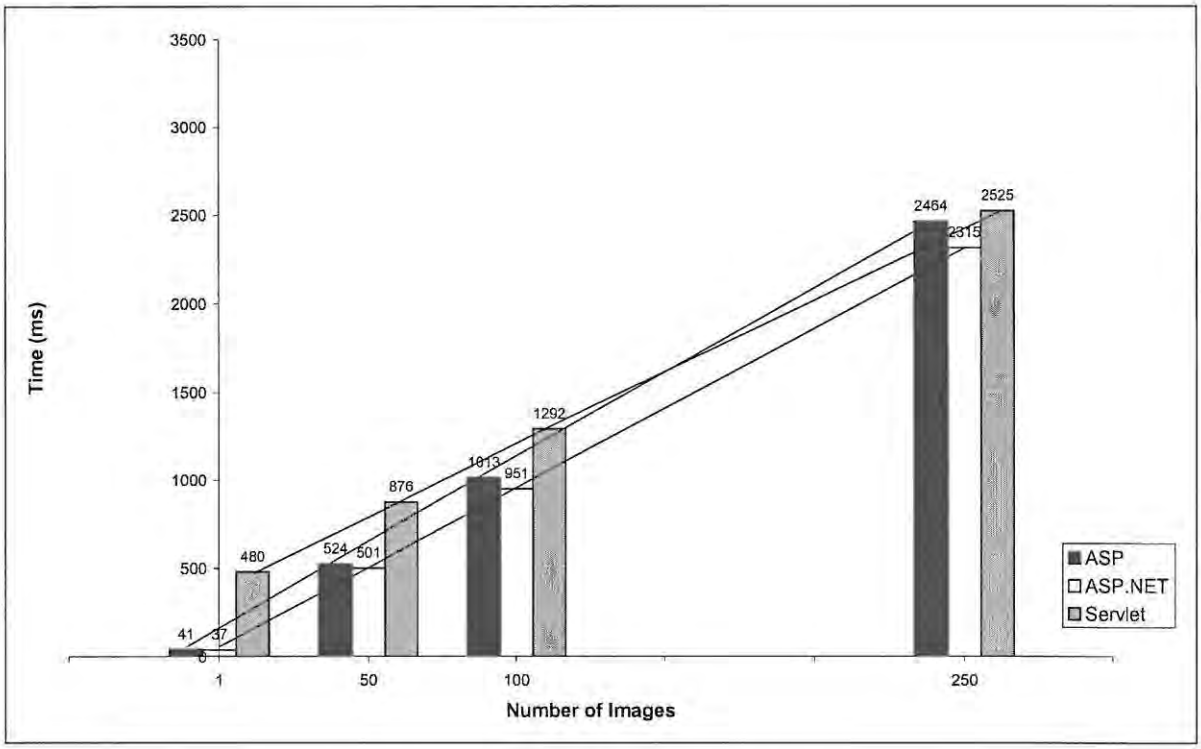

Figure 6.14 Combined multiple image serving results – Server-side response times

Figure 6.14 illustrates that the ASP.NET application outperforms the ASP application as the number of images served increases.

The Servlet application is outperformed by both ASP and ASP.NET applications across the four experimental scenarios.

6.5 Summary

This chapter detailed the response times obtained when using ASP, ASP.NET, JSP, and Servlets for image serving from an Oracle9*i* database. The performance results are based on single and multiple image serving, as described in chapter 5.

The results were firstly grouped according to the technology used, focusing within each technology on performance differences due to the various providers/drivers identified in chapter 3. Then, a combination of the results was presented, comparing the four technologies against each other for single and multiple image serving.

The results indicated that ASP and ASP.NET perform almost identically when serving a single image that is less than 1MB from an Oracle9*i* database. Thereafter, ASP.NET slightly outperforms ASP as the size of the served image increases. JSP and Servlets are consistently slower than ASP and ASP.NET when serving a single image, of varying size, from Oracle9*i*.

ASP and ASP.NET perform equally well when serving approximately 50 images, in a single request set, from an Oracle9*i* database. Thereafter, ASP.NET outperforms ASP as the number of served images increases. JSP and Servlets perform slower than ASP and ASP.NET when serving less than approximately 70 images when compared to ASP, and 190 images when compared to ASP.NET. Thereafter, JSP and Servlets outperform ASP and ASP.NET as the number of served images increases.

JSP and Servlet technologies perform almost identically when serving single or multiple images from Oracle9*i*. This finding did not include the first request for a JSP file where it is compiled and stored as a Servlet class. The DataReader object slightly outperforms the DataSet object in ASP.NET.

CHAPTER 7

DISCUSSION AND RECOMMENDATIONS

"Knowledge is more than equivalent to force."

Rasselas

This chapter discusses the response times presented in chapter 6 and details the author's recommendations for the development of data-driven Web applications that serve images from an Oracle9*i* database.

The chapter is divided into two sections. The first section provides recommendations based on a discussion of the results presented in chapter 6. It focuses on client-side response times, which considers the full image serving process. Chapter 6 identified a major discrepancy between the server-side response times in Microsoft and Java technologies. This discrepancy is intended to indicate why client-side response times are more meaningful for performance testing than server-side response times. Both client-side and server-side response times are analysed and discussed in each technology.

The second section provides further recommendations based on a view broader than just performance. Web technologies are contrasted in terms of ease of programming, error handling, tracing, and caching. Three architectures, one that integrates ASP and ASP.NET, the second that integrates JSP and Servlets, and the last integrating ASP.NET and Servlets are also proposed.

Throughout the chapter, the expression *Java technologies* refers to JSP and Servlets, and in a similar manner, *Microsoft technologies* refers to ASP and ASP.NET.

7.1 Performance

The following section recommends the selection of a particular technology, with associated driver/provider, for use in image serving applications and discusses other performance observations acquired from chapter 6.

- **Single Image Serving**

  As illustrated in figure 6.11, Java technologies are consistently slower than Microsoft technologies when serving a single image, of varying size, from Oracle9*i*. Java technologies are approximately a half a second slower than Microsoft technologies when serving a small image (+- 55KB) and approximately one third of a second slower when serving a large image (+- 4MB). This observation indicates that Servlets perform better as the size of the image increases, but are still slower than ASP and ASP.NET.

  Developers need to consider whether half a second difference is sufficient reason to select Microsoft technologies over Java technologies. For serving a single image to a small amount of users, such a difference may not be that significant. But as the number of single images to be served increases, Microsoft technologies should provide better throughput over Java technologies.

  ASP and ASP.NET perform almost identically when serving a single image that is less than +- 1MB. Thereafter, ASP.NET slightly outperforms ASP as the size of the image increases. However, ASP.NET does not outperform ASP by more than 3.3%. Again, this maximum performance increase of 3.3% may not be substantial enough to select ASP.NET over ASP.

Based solely on performance, it is recommended to use ASP or ASP.NET for serving an image that is less than 1MB. Images greater than 1MB are best served with ASP.NET. The Oracle OLE DB driver is to be used with ASP applications and the Microsoft .NET Framework Data Provider is to be used with ASP.NET applications.

Another benefit of using ASP.NET over ASP in terms of performance, is that ASP does not allow the caching of images retrieved from Oracle9*i* and relies on IIS to cache interpreted ASP files and scripting engine instances. ASP.NET provides a better model to caching, enabling the caching of images and other dynamic content from Oracle 9i.

- **Multiple Image Serving**

As illustrated in figure 6.13, ASP and ASP.NET perform equally when serving approximately 50 images from an Oracle9*i* database. Thereafter, ASP.NET outperforms ASP which suggests that ASP.NET is more scalable as the number of images to be served increases.

Java technologies are slower than Microsoft technologies when serving less than approximately 70 images when compared to ASP, or 190 images when compared to ASP.NET. Thereafter, Java technologies outperform Microsoft technologies. This suggests that Java technologies provide better scalability when serving single requests of relatively large sets of images from an Oracle9*i* database.

The following recommendations can be made, based solely on performance, when serving multiple images from an Oracle9*i* database.

If the Web application is designed to serve a relatively small number of images, the developer should use ASP.NET. For applications that need to scale, designed for serving a large number of images, that developer should use Java

technologies. The OLE DB.NET Framework Data Provider is to be used for ASP.NET applications and the Oracle OCI JDBC Driver is to be used with JSP and Servlet applications.

Again, developers need to note that ASP and Java technologies do not provide the ability to cache data from an Oracle9*i* database. ASP.NET allows for the caching of images from Oracle9*i*, which if used appropriately, can greatly improve the performance of ASP.NET applications.

- **Client-side and Server-side Response Time Difference**

As previously mentioned (section 6.4) there is a major discrepancy between the server-side response times generated in Microsoft and Java technologies. When comparing the client-side response time, illustrated in figure 6.11, to the server-side response time, illustrated in figure 6.12, there is a substantially larger difference in response time found within Microsoft technologies when compared to Java technologies.

The response time difference indicates that Java technologies do not continue with program execution until the entire image has been transmitted to the client, while investigations indicate that Microsoft technologies continue with program execution once a separate thread has been initiated to transmit the image. This would make Microsoft technologies more efficient than Java technologies.

Because client-side response times calculate the full image serving process, from initial client request until the image is displayed within the browser, this also indicate that client-side response times are more meaningful (i.e. take into consideration the entire image serving process) for performance testing than server-side response times.

- **JSP and Servlets**

  Section 6.4 illustrates the response times for single and multiple image serving, using JSP and Servlet applications. It is clearly noticeable that there is no significant difference between the performance of JSP and Servlets when serving images from an Oracle9*i* database.

  The response time graphed in section 6.4 does not include the first and second response time of each test series. Consequently, the results do not consider the first request for a JSP file where it is compiled and stored as a Servlet class. This extra translation step does make JSP's slower than Servlets for the first request (Roman, 1999). Thereafter, minimal performance difference between JSP and Servlet applications is observed.

  Consequently, developers need to take into consideration the first request for a JSP file. If the Web application is accessed infrequently or for continuous new requests, unrelated to the previous ones, it would be best to use Servlets as they outperform JSP's on the initial request. If the Web application is accessed frequently, the developer has the option of using either technology.

7.2 Language Features and Development Environment

7.2.1 Microsoft Technologies

Apart from performance considerations, it is recommended to use ASP.NET over ASP when developing image serving applications for the following reasons.

- **Error Handling and Tracing:** ASP provides an unstructured approach to handling errors through the use of the ASPError object and the *On Error Resume Next* construct. ASP.NET provides a better approach to handling errors using the *try-catch-finally* construct and introduces the capability of tracing diagnostic information during program execution.

- **Ease of Programming:** ASP, which uses object aware scripting languages, provides a relatively simple means to serve images using ADO objects and vendor supplied drivers. ASP, however, does not provide a clear separation between programming logic and user interface mechanisms and much of an application's advanced functionality is often included as third party components and COM objects.

  ASP.NET has been redesigned from scratch and is now fully object-oriented. Programmers can choose between a variety of programming languages and the ability to instantiate classes and separate business logic from the user interface makes ASP.NET applications simpler to write and more structured.

It is possible to use a combination of ASP and ASP.NET to serve images from Oracle9*i*. This is useful, for example, in the case of a developer familiar with the programming techniques and features of ASP, but wanting the scalability and performance offered by ASP.NET when serving images from an Oracle9*i* database. The developer can use ASP.NET purely for image serving and ASP for all other areas of application processing. Figure 7.1 illustrates this architecture.

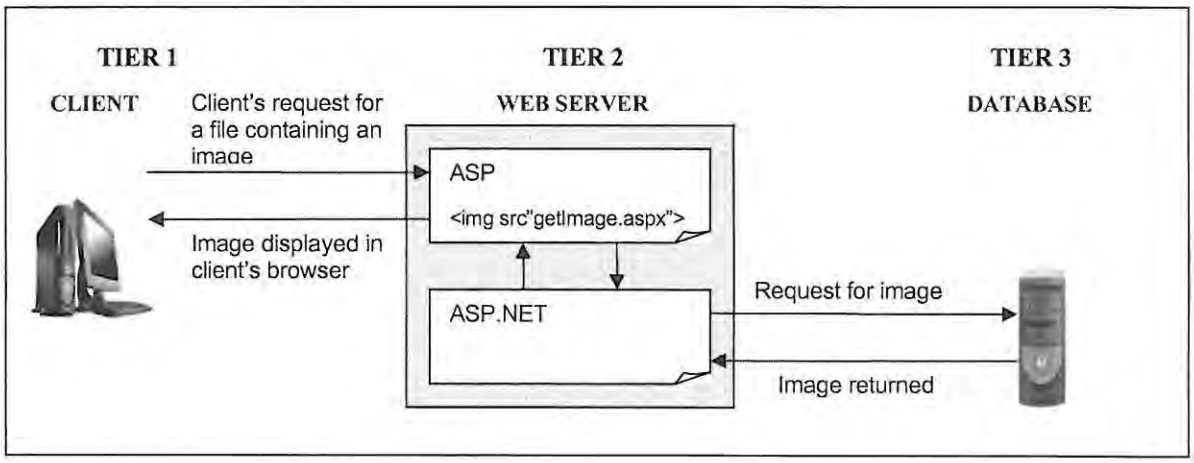

Figure 7.1 Using a combination of ASP and ASP.NET

As depicted in figure 7.1, the source parameter of an image, specified within an ASP file, points to an ASP.NET file. The ASP.NET file is responsible for the retrieval of the image from the Oracle9*i* database and its transmission back to the ASP file, where it is formatted with other data. IIS provides the processing required by both ASP and ASP.NET for this situation.

7.2.2 Java Technologies

The advantages and disadvantages experienced by the author for JSP and Servlet applications are as follows.

- The major advantage of Servlets over JSP's is that a developer has more control over the application environment. For example, in section 4.5 it was noted that Apache Tomcat throws an *illegal state* exception if a developer opens a binary output stream to serve images. This was due to the character output stream already been implicitly opened in JSP. Although the exception is handled by Apache Tomcat, it is not known how the exception is handled by other Servlet containers.

- The disadvantage of Servlets over JSP's is that the developer must produce all HTML and dynamic content using Java code. In JSP, the PrintWriter stream is automatically opened enabling a mixture of HTML and code, similar to what happens in ASP. This means that the amount of code required to write JSP files is a lot less than the one required by Servlets. Naturally, developers comfortable writing HTML through Java might not see it as an important disadvantage.

A combination of JSP and Servlets is possible. Servlets can be used purely for image serving and JSP for all other areas of the application. Figure 7.2 illustrates this architecture. This approach is similar to the one illustrated in figure 7.1.

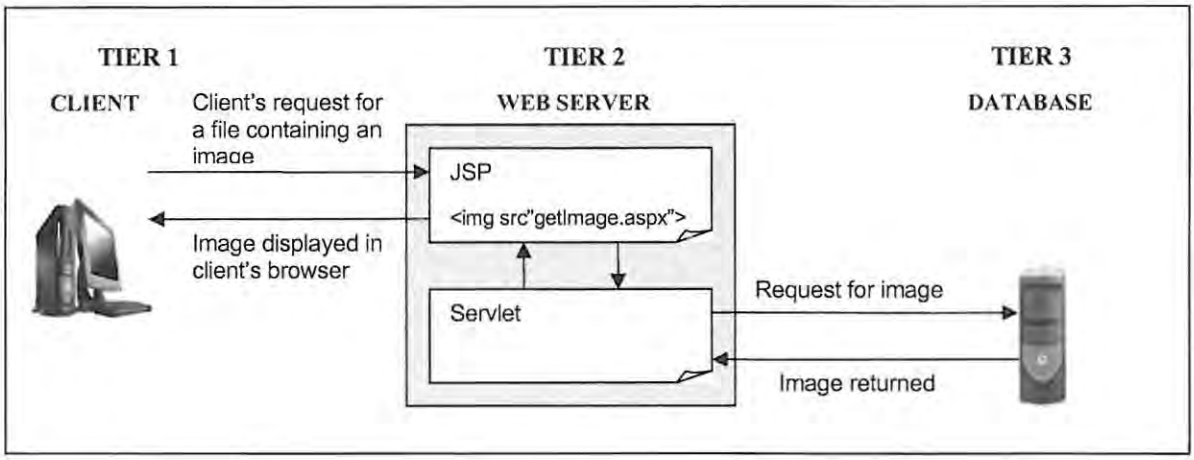

Figure 7.2 Using a combination of JSP and Servlets

As can be seen, JSP is used to serve HTML and other application data to the client. When an image is requested, the source parameter of the image tag points to a specified Servlet that retrieves the image from Oracle9*i* and delivers it for formatting in the JSP file.

Because JSP and Servlets use identical mechanisms for error handling and tracing, there is no need to contrast them as was done in the previous section for ASP and ASP.NET. The mechanisms are covered in the following section, where a comparison is made between ASP.NET and Java technologies.

7.2.3 Combining Microsoft and Java technologies

Finally, it is interesting to compare ASP.NET with Java technologies and to see if an integration of them is possible and useful.

- **Caching mechanisms:** ASP.NET provides *page*, *fragment* and *data* caching mechanisms whereas Java technologies do not provide any built-in dynamic content caching mechanisms. Since the caching of dynamic Web content can greatly improve the performance of Web applications when serving images requested from a database, ASP.NET is the recommended technology. It should be noted, however, that various application servers provide components

to cache dynamic data using JSP and Servlet technologies. The Oracle Applications Server is one such example.

- **Error Handling and Tracing:** Both ASP.NET and Java technologies provide well designed features and techniques for handling errors and tracing information flow. Both technologies support the *try-catch-finally* construct to handle exceptions and support the tracing of data and information flow during program execution.

- **Ease of Programming:** ASP.NET and Java technologies provide fully-fledged object-oriented techniques and paradigms for developing robust Web applications that support the separation of programming logic from page presentation. ASP.NET allows programmers to choose from a variety of programming languages from the .NET environment whereas JSP is developed in the Java programming language. The choice of language is most likely dependant on the developer's familiarity of the language and development environment. A key consideration is that JSP is both platform independent and based upon open-source software, whereas ASP.NET is integrated within the Microsoft Windows environment and subject to a commercial licence.

A combination of ASP.NET and Servlets is possible. Figure 7.3 illustrates a possible architecture for serving images from an Oracle9*i* database to Web-enabled clients based on the results discussed in section 7.1. The environment shown combines ASP.NET and Servlet technologies for developing a scalable and robust image serving Web applications.

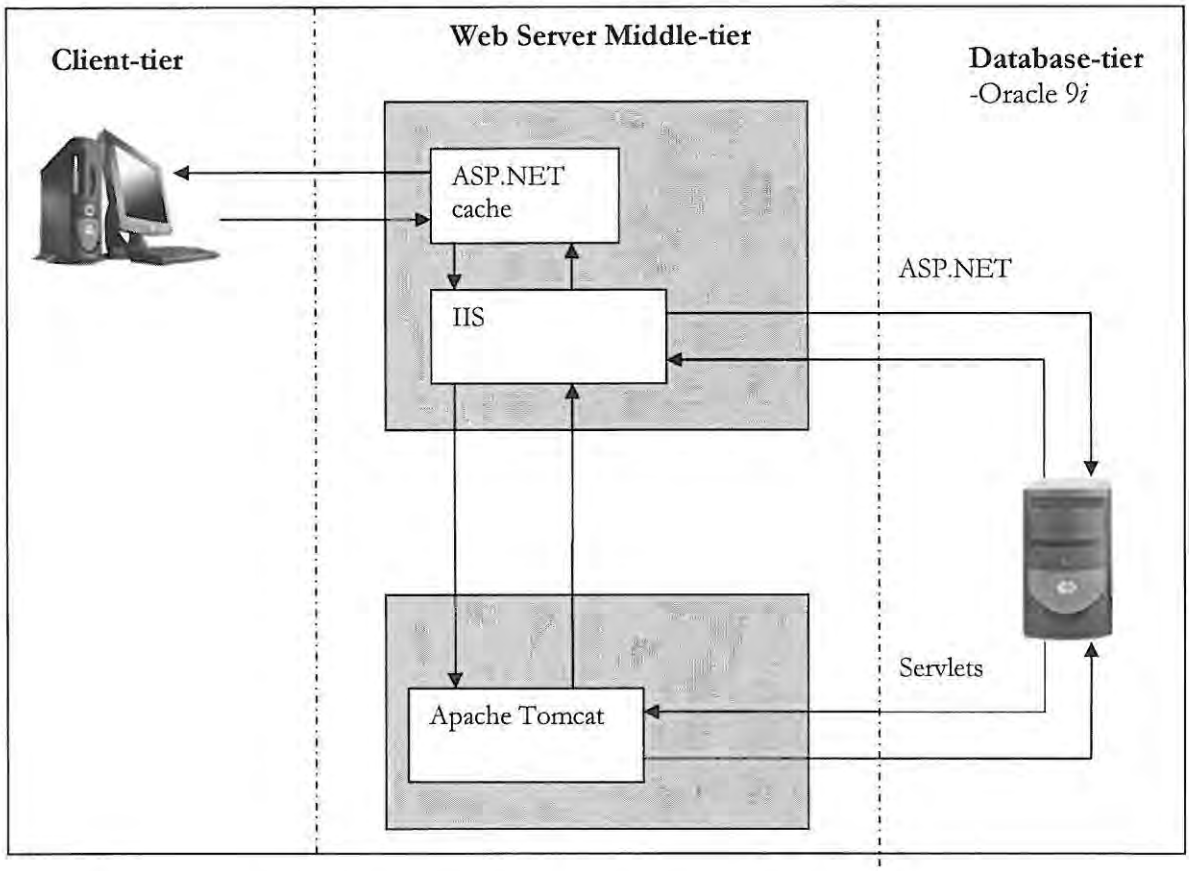

Figure 7.3 Using a combination of Servlet and ASP.NET technologies

The proposed architecture has two Web Servers in the middle-tier, IIS to run ASP.NET applications and Apache Tomcat to run Servlet applications. The architecture also contains a single Oracle9*i* database, preferably running on a separate machine. (The Web servers can run either on a single machine or on two separate machines.)

The proposed functionality of the environment for serving images from an Oracle9i database occurs as follows:

- A client (or multiple clients) makes an HTTP request for a Web page containing an image. All HTTP incoming requests are handled by ASP.NET.

- For minor applications, whereby the number of images requested from the Web application is relatively small, there is no need to route incoming requests to a Servlet application running within Apache Tomcat. This is because ASP.NET produces the fastest response time when serving a single image, of varying size, from Oracle9*i*. The caching abilities of ASP.NET should also be used to store frequently requested images.

- If the number of images requested from the Web application increases, the requests for the images is routed to the Servlet application running within Apache Tomcat. This is good for two reasons. Firstly, from the results presented earlier, Servlet applications respond faster than ASP.NET applications when the number of images exceeds 190. Secondly, by routing image requests to Apache Tomcat the processing load required by the application is shared between ASP.NET and Servlets. Essentially, Servlets handle the request for images from Oracle9*i* whilst ASP.NET processes all other areas of the Web application.

- Finally, all images are formatted within an ASP.NET file before being sent to the client's browser for viewing.

7.3 Summary

This chapter discussed the response times presented in chapter 6 and detailed the author's recommendations for the development of data-driven Web applications that serve images from an Oracle9*i* database.

- Based solely on performance, ASP or ASP.NET are recommended for serving an image that is less than 1MB. Images greater than 1MB are best served with ASP.NET. If the Web application is designed to serve a relatively small number of images, the developer should use ASP.NET. For applications that need to scale, designed for serving a large number of images, that developer should use Java technologies.

- Both ASP.NET and Java technologies provide well designed features and techniques for error handling, tracing, and ease of programming. ASP.NET was also identified as the only technology supporting dynamic data caching.

CHAPTER 8

CONCLUSION AND FUTURE WORK

"Take time to gather up the past so that you will be able to draw from your experiences and invest them in the future"

Jim Rohn

This thesis was motivated by a desire to make an informed choice of the technology to use when developing a Web-based application for serving multimedia over networks. The thesis focused on a single multimedia type, namely images, and the server-side Web technologies considered were Active Server Pages (ASP), Active Server Pages .NET (ASP.NET), Java Server Pages, and Java Servlets. The database used for the experiments was Oracle9*i*.

The selection of a particular Web technology is based on several factors. An important factor is performance. Performance tests were conducted on prototype applications developed using each Web technology, to measure the response time taken to serve a single image or multiple images, in a single request set, from the Oracle9*i* database to the user machines Web browser. The purpose of the performance tests was twofold. Firstly, to determine which driver/provider, possibly vendor-specific, generated the fastest response within each Web technology and, secondly, to determine which Web technology performed best when serving single and multiple images.

However, performance was not the only factor considered. During application development, various features relating to each Web technology and related

environment were documented, including caching mechanisms, error handling and tracing, and ease of programming. These features were identified as contributing to a developer's decision in selecting one Web technology over.

This chapter brings to a conclusion this thesis on data-driven multimedia Web applications. It summarizes the most important findings of the research, providing a summary of the performance results as well as a comparative assessment of each Web technology.

8.1 Summary of work covered

Substantial experience with the development and deployment of data-driven multimedia Web applications was gained during the course of the research and the project has made several useful findings.

8.1.1 Performance

The following observations, based on performance tests, were made.

Single Image Serving

- ASP and ASP.NET perform almost identically when serving a single image that is less than +- 1MB from an Oracle9*i* database. Thereafter, ASP.NET slightly outperforms ASP as the size of the served image increases. JSP and Servlets are consistently slower than ASP and ASP.NET when serving a single image, of varying size.

- The Oracle ODBC and Oracle OLE DB drivers perform almost identically when serving a single image of varying size in ASP. The Microsoft .NET Framework Data Provider outperforms both the OLE DB Data Provider (with associated Oracle OLE DB driver) and ODBC .NET Data Provider (with associated Oracle ODBC driver) in ASP.NET. In JSP and Servlet technologies, the Oracle Thin JDBC driver outperforms the Oracle OCI

JDBC driver when the image to be served is less than approximately 1 MB in size. Thereafter, the Oracle OCI JDBC driver outperforms the Oracle JDBC driver as the size of the image is increased.

Multiple Image Serving

- ASP and ASP.NET perform equally well when serving approximately 50 images, in a single request set, from an Oracle9*i* database. Thereafter, ASP.NET outperforms ASP as the number of served images increases. JSP and Servlets perform slower than ASP and ASP.NET when serving less than approximately 70 images when compared to ASP, and 190 images when compared to ASP.NET. Thereafter, JSP and Servlets outperform ASP and ASP.NET as the number of served images increases.

- The Oracle OLE DB driver consistently outperforms the Oracle ODBC driver in ASP. The OLE DB .NET Data Provider (with associated Oracle OLE DB driver) outperforms both the Microsoft .NET Framework Provider and the ODBC .NET Data Provider (with associated Oracle ODBC driver) in ASP.NET. The Oracle OCI JDBC driver outperforms the Oracle Thin JDBC driver in JSP and Servlets.

Other findings

- JSP and Servlet technologies perform almost identically when serving single or multiple images from Oracle9*i*. The first request for a JSP file where it is compiled and stored as a Servlet class was not considered.

- ASP.NET is the only technology providing inherent mechanisms to cache dynamic data from Oracle9*i*. ASP, JSP, and Servlets have no inherent mechanisms to cache dynamic data. ASP relies on IIS to cache interpreted script files and scripting engine instances. If used appropriately, the

caching mechanisms available within ASP.NET can greatly improve the performance of a Web.

8.1.2 Language features and development environment

Apart from performance considerations, several observations were made on the various Web technologies based on error handling, tracing, and ease of programming.

Error Handling and Tracing

- ASP provides an unstructured approach to handling errors through the use of the ASPError object and the *On Error Resume Next* construct. Both ASP.NET and Java provide well designed features and techniques for handling errors and tracing information flow. Both technologies support the *try-catch-finally* construct to handle exceptions and support the tracing of data and information flow during program execution.

Ease of Programming

- ASP, which uses object-aware scripting languages, provides a relatively simple means to serve images using ADO objects and vendor supplied drivers. ASP, however, does not provide a clear separation between programming logic and user interface mechanisms. Much of an ASP applications advanced functionality is often included as third party components and COM objects.

- ASP.NET and Java technologies provide fully-fledged object-oriented techniques and paradigms for developing robust Web applications that support the separation of programming logic from page presentation. ASP.NET allows programmers to choose from a variety of programming languages from the .NET environment whereas JSP and Servlets are developed in the Java programming language.

- The major advantage of Servlets over JSP's is that a developer has more control over the application environment. The disadvantage of Servlets over JSP's is the fact of a developer having to produce all HTML and dynamic content using Java code. This means that the amount of code required to write JSP files is a lot less than the one required by Servlets. Naturally, developers comfortable writing HTML through Java might not see it as an important disadvantage.

8.1.3 Technology selection and integration

Before the advent of ASP.NET, Java technologies seemed more suited to build scalable and robust Web solutions than ASP. With the advent of ASP.NET, both Java technologies and ASP.NET provide fully functional object-oriented environments for building Web applications.

The choice of selecting a Web technology is still most likely dependent on the developer's familiarity of the language and development environment. A key consideration is that Java technologies are both platform independent and free, whereas ASP.NET is integrated within the Microsoft Windows environment and subject to a commercial licence.

A combination of Web technologies is possible. In this thesis, an architecture was presented which integrates both ASP.NET and Servlet technologies for serving images from an Oracle9*i* database to browser-enabled client's. Based on the performance results discussed in the thesis, it was recommended to use ASP.NET for Web applications that serve a small number of images. As the Web application expands and the number of images to be served increases, Servlets should be used purely for the image serving process and ASP.NET for all other areas of application processing.

8.2 Future work

The work conducted in this thesis focused on a single area of multimedia content distribution, namely images, and the selected Web technologies were ASP, ASP.NET, JSP, and Servlets.

- An extension to the research can include an investigation into other multimedia types. These would include the streaming of video and audio from the Oracle9*i* database using streaming servers such as RealNetworks and Microsoft Media Services. The streaming of multimedia would include the Oracle *inter*Media Plug-in 2.0 for RealNetworks Streaming Servers. The Oracle *inter*Media Plug-in 2.0 for RealNetworks Streaming Servers allows RealNetworks servers to stream multimedia content to a client directly from an Oracle database (Oracle Corporation, 2004).

- In section 3.4 it was mentioned that the Oracle Data Provider for .NET (ODP.NET) was not tested because the Oracle Client (Version 1) did not support the provider in ASP.NET. An extension to the work conducted in this thesis would be to test the ODP.NET.

- Lastly, the use of Application servers on the middle-tier in place of stand-alone Web servers could be considered. The Oracle9*i* Application Server is one such example and has mechanisms such as a Web caching component for JSP and Servlet applications

REFERENCES

Ahmed, M., Garrett, C., Faircloth, J., Payne, C., DotThatCom.com, Lee, W. M., & Ortiz, J. 2002, *ASP.NET Web Developer's Guide.* Syngress Publications. ISBN: 1-928994-51-2

Barthalo, P. *The Word Spreads: Developers Abandon CGI in Favour of Servlets.* 2003. Available [On-line]: http://java.sun.com/features/1998/03/inetbiz.html Last accessed: 28/04/04

DeSoto, A. *Using the Beans Development Kit 1.0.* 1997. Available [On-line]: http://java.sun.com/products/javabeans/docs/Tutorial-Sep97.pdf Last accessed: 28/04/04

Esposito, D. *A Soft Landing to ADO.NET.* SQLServer Magazine. 2002. Available [On-line]: http://www.winnetmag.com/SQLServer/Article/ArticleID/24025/24025.html Last accessed: 28/04/04

Gladwin, L. C. *Backgrounder ASP and JSP: Overview of the technology.* 2000. Available [On-line]: http://www.itworld.com/AppDev/4065/CWSTO58756/ Last accessed: 28/04/04

Hall, M. 2001, *Core Servlets and Java Server Pages.* Rachel Borden. ISBN: 0-13-0893404

Ixia Corporation. *Qcheck: Performance Application.* 2004. Available [On-line]: http://www.qcheck.net Last accessed: 28/04/04

Leake, G. *Using .NET Framework Data Provider for Oracle to Improve .NET Application Performance.* 2002. Available [On-line]: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/manprooracperf.asp Last accessed: 28/04/04

Lee, W. *The Evolution of Data Access Technologies: Understanding the technology to use.* SQLServer Magazine. 2002. Available [On-line]: http://www.winnetmag.com/SQLServer/Article/ArticleID/23988/23988.html Last accessed: 28/04/04

Mader, L. *Using the Component Object Model (COM).* 1996. Available [On-line]: http://www.puremeta.com/ole/olepaper.html Last accessed: Second Semester 2003

Mahmoud, Q. H. *Servlets and JSP Pages Best Practices.* 2003. Available [On-line]: http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/ Last accessed: 28/04/04

Mauro, J. *Oracle interMedia: Managing Multimedia Content.* 2001a. Available [On-line]: http://otn.oracle.com/products/oracle9i/pdf/managing_multimedia_intermedia.pdf Last accessed: 28/04/04

Mauro, J. *Oracle interMedia: Services for the Management of Media-rich Internet Content.* 2001b. Available [On-line]: http://www.oracle.com Last accessed: Second Semester 2003

Meier, J. D. *Improving ASP Application Performance.* 2000. Available [On-line]: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnserv/html/server03272000.asp Last accessed: Second Semester 2003

Michalas, A., Zoi, V., Sotiropoulos, N., Mitrou, N., Loumos, V., & Kayafas, E. 2000, "A comparison of multimedia application development platforms towards the object Web", *Elsevier Computer Standards and Interfaces.*

Microsoft Corporation. *ActiveX.* 1997. Available [On-line]: http://www.host-web.fr/iishelp/adc/docs/adcdef01_41.htm Last accessed: 28/04/04

Microsoft Corporation. *Active Server Pages.* 2001a. Available [On-line]: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cdo/html/_olemsg_about_active_server_pages.asp Last accessed: 28/04/04

Microsoft Corporation 2001b, *Implementing Sun Microsystem's Java Pet Store J2EE Blueprint Application using Microsoft .NET.*

Microsoft Corporation. *Microsoft .NET vs. Sun Microsystems's J2EE: The NileEcommerce Application Server Benchmark.* 2001c. Available [On-line]: http://www.gotdotnet.com/compare Last accessed: Second Semester 2003

Microsoft Corporation 2002, *Introduction to Microsoft ASP.NET (MSDN training).*

Microsoft Corporation 2003, *Microsoft Internet Information Services 5.0 Documentation. Microsoft Press*

Microsoft Corporation. *ASP Template Caching*. 2004a. Available [On-line]: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/iis/iis6/proddocs/resguide/iisrg_sca_leaw.asp Last accessed: Second Semester 2003

Microsoft Corporation. *Script Engine Caching*. 2004b. Available [On-line]: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/iis/iis6/proddocs/resguide/iisrg_sca_xxvy.asp Last accessed: Second Semester 2003

Mirage, M. *Migrating from classic ASP to ASP.NET*. 2003. Available [On-line]: www.microsoft.com/taiwan/msdn/seminar/aspDotNet/ASP-to-ASPdotNET.ppt Last accessed: 28/04/04

Mitchell, S. & Atkinson, J. 2000, *Active Server Pages 3.0*. Sams Publishing. ISBN: 0-672-31863-6

Moore, D. *Best Practices for ADO.NET Development*. 2002. Available [On-line]: http://www.microsoft.com/usa/presentations/QTB_ADO-NET_BestPractices.ppt Last accessed: 28/04/04

Oracle Corporation. *Oracle® interMedia Audio, Image, and Video: User's Guide and Reference (Release 1)*. 2000b.

Oracle Corporation. *Oracle® interMedia Audio, Image, and Video: User's Guide and Reference (Release 2)*. 2000a.

Oracle Corporation. *Oracle9i Application Server Business White Paper*. 2001a. Available [On-line]: http://otn.oracle.com Last accessed: Second Semester 2003

Oracle Corporation. *Oracle9i: JDBC Developer's Guide and Reference*. Part No. A90211-01. 2001b. Available [On-line]: http://otn.oracle.com Last accessed: Second Semester 2003

Oracle Corporation. *Oracle9iAS J2EE Performance Study Results*. 2001. http://otn.oracle.com/tech/java/oc4j/pdf/java_performance_results.pdf Last accessed: 28/04/04

Oracle Corporation. *Delivering Rich Media Java Applications on Oracle9i Release 2.* 2002a. Available [On-line]: http://otn.oracle.com/products/intermedia/pdf/imedia_doc_twp.pdf Last accessed: 28/04/04

Oracle Corporation. *Oracle interMedia Java Classes for Servlets and JSPs.* 2002b. Available [On-line]: http://download-west.oracle.com/otn_hosted_doc/products/intermedia/htdocs/intermedia_servlet_jsp_software/oracle/ord/im/OrdHttpJspResponseHandler.html Last accessed: Second Semester 2003

Oracle Corporation. *Oracle9i Application Server.* 2002c. Available [On-line]: http://otn.oracle.com Last accessed: Second Semester 2003

Oracle Corporation. *Oracle Call Interface.* 2003. Available [On-line]: http://otn.oracle.com/tech/oci/index.html Last accessed: 28/04/04

Oracle Corporation. *interMedia Plugins for RealNetworks Streaming Servers on Windows and Solaris.* 2004. Available [On-line]: http://otn.oracle.com/software/products/intermedia/htdocs/descriptions/real.html Last accessed: 28/04/04

Platt, D. S. 2001, *Introducing Microsoft .NET.* Microsoft Press. ISBN: 0-7356-1377-X

Prasad, S. G. e. a., Kunisetty, S. K., & Basu, J. 2000, *Developing Web Applications Using Servlets and Java Server Pages on the Oracle Internet Platform.*

Roman, E. 1999, *Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition.* John Wiley & Sons, Inc.. ISBN: 0-471-33229-1

Sun Microsystems. *Java Pet Store 1.3.2 Release .* 2003a. Available [On-line]: http://developer.java.sun.com/developer/releases/petstore/ Last accessed: 28/04/04

Sun Microsystems. *JDBC Data Access API.* 2003b. Available [On-line]: http://servlet.java.sun.com/products/jdbc/drivers Last accessed: 28/04/04

The Middleware Company. *J2EE and .NET (RELOADED) Yet Another Performance Case Study.* 2003. Available [On-line]: http://gotdotnet.com/team/compare/Middleware30.pdf Last accessed: 28/04/04

Tice, E. *Use of OutputStream from JSP causes IllegalStateException*. 2000. Available [On-line]: http://archives.real-time.com/pipermail/tomcat-users/2000-May/007539.html Last accessed: 28/04/04

Wassef, M. *ASP vs. JSP: Overview of the Technology*. 2000. Available [On-line]: http://www.palestinewebdevelopers.com/ser_aspjsp.asp Last accessed: 28/04/04

White, S., Fisher, M., Cattell, R., Hamilton, G., & Hapner, M. 2002, *JDBC API Tutorial and Reference: Universal Data Access for the Java 2 Platform*. Addison Wesley, ISBN 0-201-43328-1 edn,