INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Comparing Electronic Commerce Solutions for Small Businesses

Yu Xing

School of Computer Science

McGill University, Montreal

March 2001

Thesis submitted to the

Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Science

Abstract

E-commerce is a new way of doing business. It is becoming increasingly important to everybody. The objective of this thesis is to compare various design approaches and to find out the best solution for small businesses. Some commonly used technologies in developing e-commerce systems are introduced in the first three chapters. The topics covered are from Web standards and protocols to Web planning and design, from Web servers to server side programming, and most importantly, Java technology and IBM's WebSphere. Based on these technologies, two solutions are presented: building from scratch with Java servlets and building with IBM's WebSphere. We develop an online store with each of the solutions. The last part of the thesis is a comprehensive comparison. Time, complexity, performance and cost are evaluated in this comparison. A simple conclusion is drawn afterwards with our suggestions to small businesses regarding which solution they should choose.

Résumé

Le Commerce Électronique est un nouveau moyen de faire le commerce. Il devient de plus en plus important pour tout le monde. L'objective de cette thèse est de comparer plusieurs solutions et de trouver la meilleure pour les petits commerces. Certaines technologies utilisées communément pour développer les systèmes de commerce électronique sont introduites dans les trois premiers chapitres. Les sujets couvrent des normes et des protocoles de Web aux planifications et aux conceptions de Web, des serveurs de Web aux programmations de côté serveur, et surtout la technologie de Java et Websphere de IBM. Basé sur ces technologies, deux solutions sont présentées : la conception à partir de zéro utilisant les servlets de Java et WebSphere de IBM. Nous avons développé un magasin-sur-ligne avec ces deux solutions. La dernière partie est une comparaison compréhensive. Le temps, la complexité, la performance et le coût sont présentés dans cette comparaison. Une simple conclusion est retirée après, avec nos suggestions sur quelle solution les petits commerces doivent prendre.

Acknowledgements

I wish to thank my thesis supervisor Professor Monty Newborn for his guidance, advice and encouragement throughout the research. This thesis would not have been possible without his support.

I also truly thank Gao Liqian for her collaboration as a partner in this research.

I would like to thank the School of Computer Science for the graduate courses and the research environment. Thanks to Teresa De Angelis, Vicki Keirl, Lise Minogue and Lucy St-James for their great help.

Many thanks also to Zhou Xiaowen, Zhao Hongyu, Babak Mahdavi, Nagi Basha, Wang Qijia, and Lin Yi. Their support in the survey provided me with plenty of information necessary in this thesis.

Finally, I wish to thank my husband Liu Tao for his support and encouragement during my study.

Table of Contents

Table of Figures

Chapter 1

Introduction

The Internet is one of the greatest inventions of the 20th century. It created a new world – a virtual Internet world. Based on the Internet, electronic commerce (e-commerce) emerged as another important technical trend. The power of e-commerce affects not only commerce, but also everybody's life style. Due to its increasingly important role in businesses, several different solutions have been developed. The objective of this thesis is to compare various solutions and to find out the best solution for small businesses.

In this chapter, we will discuss some background information about e-commerce, such as its development history and architecture. We will briefly talk about various mainstream solutions later in this chapter, and we will be focusing on two of the good solutions. In Chapter two and three, some necessary technologies will be discussed. Most of these technologies are used in Chapter four to develop two different solutions for a virtual E-business. We compare these two solutions comprehensively in Chapter five and draw a conclusion eventually.

1.1 The Development of E-commerce

E-commerce has a longer history than just the Internet. Business-centered e-commerce began more than two decades ago with the introduction of electronic data interchange (EDI) between firms. EDI serves as the interface for sending and receiving order, delivery and payment information. Consumer-oriented e-commerce also has a long history. We use automatic teller machines or credit cards for transactions. These transactions are actually carried out electronically. EDI and ATM, however, operate in a closed system. They are a convenient communications medium, strictly between the authorized parties.

The World Wide Web (WWW), created at the CERN Lab for Particle Physics in Geneva in 1991, opened up a new age by combining the open Internet and the easy user interface. The rapid development of World Wide Web makes the term "electronic commerce" evolved from its original meaning of electronic shopping to mean all aspects of business and market processes enabled by the Internet and the World Wide Web technologies.

Narrowly speaking, e-commerce means selling and buying products and services through Web storefronts. In a way, e-commerce is similar to catalog shopping or cable TV shopping. Products being traded include physical products (such as books and cosmetics) and digital products (such as audio, video, and software).

In a wider definition, e-commerce is not limited to buying and selling products online. Along with customers, an online business will also find its suppliers, accountants, payment services, government agencies and competitors online. These online partners demand changes in the way they do business from production to consumption. Along with online selling, e-commerce will lead to significant changes in the way products are customized, distributed and exchanged and the way consumers search and bargain for products and services and consume them.

In short, e-commerce is a new way of doing business. Business-to-consumer processes, within-business processes (such as manufacturing, inventory, corporate financial management), and business-to-business processes (such as supply-chain management, bidding) are all affected by the e-commerce technology. Many leading software companies have their own definition of e-commerce [23]. For example:
Intel's definition: e-commerce = electronic market + electronic transaction + electronic service;
IBM's definition: e-commerce = Information Technology + Web + transaction;
HP's definition: e-commerce is an electronic way to accomplish commercial transactions.

E-commerce sites can be divided into two main types: Business-to-Consumer (normally called B2C) and Business-to-Business (B2B).

For B2C sites, an individual shopper browses through an online catalog, chooses some goods he/she wants to buy, and then enters the credit card number to place an order. The shopping flow is quite simple.

For B2B sites, the target consumers are business people. They order goods for their office or their business. Their orders may need to be approved before the order can be placed. They may not need to supply a credit card number. Instead, they enter a purchase order number. They usually need to register before shopping. This thesis will mainly focus on B2C sites.

1.2 The Architecture of E-commerce

Although every e-commerce system has a different background, uses different software and serves different goals, they all have a common architecture.

1.2.1 Common Electronic Commerce Components

Most e-commerce systems include a Web server, an application server, databases, browsers, and usually a backend system. The relationships between these components are shown in Figure 1.1.

- Web Server: Web servers use HTTP (Hypertext Transfer Protocol) over TCP/IP to send and receive HTTP requests for pages and return the data to clients' browsers. HTTP is the protocol that transfers hypertext files (such as html files) across the Web. The most popular Web servers include Lotus Go, NES, IIS, Apache, etc.
- Database Server: Database servers run database management software. Databases are kept in a directory that is part of a Web site, and accessed via server side programming. Popular databases include DB2, Oracle, Access, Informix, etc.

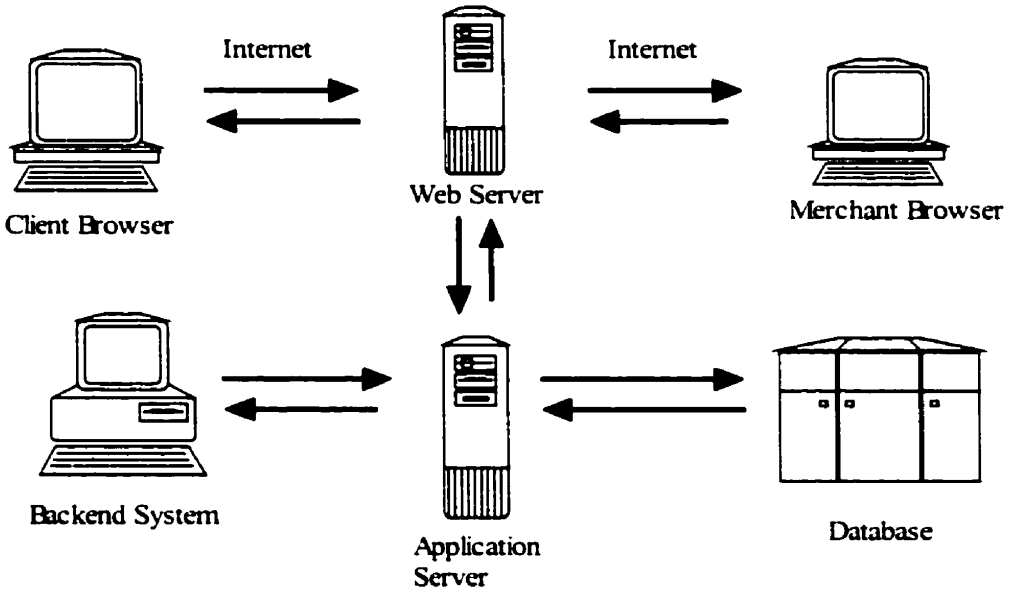

Figure 1.1 The Architecture of E-commerce

- Application Server: Application servers hold and support a set of addressable resources. The resources could be JSPs, EJB, or CGI programs depending on

different systems. Some examples of application servers could be WebSphere Application Server, MS merchant server.

- Connection to Backend System: Backend systems are usually the existing merchant systems containing business logic. Databases are kept in a directory that is part of a Web site, and accessed via CGI or ASP programming. EDI, SAP, IBM MQ series, CICS and IMS are normally used to connect to backend systems.
- Browser: Browsers are tools used by both consumer and merchant to connect to the e-commerce system via Internet. The most popular browsers are Internet Explorer and Netscape.

Most of the above components will be further discussed in the following chapters.

1.2.2 Electronic Commerce Development Team

The previous section talked about the software and hardware architecture of e-commerce system. More important than that is the human efforts in setting up an e-commerce system. To develop a fully functional e-commerce system, a development team is usually necessary. The team can be divided into three groups,

- Internet Marketing and Strategy Formulation:
  This group works to provide Internet marketing consultation and innovative site marketing strategies.
- Web Site Design & Development
  This group contains both artists and programmers. Members work together to present site layout and design, create graphics, animations and video production. Programmers would mainly work on HTML, Java, JavaScript, Shockwave and other tools.
- Systems Integration & Programming
  The group works on database design and programming, multi-platform API programming and integration of Web-collected data with existing backend systems.

1.3 The Importance of E-commerce

To oversimplify, e-commerce is actually "Commerce on the Internet". The importance of e-commerce is largely depended on the growth of the Internet.

1.3.1 The Growth of the Internet

Here is a closer look at this dramatic growth and promising future of the Internet,

- Internet traffic is doubling every 100 days, annual growth rate more than 700 percent.
- The number of Internet users soared to more than 100 million in 1997 from 3 million in 1993
- Business-to-business transactions on the Internet could surpass $300 billion by 2002
- By the end of 1997, 10 million people in the United States and Canada had purchased something on the Web, up from 7.4 million six months earlier
- Sixteen percent of all new car and truck buyers used the Internet as part of their shopping process in 1997. That figure will grow to 21 percent by 2000

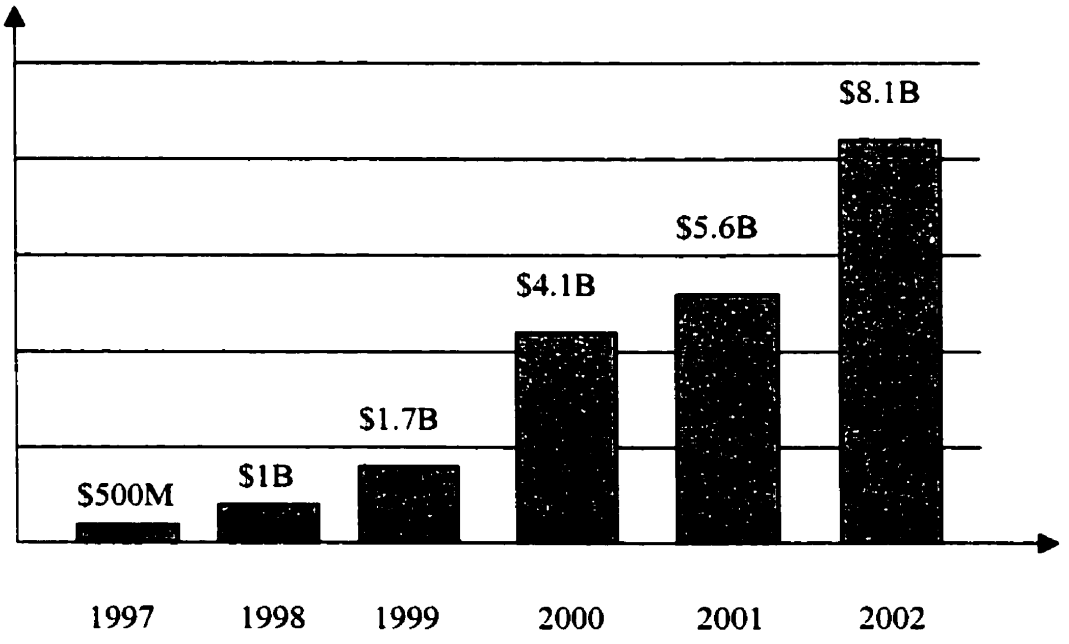

Figure 1.2 Online Advertising, Direct Marketing Revenues

- Total Internet commerce in the US will be close to $250 billion in 2002.
- First-year return on investment for Web applications was 245 percent, compared to popular client/server applications at 95 percent.
- Availability of venture capital to invest in Internet companies will nearly double this year to over $8 billion
- Online advertising, direct marketing revenues are shown in Figure 1.2.

[14]

1.3.2. Benefits of E-commerce

E-commerce has a lot of unique benefits that traditional commerce could not achieve. First of all, e-commerce provides new channels for delivery of products. Traditionally, products can only reach consumer via retailers, store branches. Now with the new technology of e-commerce, consumers can browse over the Internet and order online. Products can be delivered immediately.

Secondly, distribution costs are reduced. Traditional distribution channels adds 135% to the cost of a manufactured item; e-commerce enabled manufacturer to consumer will only add 10% (Manufacturing News, March 17, 1998). Traditional bank transaction costs are $1.07; e-commerce transaction will cost about 1 cent. Traditional airline ticket costs $8 to process; e-ticket will cost just $1(Booz-Allen & Hamilton Study). These differences make lower cost and higher efficiency possible.

Thirdly, customer services are improved. Like distribution costs, service costs are also dramatically lowered with the help of e-commerce. Merchants and manufacturers are able to provide high-quality services, and even personalized services. For example, many online stores provide consumers with customizable storefronts according to every consumer's wish, which is not possible in traditional commerce.

With e-commerce, stores are no longer limited by geography and time. Any online store can be accessed by customers all over the world, 24 hours a day, 365 days a year.

1.3.3 E-commerce is Everybody's Business

Until now, a big percentage of the e-commerce participants are so-called digital product companies such as those in publishing, software, entertainment and information industries. However, the power of e-commerce affects all of us.

More and more small businesses have become involved in e-commerce. Statistics gathered by Access Media International are shown in Table 1.1.

	1997	1998	2000
Number of small businesses that have Web sites	900,000	2M	2.7M
Number of small businesses conducting e-commerce	400,000	600,000	1.3M
E-commerce sales captured by these small businesses	$3.5B	$7.5B	$25B

Table 1.1 Small Business Leverage Electronic Storefronts

Even seemingly mundane bookstores face different challenges in the electronic marketplace. The case of Amazon.com vs. Barnes & Noble shows the definition of "stores" has to be re-evaluated. Distributing books require numerous local branches to provide convenient access to customers. However, with the help of e-commerce, Amazon.com has become the leading online bookstore, ranking itself as the "largest bookstore" on earth not by opening numerous branch stores but via the Internet. The "biggest bookstore", Barnes & Noble with a towering share of revenues and physical bookstores, has been forced to respond to Amazon.com's challenge by opening its own Web store.

E-commerce has changed not only business operations, but also every individual in their way of living. Through Web TV, the way we watch TV news and entertainment programs changed. Changes in telecommunication such as mobile e-commerce affect the way we receive information. We do shopping online, we pay bills online, and we make phone calls online. There is nothing we cannot find on the Internet, there is almost nothing not yet been affected by e-commerce.

1.4 E-commerce Solutions

E-commerce is important to every business, but e-commerce does not mean the same thing to every business. Different companies have different requirements on their Internet activities; therefore, they would require diverse e-commerce solutions. For many companies, the only aim of their Internet activities is to create an Internet presence, which usually is somewhat equivalent of an "electronic brochure". This may include a description of the company, a description of the company's product lines and contact information. If properly advertised, even these simple sites can bring new business to the companies. More companies need commerce-enabled Web sites. Internet users are becoming more and more willing to make purchases online. If the users could not buy the products or services immediately, the selling opportunities may be missed. Some large sized or well-organized companies need more complicated e-commerce solutions that can reformat and improve their business processes.

Based on these different requirements, there are three main kinds of solutions to choose from: (1) using Web-based store building services, (2) building from scratch or (3) using ready-made e-commerce packages.

1.4.1 Web-Based Store Building Services

Many small businesses are looking for a way to launch e-commerce without having to get too involved in the technical issues of online Web site development. Several companies have begun offering all-in-one Web-based solutions for setting up an online store. These

solutions involve building sites entirely online so there is no software to download, install or configure. What businesses need to do is just pick a look, fill in product information and choose some settings. These solutions are frequently inexpensive and include many common features. They have so-called "instant storefront" because they are fast. The whole store is set up and administered through the Web. The development cost is close to zero. However, for all the services, there is a monthly maintenance fee of between $50 and $200.

The downside is that these services provide very limited functions. They may not support the features a specific business wants. They saved the work of dealing with the complexities of installation and configuring, but that's because they only offer limited ways to do these things.

Some successful examples of these services are "Microsoft bCentral" at http://www.bcentral.com, "Yahoo Store" at http://www.yahoostore.com, "Bigstep" at http://www.bigstep.com, and "freemerchant" at http://www. freemerchant.com.

These online services are still in their beginning stages, but are making progress towards providing a simple and cheap solution to build online stores. Now, they are only a good choice for very small stores.

1.4.2 Building the System from Scratch

An alternative to these Web-based services is to build the system from scratch using some of the many software tools available. The advantage is that this approach could provide the exact solution a business needs with greater flexibilities. The business could build the features and functions they need to be unique and competitive in the marketplace.

This solution requires expertise, time, and a sizable budget to develop an e-commerce system. The developers need to work from interface design to database design, from main

application development to integrating tax, shipping and payment processing software modules. There are a series of tools and standards in developing Web systems. An e-commerce system can be developed in almost any programming language. Many early Web-based business interfaces were created in PERL or C++. More recently, a lot of work has been done with Microsoft's Active Server Pages (ASP) and Sun's Java Server Pages (JSP). Also, new standards like CORBA (Common Object Request Broker Architecture) and ActiveX are becoming more mature.

1.4.3 Ready-made E-commerce Packages

The high end solution is to buy a ready-made software package. Medium and large companies require more comprehensive solutions. They need an e-commerce system that can streamline their existing business processes, improve their efficiency and handle heavy load transactions. These companies will usually choose dedicated or co-located hosting and a whole set of e-commerce package. However, this option will generally require more technical knowledge. The installation and configuration of these packages are normally much more complicated than in the previous solution, let alone the programming.

The advantage of these packages is that there are not normally many restrictions on the applications or functionality they can offer to the Web site. The only restrictions are what the company can afford and what the technical experts can manage.

Almost every leading software company has its own integrated e-commerce solutions package. These packages include Microsoft's Site Server Commerce Edition, IBM's WebSphere, Netscape's CommerceXpert, Intel's iCat Pro. IBM's WebSphere, for example, has facilities for establishing and maintaining a large e-commerce site including payment manager, security server, catalog architecture management, database and sophisticated tools for analyzing usage patterns. These packages varied in price from $10,000 to $50,000.

1.5 Summary

In this chapter, we talked about the development, the architecture and the importance of e-commerce. Some possible e-commerce solutions are also discussed. These are the basis of this thesis.

As mentioned earlier, there are basically three kinds of solutions to e-commerce. The first one, "Web-Based Store Building Services", is pretty limited and involves little technical issues. Although it could sometimes be a good choice for very small stores, it is out of the scope of this thesis and we will not discuss it furthermore.

The second kind, namely "Building from Scratch", has many implementation possibilities and involves a whole bunch of technologies. We will talk about some commonly used technologies in Chapter two. After a brief comparison between these technologies, we will present a detailed solution in Chapter four.

The last solution is to use well-made packages. Many packages are available in the market. Although they each has some strong points, IBM's WebSphere turned out to be the most complete and reliable package. We will introduce WebSphere in Chapter three and develop a functional Web site with WebSphere in Chapter four.

For the two solutions presented in Chapter four, it is very hard to say which one is better. When evaluating these solutions, only considering the developing cost is far from enough. Various other factors should be considered, such as development time, maintenance fee, performance and a lot more. In Chapter five, we are going to compare the two solutions comprehensively in order to find out which one is better. Chapter six will be a conclusion and future work.

Chapter 2

Understanding the Technology

In this chapter, some essential technologies in e-commerce are discussed. These technologies include Web standards and protocols, the steps in setting up a Web site (Web planning, Web design, creating pictures and animations, dynamic Web site using JavaScript), database, Web server, server side programming (server side JavaScript, CGI, ASP, JSP) and Web security.

2.1 Web Development Overview

Before any introduction to the technology, a brief overview of the Web development is shown in Figure 2.1. We can see when all the technologies are developed and how they contributed to the World Wide Web.

The Web has three main development cycles. The first one is Hypertext Web, where Web servers served as URL based static HTML file servers. Users could access a static HTML page by a Web address. This structure works well in electronic publishing. The tremendous amount and diversity of hyperlinked information available on the World Wide Web has proved its success. Actually the new Web structures are all based on Hypertext Web.

However, the information on the Hypertext Web is static and is served directly from HTML files. The information in these files only changes when the site administrator updates the contents. Such static pages allow for little interactivity. They reduce much of Web exploring to a semi-passive activity.

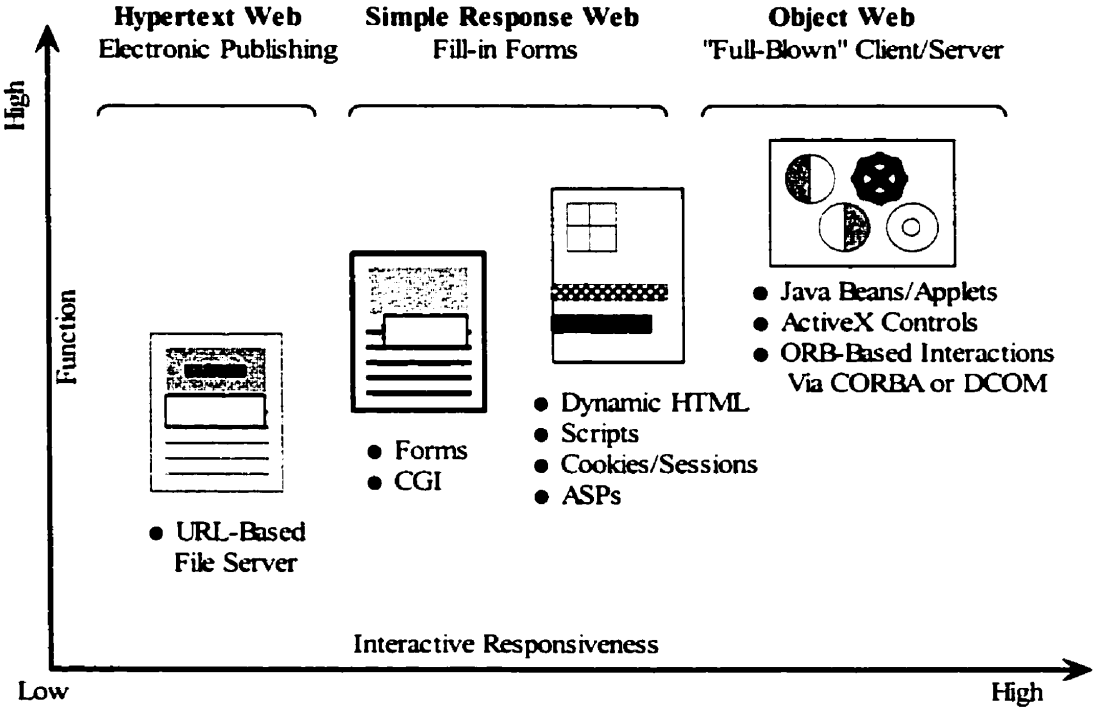

Figure 2.1 Web Site Development

As the user requirements increase, more interactivity becomes necessary. From online ordering, feedback mechanisms, database access, to searching capabilities, the most interesting Web sites depend on interactions between users and Web servers. Common Gateway Interface (CGI) protocols and Active Server Pages (ASP) are the most commonly used standards to implement interactivity. We can call these Web sites Simple Response Web sites.

Because the World Wide Web is developing at a rapid speed, all kinds of platforms and hardware are possible on the net. A platform-independent language like Java has become the new trend of Web development. Reusability has always been a big issue in software engineering. It also becomes more and more important in Web development. Due to the

previous two reasons, a so-called Object Web appears. The Object Web is based on some new technologies like Java Beans, Java Applets, JSP, ActiveX, and Common Object Request Broker Architecture (CORBA).

Most of the essential technologies mentioned above are discussed later in this chapter.

2.2 Web Standards and Protocols

The Web encompasses many standards and protocols. Users may access information using any one of them. More importantly, the Web hides protocol layers from the users. The users see a standard interface and don't have to know whether some information is coming from one protocol or another. In addition, the Web does not restrict the types of objects that can be sent. Although several standardized content types already exist (GIF images, MPEG video, HTML text, etc.), theoretically any content type can be transmitted over the Web.

The Web is primarily defined by four standards: URLs (Uniform Resource Locators), HTTP (Hypertext Transfer Protocol), HTML (Hypertext Markup Language), and CGI (Common Gateway Interface). Servers and clients on the Web use these standards as simple mechanisms for locating, accessing, and displaying information. Standards have also emerged for a Common Log Format, so that programs could be written to analyze the logs of any server. Security standards are the hottest issue under debate currently, but it will probably be a while before one encryption or authentication standard is widely accepted.

2.2.1 URL

URL stands for Uniform Resource Locators. It is a standard way of specifying the location of an object, typically a Web page on the Internet [25]. URLs are the form of addresses used on the World Wide Web. They are used in HTML documents to specify the target of a hyperlink, which is often another HTML document (possibly stored on

another computer). By means of a standardized addressing scheme, Uniform Resource Locators can be used to locate and retrieve information anywhere on the Internet. URLs can also be used to specify FTP file retrieval, find newsgroups and other data objects.

Here are some sample URLs:

```
http://www.greenecommerce.com/index.html
http://www.geocities.com/clickcall/bg.jpg
http://search.yahoo.com/bin/search?p=computer
http://www.geocities.com/clickcall/energica/product.htm#product1
gopher://infomcgill.mcgill.ca
mailto:xyu@cs.mcgill.ca
telnet://willy.cs.mcgill.ca
```

The part before the first colon specifies the access scheme or protocol. Commonly implemented schemes include ftp, http or gopher. The "file" scheme should only be used to refer to a file on the same host. Other less commonly used schemes include news, telnet or mailto (e-mail).

The part after the colon is interpreted according to the access scheme. In general, two slashes after the colon introduce a hostname (host:port is also valid, or for ftp, user:passwd@host or user@host). The port number is usually omitted and defaults to the standard port of the scheme. For example, port 80 for HTTP.

For an HTTP or FTP URL, the next part is a pathname, which is usually related to the pathname of a file on the server. The file can contain any type of data but only certain types are interpreted directly by most browsers. These include HTML and images in GIF or JPEG format. The file's type is given by a MIME type in the HTTP headers returned by the server, for example, "text/html" or "image/gif". The file type is usually also indicated by its file name extension. A file whose type is not recognized directly by the browser may be passed to an external "viewer" application, e.g. a sound player.

The last part of the URL may be a query string preceded by "?" or a "fragment identifier" preceded by "#". The later indicates a particular position within the specified document.

2.2.2 HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0 improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics. [26]

HTTP is a relatively simple, highly flexible protocol. To deliver requested information from a server to a client, HTTP defines a simple transaction that consists of four parts:

1. The client establishes a connection to the server.
2. The client issues a request to the server specifying a particular document to retrieve.
3. The server sends a response containing a status code and the text of the document, if it is available.
4. Either the client or the server disconnects.

One of the main goals of HTTP is to provide a simple algorithm that makes fast response times possible. To achieve this goal, HTTP is defined as a stateless protocol that does not retain information about a connection from one request to the next. A lot of protocols do retain state information, i.e. FTP. When a user uses FTP, he/she can change the directory he/she is working in, and the server remembers that directory when it gets his/her next request. With HTTP, on the other hand, the server does not remember the directory because the protocol does not allow the server to retain information from a previous connection.

HTTP is limited to one request per connection. Unlike other protocols such as FTP, the connection between server and client is broken after each request is made. Every time a client wants to fetch a document, it has to establish a new connection to the HTTP server. This is one of the main reasons why it takes so long to load HTML pages that have a lot of graphics even if those graphics are similar. A separate connection must be established for each graphic. Establishing a connection is not usually time-consuming, but it can seriously affect performance at distant or heavily loaded sites.

The primary difference between HTTP 0.9 and HTTP 1.0 is greater flexibility, which is made possible by two new enhancements. The first is the addition of transaction headers, and the second is the addition of several new methods. Headers make it possible to pass along information for facilitating authentication, encryption, and user identification. The new methods offer additional capabilities.

HTTP 1.0 adds headers to all transactions, whether they are client requests or server responses. When a client requests data, the request transaction header can include the name of the client software making the request (i.e., the browser type, whether it be Mosaic, Netscape Navigator, or another browser,), which type of the data the client understands (i.e., data formats), and what languages the client understands.

Similarly, when the server responds to a request, it can return a header along with the requested data. A response header can include information about the status of the request (whether it was successful or not), the length of the data being returned (i.e., the number of bytes transmitted, not including the header), the content type, the language of the content, and the date that the content was last modified.

When a client makes a request to a server, it chooses a requesting method based on the purpose. Under HTTP 0.9, the only valid method was GET. HTTP 1.0 adds six new methods: POST, HEAD, PUT, DELETE, LINK, and UNLINK.

2.2.3 HTML

HTML means HyperText Markup Language. It is a hypertext document format used on the World Wide Web. HTML is built on top of SGML (Standard General Markup Language). SGML is a generic markup language for representing documents. SGML is an International Standard that describes the relationship between a document's content and its structure. SGML allows document-based information to be shared and re-used across applications and computer platforms in an open format. [25] HTML is a subset of SGML. The World Wide Web Consortium (W3C) is the international standards body for HTML.

HTML contains commands, called "elements" or "tags", to mark text as headings, paragraphs, lists, quotations, and so on. A tag consists of a "<", a "directive" (case insensitive), and a ">". Matched pairs of directives, like "<TITLE>" and "</TITLE>" are used to define area of texts with a special place or style.

There are no rendering restrictions on browsers, so a browser can render the tagged text according to its own definition. For example, a piece of text can be identified as a first-level header with a pair of HTML tags. Each browser is free to render that text in whatever format it uses for first-level headers. For example, one browser might center first level headers, and another browser might underline them.

More important than semantic markups, HTML provides the capability to create hypertext links between one document and another document or object, or even between locations in a single document. Links are the information threads that form the structure of the World Wide Web. The HTML links are specified through URLs as referred to in Section 2.2.1.

Like any other programming language, features were added to HTML to meet needs that were not met under previous versions. HTML 1.0, the first version, was a basic document markup including links. HTML did not support forms until version 2.0. With version 3.0

comes support for tables and mathematical equations. Version 3.0 also offers much greater control over layout. Text can be wrapped around graphics and figures. New features gave the user even more control over the rendering of Web pages, and new tags allowed users to center text and specify font sizes. These new tags were not in the official HTML specification, but many have become de facto standards.

2.2.4 CGI

Although many Web sites are pure static information publishing, the Web is designed as an interactive medium. To make themselves more interactive, some sites generate HTML pages "on the fly" based on client input. Generally, this functionality is achieved through the CGI (Common Gateway Interface) protocol. CGI is not a programming language. It is a standard for running external programs from a World Wide Web HTTP server. CGI specifies how to pass arguments to the executing program as part of the HTTP request. It also defines a set of environment variables. To make the definition simpler, CGI actually defines an interface through which the server can pass information to a program, and the program can return information to the server.

Initially, each server had its own standards. Scripts written for one server could not be used on another server without major modification. But once the CGI standard was defined, it became possible to take a script or program written for one server and successfully run it on a different server. CGI is a good example of how standards can help bridge the incompatibilities inherent in systems with many participants who use lots of different software.

CGI programs make interactive features such as information gateways, feedback mechanisms, database access, ordering capabilities, personalized documents, and searching capabilities much more possible. It is the main tool for creating truly interactive experiences on the Web.

Commonly, the program will generate some HTML that will be passed back to the browser. It can also request URL redirection. CGI allows the returned HTML (or other document type) to depend in any arbitrary way on the request. The CGI program can, for example, access information in a database and format the results as HTML. A CGI program can be any program that can accept command line arguments. Some HTTP servers require CGI programs to reside in a special directory, often "/cgi-bin" but better servers provide ways to distinguish CGI programs so they can be kept in the same directories as the HTML files to which they are related.

2.2.5 Common Log Format

The Common Log Format is a standard format for access logs. It was developed so that programs could be written to analyze the logs of any server. The Common Log Format specifies what information is logged and in what sequence the information appears. The standard also describes delimiters, such as square brackets and quotation marks, which are used to help identify the data. For example, the standard calls for date and time information to be placed inside square brackets, and requests to be enclosed by quotation marks.

The server automatically gathers the available data into the log. It lists which data was requested, when it was requested, what host the user was on when requesting the data, and whether the request was successful. The log might also record user names, but it does that only if you have modified your server software or if the server automatically requires that requests for documents and objects be authenticated.

2.2.6 Security Measures

E-commerce is based on the Internet. Unlike traditional procedures, electronic transaction information transmits among clients, merchants and banks through the open Internet. The method of transmission raised a lot of potential security problems,

- Internet is an open system without a central control system. The net could be down or congested, which would lead to malfunction.
- Internet data transmission is based on TCP/IP. The protocol itself does not provide any security measure to insure no information is stolen during the transmission.
- Electronic information is virtual information. It is harder to identify the true origin and destination.

Both providers and customers on the Internet want to transfer credit card numbers and other private data securely. The Security Working Group was formed in December 1994 in San Jose by IETF. They worked on providing security services to HTTP. Three of the protocols were developed to meet the security needs of commercial providers on the Internet: Shen, S-HTTR (Secure HTTP) and SSL (Secure Sockets Layer). Each protocol works in a different way.

The Shen approach, calls for three separate security-related mechanisms:

- Weak authentication with low maintenance overhead and without patent or export restrictions
- Strong authentication via public-key exchange
- Strong encryption of message content

Shen is built on top of the existing HTTP 1.0 specifications. It simply extends HTTP by defining new HTTP headers that can be used in the request response to perform these functions.

The second security protocol, S-HTTP, was developed by Enterprises Integration Technologies (EIT). S-HTTP was designed to provide encryption and to authenticate the content, but it is not very popular yet.

The third protocol is SSL (Secure Socket Layer). SSL is base on PKI tech (Public Key Infrastructure). PKI tech is a classic algorithm. Now it is widely used on the Internet.

Normally, when a person (A) wants to send an encoded message to another person (B), he/she uses a password/key/sin/pin (secure identify number or personal identify number) to encrypt it: E((Plain Text), keyA) => Cipher Text. After person B receives the encoded message, B uses the same key to decode it: D((Cipher Text), keyB) => plain text. (keyA = KeyB, E and D are algorithms). How to send the key to the other person in a safe way is the big problem. Public key can solve this problem. Person A has a pair of keys, say key1 and key2, the message encoded by key1 can only be decoded by key2. E((Plain Text), key1) => Cipher Text, D((Cipher Text), key2) => plain text. Key1 is different than key2, and there is no way to find out key2 from key1. If B wants to get message from A, he/she let A know key1 (public key). Person A encodes message by key1, and sends it to B. B uses key2 (private key) to get the plain text. In this way, B won't have to worry about the key1 being peeked by any third part during transmission, since key1 (public key) is useless in decoding the cipher text. Only person B has key2, the private key. Only B can get the plain text.

Some Web address starts from "https://" instead of "http://". Https:// means this Web site uses SSL. The public key algorithm has some limit. It's relatively slow and cannot encrypt long message, so it normally works together with other encrypt algorithms,

1) The SSL server gives its public key (in certificate) to the browser.
2) The browser produces a fresh normal key, and encrypts it by the server public key.
3) The server decodes the normal key.
4) Now the server and browser get the same key. They start up a secure SSL session.

2.3 Web Planning and Design

Building a Web site is a complex process. A single Web site may have to fulfill many roles from document delivery to business process automation. It also has to be built to suit the needs of diverse groups including potential customers or departments in an organization. The work includes the efforts from not only technical people, but also business people, artists and etc. as described in Section 1.2. This thesis only focuses on the technical part. From the technical point of view, the work of setting up a Web site

ranges from planning, design, client side programming, database programming, to Web server set up and programming. Each step will be presented with more details in the following sections.

2.3.1 Web Planning

Like any other software development, the first step to set up a Web site is Requirement Analysis and Specification.

2.3.1.1 Environmental Requirements

The environmental requirements will influence the Web site in many ways. Before we begin to design a Web site, we need to know the end-user system requirements of browser, connection speed, and monitor size as well as the server-side requirements.

Determining the target platform for a Web site could be difficult. While some general assumptions about the target users can be made, surveying a sample of the user population or analyzing the statistical reports for the existing site can be very useful. For example, a consumer-oriented site is likely to be accessed via a modem connection, and the users probably have only 15-inch monitors. Conversely, a Web site for a large components company probably will be accessed primarily by well-equipped engineers with high-speed connections. [22]

One way to solve the problem is to identify the most likely least common denominator platform. However, it will be a waste for those who are using high-resolution monitors and high-speed connection users. Another option is to provide a separate site for each type of browser. It is possible to design a site that can sense connection bandwidth, browser type, screen size and other such variables. However for some Web sites, the cost and difficulty of this method would outweigh the benefits. The cost/benefit ratio of the engineering effort to support all possible platforms must be considered. The best way to

solve this problem is to identify the most common platform and design around it. For a business to consumer site, some common Web design conventions, such as designing for 800*600 resolution in order to prevent the need for horizontal scrolling and avoiding the use of proprietary tags can be useful. Also, designers should always try to avoid dealing with the difference between browsers, only use their common characteristics.

In addition to end-user system requirements, the server-side requirements for the site should be considered. A different type of server could make the Web site design totally different. The server-side requirements will be determined by the functionality demands of the site, corporate standards, and performance requirements.

The functionality of the Web site must be considered when determining what Web server to use. For a dynamic site, the database platforms must be compatible with the server. The server should be chosen with consideration for the number of transactions that will be occurring. Existing business back-ends will also influence decisions such as what database, hardware platform and software should be used. Some business may require a certain platform to ensure compatibility with other company processes. Performance issues will affect the server-side requirements as well. For example, a simple static site would have a totally different server requirement than an interactive retail site with product ordering functionality.

After choosing a specific server according to the above requirements, compatible software, programming languages and technologies can then be decided.

2.3.1.2 Content Requirements

When most users surf the Web, they are looking for information. To them, content is the most important component in a Web site. Every day the Web is becoming a more powerful tool to help these people, as the content becomes stronger and more complete. However, the problem is that as Web content becomes more complete, it is also becoming alarmingly confusing. Every site takes users through a new navigation process

to find what they're looking for. Search engines return results in the thousands. Many times, when a user gets through this maze to the information he/she is looking for, it's out of date or irrelevant. Therefore, a content requirement analysis in the early stages of Web planning becomes extremely important.

The number one concern should be "what to include". Normally, Web site builders tend to simply digitize all the existing information and put it online. However, a Web site is not the same as the business in real world, thus the presentation format should not be the same, either. A site builder needs to analyze the customer demographics. What customers is the Web site trying to attract? Are these customers using the Web? What are they expecting to get from the Web? How useful the content will be for them?

Here are some useful content suggestions for a Web site. Publishing some of this information online can save in customer service costs [3]:

- Detailed information about products or services
- Customer-service information and applications. For example, product manuals, how-to materials, and frequently asked customer questions
- A newsletter that talks about events, sales, or other time-critical information about the business
- Online catalogs, ordering information, and ordering systems
- Archives of publications
- Job openings
- A library of press releases or lists of the partners and clients
- Information for stockholders

After the contents are defined, the format may be an issue. Web site builders need to organize these contents into groups. Research has generally shown that five or fewer options at one time seem to be the best number to present to a user. [27] So always try to get the number of groups to five or fewer. These five groups will form the top levels of the information architecture for the Web site. The main principle for any design is to be

user friendly. The organization of the contents should make it easy for the users to catch the information they are looking for.

Another important thing to do is to establish an ongoing content development process. It makes no sense to spend the time and money to build a Web site and then not maintain it. Many companies devote a lot of time and resources to building Web sites that quickly go out of date and become useless to customers. If a company does not have the time, money, and people assigned to maintain their Web site, this company will very quickly be dead to the World Wide Web. A plan for editing and updating the Web site should be made. Someone in the development team needs to ensure that the Web site is communicating the right message, presenting the up-to-date contents.

On a Web site, the job of maintenance is to keep the site useful, develop the correct content, learn and respond to customer feedback. Consider the following questions about the continuing life cycle of a Web site when figuring out how to maintain it on an ongoing basis [3],

- Who will ensure that outdated content is removed from the site?
- Who will ensure that content from all the different groups in the organization looks and sounds as though it came from one company?
- Who will continually post changes and updates?
- Who will listen to what customers are saying and respond appropriately?
- Who will ensure the site is always fresh with new content?
- Who will add the very important "last updated" dates to every page on the Web site?

An up-to-date, fresh Web site will not only attract customers to the site, but also keep them coming back.

2.3.2 Web Design

After all the content plans are decided, a big part of developing a Web site is how to present these contents. Print media are full of examples of well-presented content. For

instance, most advertising in print magazines and newspapers aims to attract the passing viewer's attention by using a simple concept contained in a short phrase or striking graphic. Most newspaper paragraphs are relatively short to help the reader scan easily. Content presentation is just as important on the Web. In fact, content presentation may be even more important in this new medium, where speed of information access is key for users. There are some basic guidelines for content layout on the Web [27].

2.3.2.1 Text Content

How the text looks like helps determine whether users spend time with the pages or hit the Back button.

People encounter a lot of text in their lives and do not have time to read from beginning to end. Instead, they scan quickly. If they don't find what they want quickly, they quit. So, the text should be structured, and the structure should be visible.

To structure the test: Organize content in short chunks, typically a paragraph or so, with each chunk aimed at helping a user take one step toward a decision. Arrange the chunks in logical order: for example, a sequence in time, or the steps in making a decision. If information presented elsewhere is crucial to understanding a particular chunk, repeat it. Label the chunks clearly so that the user can instantly know what they're about. Highlight the most important points. Use these points of entry into a page to identify, summarize, and promote the surrounding text:

- Images and captions
- Links
- Headlines, subheads
- Tables and listings and forms
- Boxes and sidebars
- Bullet lists, numbered lists
- Very short text

Clear is more important than clever. Make headlines and labels perfectly obvious and make sense.

Try to write Lists: structured collections of information. Table 2.1 compares normal paragraphs and lists,

Paragraphs	Lists
Oral, linear form that works best orally or in print	Visual, non-linear form that works best in interactive media
Hard to scan quickly	Easy to scan quickly
Hard to update, normally have short shelf life	Easy to update, often have long shelf life
Long, unstructured text repels the eye	Short, structured text (bullets, numbers) attracts the eye
Hyperlinks detract from narrative	Hyperlinks make lists more useful

Table 2.1 A Comparison between Normal Paragraphs and Lists

Write tight is also important. How long is too long? It varies by topic and level of reader interest. Pages on the upper levels of a site attract more casual passersby, so they should be shorter and faster. As users go deeper on a topic, they're willing to accept more and denser information on a page. Even so, one rule is that text should be half as long online as it would be in print.

2.3.2.2 Accessibility

The World Wide Web is supposed to be a place where everyone has the ability to find information or shop. The designers need to be mindful of how accessible the Web pages are. For example, if a user's mouse happens to be out of order, the software must be accessible via the keyboard.

So, all clickable elements should be keyboard accessible (for those who cannot use a mouse), all non-textual elements should have a caption (that can be read), 2-D layouts should be able to resize properly. Also consider creating a "text-only" version of the site. This is an important accessibility feature for users who are running a browser that displays only text.

2.3.2.3 Style Guide

- Frames

Avoid frames. Frame slows down the loading and displaying of a Web page, as well as making it more difficult for the user to "bookmark" a specific view and be able to easily return to it.

When frames are necessary, always use three. The first frame should lie across the top of the page and contain the corporate logo and links to the base level pages of the site. Underneath that go the other two frames. On the left should be a section menu to provide links to additional details about this area. On the right should be the main content frame, which should take up at least 75 percent of the content area and display the document the user selected.

- Images: Keep image files as small as possible. Types of files are best formatted as JPEG and GIF.

- Use less animated GIFs: Animated GIFs should be used to draw attention to something, or enforce a message. If all the images are animated on the Web site, it not only confuses the users, but adds unnecessary download time.

- Scripting: Use scripting to add "functionality" to the Web site, not just for the fun of it. Use the appropriate scripting language to meet the user's needs. Be aware that some people might hit the Web site with browsers that do not support scripting of any fashion.

- Font: Avoid requiring special fonts. The user might not have installed the font specified on the Web. If a particular font must be used, provide the appropriate alternate font names that include names for "common" fonts.

- Avoid message boxes: Using the alert function in JavaScript or VBScript to put up a message box is usually annoying to the user. Limit the use of this function to occasions.

- Print: A lot of users love to print out Web pages and read them later at their convenience. This makes it important how the Web page prints out. Conflicts in background color and font color will make all the information on the page totally unreadable when printed on a black and white printer.

- Other browsers and platforms: Not everybody is using the same browser. At least, the Web page should be OK on the latest versions of Microsoft Internet Explorer and Netscape Navigator, using Windows or Mac.

2.3.3 Pictures and Color Management

The time required to download a page is a big problem for users, and should be the major concern for designers. Large graphics files make the downloading slower. Craig Kosak, Art Director for the Microsoft Developer Network points out, "You'll make a big difference in the user's experience and your client's happiness if you make your images download quickly." [3]

But making little tiny images is apparently not a good solution. Users need beautiful pages. What a designer should do is to make those beautiful pages' size as small as possible so that the pages download quickly. Web color management techniques are techniques to do this.

Creating an image that downloads quickly depends on two things: choosing the right file format and working with that image until getting the right balance between download time and image quality.

2.3.3.1 Choose the Right File Format

Currently, two graphics file formats are dominant on the Web, JPEG and GIF.

- JPEG: JPEG (Joint Photographic Expert Group) is a method for compressing color bitmapped images that allows for variable compression. JPEG is also the name of the file format for storing the resulting image. With JPEG, a designer can select compression setting for each image. The marvel of JPEG compression is that the format works with 24-bit images (16.7 million colors) can be significantly smaller than the same image saved in GIF format that works with only 8-bit or less (256 colors) images.

  When the Web content is about pictures, as in a site about fine art or photography, the images need to be as high quality as possible. In this situation, save in JPEG at the Maximum quality setting (lowest compression) for the sharpest and most accurate color. But do not use JPEG for line drawings or for images with large areas of flat color because JPEG compression causes distortion in the flat color areas on machines running 256 colors. Even though JPEG can display over 16 million colors, users with only 8-bit color (most users) will still see the image at only 256 colors at best. This affects the user's experience adversely.

  One confusing behavior of JPEGs is that, even when a JPEG is smaller than a GIF, the JPEG can take longer to load. This is because the JPEG decompresses in the browser. So a smaller JPEG file with more colors might take longer to download and decompress than a bigger GIF file with fewer colors.

- GIF: CompuServe's GIF (Graphics Interchange Format) is the industry standard for Web pages. With GIF, designers can easily move Indexed Color, Gray Scale, or Bitmap images between computer platforms. GIF files support only 8-bit, or 256 colors. For flat color graphics, use GIF.

2.3.3.2 Manipulating GIF Files

- Using an adaptive palette: Adaptive palette GIF may be used if the image contains sharp edges and type, or if it requires transparency. The process of downsizing a GIF file is straightforward in PhotoShop:

1. Change the mode from RGB to Index Color to convert the image to 256 colors.
2. In the Indexed Color dialog box, under Resolution, check 8-bit/PIXEL for the color bit depth. (Experiment from 3-8 bits, depending on the image.) Under Palette, choose Adaptive.
3. Experiment with specifying fewer than 256 colors to reduce file size.

- Using an exact palette: Because using all 256 colors creates a large GIF, consider using an exact palette whenever possible. In some cases designers may be able to abandon the exact palette for an adaptive palette where they specify even fewer colors, but going with an exact palette is usually a safe choice.

2.3.3.3 Using a Browser-Safe Color Palette

If an image uses a color that is not available from one of the color slots in a browser's 256-color palette, the browser will use the closest match that it can find on the palette. In an exact palette, if the colors are way off-scale with the colors in the browser's palette, the browser won't recognize the colors, and will change them in completely unexpected ways. The best approach to avoid this is to select the colors for the image from the colors that the user's browser is prepared to use. There are some browser-safe palettes available for designers. They contain only 216 colors out of the possible 256 colors eliminating the 40 colors that vary on Macs and PCs.

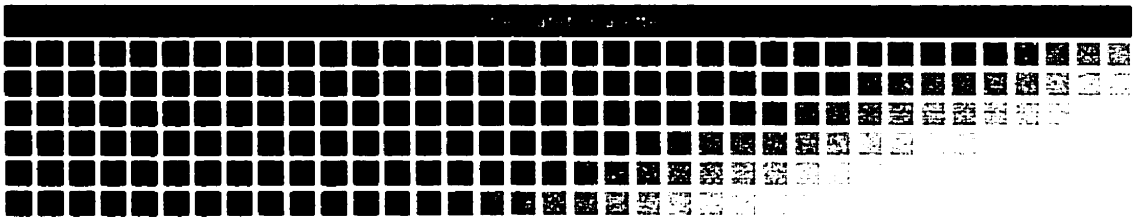

Figure 2.2 Browser-Safe Color Palette

2.3.3.4 Additional Methods for Decreased Download Time

- Reuse common components. The first time an image is used, it is stored in the user's computer cache, and thus can be redisplayed quickly.
- When creating a gradated or textured background, choose gradations or textures that are subtle and continuous in tone.
- Create small files for the background so that the background loads quickly.
- Use specific color schemes. The fewer colors, the better the compression.

2.3.4 Summary

Web planning and design is the first step toward a successful e-commerce solution. The point of this phase is to figure out exactly what should be done. The structure of the site should be well considered, along with how the user will navigate the site. There is no one right way to design things, but there is one right principle to follow: always be user friendly. Users will be attracted to those easy to navigate, easy to catch information, fresh, simple and beautiful sites. Only after all these planning and design are complete should the site be implemented. Implementing too soon usually leads to a poor Web site.

2.4 Database

Many Web sites have functions like search index, yellow pages, portfolio tracking, product support information, catalog ordering and etc. All of these functions require that data be retained and made available based on specific requests. There are a number of

different ways to approach this, but by far the most structured and straightforward one is to use databases. A database is one or more structured sets of persistent data, usually associated with software to update and query the data [25]. A database might be as simple as a single file containing many records, each of which contains the same set of fields where each field is a certain fixed width. More complex database could occupy a whole server and serve several different systems.

Nowadays, the commonly used databases are typically relational databases. Relational databases allow the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields. Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up.

Usually, SQL (Structured Query Language) is used to access relational databases. SQL is an industry-standard language for creating, updating and, querying relational database management systems. SQL was developed by IBM in the 1970s. It is the de facto standard as well as being an ISO (International Organization for Standardization) and ANSI (American National Standards Institute) standard. It is often embedded in general purpose programming languages.

In a Web site, a database is kept in a directory that is part of the site, and accessed via programs like CGI, ASP or JSP. We choose to use two specific databases, DB2 and Access, in Chapter 4, The Solution.

2.5 Web Server

The Internet is a worldwide network of computers that send and receive files and information over a TCP/IP based network. An interwoven system of IP gateways, or routers, directs client requests to their destinations and back. Computers on the Internet are identified by their numeric IP address (for example, 216.55.463.51). Those IP

addresses and each machine's corresponding domain name are managed by DNS (Domain Name Service) servers on the Internet. World Wide Web traffic accounts for much of the traffic on the Internet today. As Figure 2.3 shows, the Web server on the Internet is part of a worldwide network.

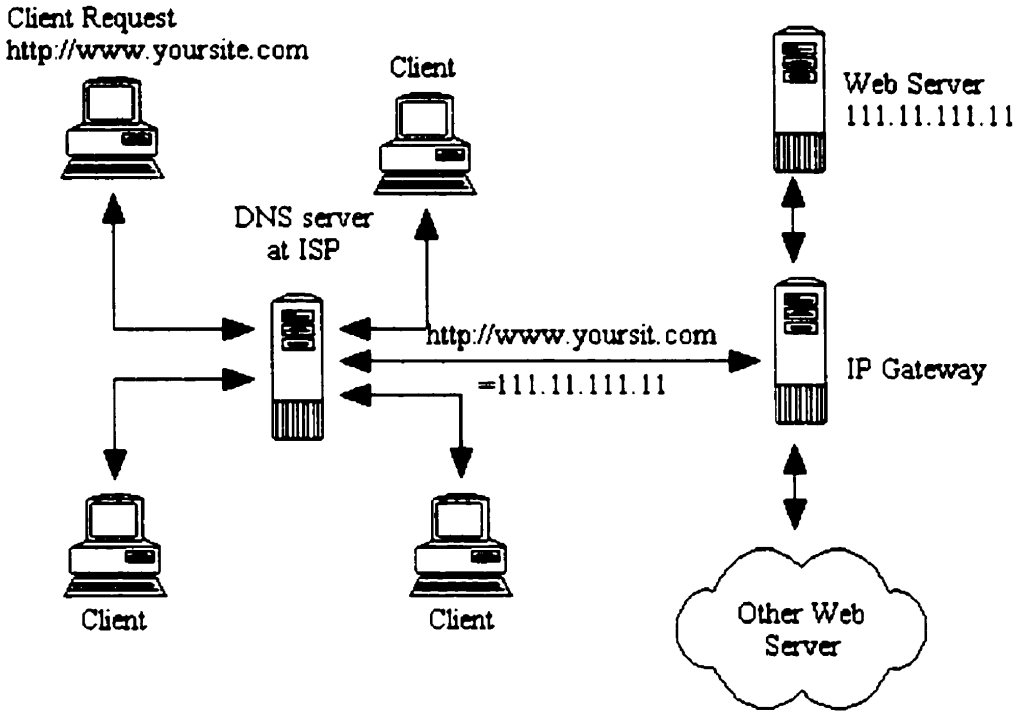

Figure 2.3 Web Server

A Web server is also called an HTTP server. Web servers are connected directly to the Internet via high-speed communication lines. These servers run Web server software that uses HTTP (Hypertext Transfer Protocol, as described in Section 2.2.2) over TCP/IP to send and receive HTTP requests for pages and return the data to clients' browsers. If one site runs more than one server they must use different port numbers. Alternatively, several hostnames may be mapped to the same computer in which case they are known as "virtual servers".

There are all kinds of servers for practically any combination of hardware and operating system. Servers differ mostly in the "server-side" features they offer such as SSL (server

side include), and in their authentication and access control mechanisms. We will build our Web site on IBM HTTP Server and Apache in Chapter four.

2.6 Programming

The heaviest work in Web implementing is programming, which includes client side and server side programming. Client side programming mainly deals with the browser. The program describes how all the contents should be presented on the browser. HTML (Hypertext Markup Language) is the dominant language for client side programming. A large percentage of Web sites contain merely HTML. However, HTML provides no capabilities to design pages that dynamically respond to user inputs. To achieve some simple interactivities, like obtaining user input and sending emails, client side JavaScript can be used. For more complicated functionality, we will have to program on the server side. CGI, ASP, JSP are some of the common ways of server side programming. We have already talked about HTML in Section 2.2.3 and CGI in Section 2.2.4. In this chapter, we mainly focus on JavaScript, JSP and servlet, which will be used in Chapter 4. We will also cover another related server side programming method ASP.

2.6.1 JavaScript

JavaScript is a scripting language that enables the developer to embed programming commands into Web pages, integrate Java applets, Browser plug-ins, server scripts and other Web objects. This allows the developer to create pages capable of high levels of interaction with their users, and access advanced browser capabilities such as multimedia, VRML, layers, and style sheets.

JavaScript was developed by Sun Microsystems and Netscape Communication Corporation as a cross-platform language, that is, a language enabling the same program to run on different types of computers and operating systems. The European standards body ECMA has released a language specification, ECMA-262, derived from JavaScript.

It has been submitted to the International Standards Organization for adoption as an international standard [2].

JavaScript supports the development of both client and server component of a Web site. JavaScript that is interpreted within the reader's browser is known as client side JavaScript. Web pages written in HTML with embedded JavaScript commands are no different from normal HTML pages. They are both downloaded from a server to the user's browser. The browser displays the HTML and executes the JavaScript commands. Server side JavaScript commands are also embedded within the HTML, but they are processed on the server. Server side JavaScript can process information submitted by a Web browser and then update the browser's display accordingly. In other words, the Web pages' responses to actions performed and information provided by their users take place on the server as opposed to the user's computers. Client side and server side JavaScript can be mixed within the same Web page.

2.6.1.1 Client Side JavaScript

Client side JavaScript scripts are included in HTML documents via the <SCRIPT> tag. When a browser loads an HTML document containing scripts, it evaluates the scripts as they are encountered. The scripts may be used to create HTML elements or to define functions, called event handler, that respond to user action, such as mouse clicks and keyboard entries. Scripts may also be used to control plug-ins and Java applets.

Simple JavaScript programs could consist of only a single command. For example, the following script displays a message to the user,

```
<HTML>
<HEAD> <TITLE> Hello world! </TITLE>
</HEAD>
<BODY>
<SCRIPT language = "JavaScript">
document.write ("Hello World!")
```

```
</SCRIPT></BODY>
</HTML>
```

JavaScript can also be complex. They can process information provided by the user in sophisticated ways to add value, and then display the results. They can be triggered by actions performed by users as they read the Web page, for example, clicking buttons, moving the mouse over some specific areas, or selecting from options. Basically, the functions JavaScript can provide are listed below,

- Obtaining information from users: knowing when a Hypertext link is clicked, when a button is clicked or moved over, when a user gets to or leaves a page, using a form, prompting the user for entries, confirming whether something is true, etc.

- Processing the information: Storing and retrieving information over time, text and form processing, doing math, comparing entries, validating user input, etc.

- Communicating the results to the users: updating Web pages "on the fly", opening or closing windows, writing information to forms, loading new documents and images, sending e-mails, etc.

2.6.1.2 Server Side JavaScript

JavaScript can be executed on the server as opposed to the browser. Server side JavaScript can take information from the client, process it to add value, then generate and send HTML, incorporating the results of this processing to the client browser. The fact that this processing has taken place on the server rather than the browser is irrelevant to the user because the display of information is the same.

Server side programming is much powerful than client side. First of all, servers are usually more powerful computers than clients are, although it is not always the case.

Running JavaScript on the server can therefore be appropriate for applications requiring more speed, power, and memory.

The advantages of server side JavaScript is not limited to power, size, and speed. Server side JavaScript can offer functions not available to client side JavaScript. It enables read and write access to files and databases stored on the server as well as calls to C libraries. Client side JavaScript, for security reasons, has no such access.

Client side JavaScript is particularly suitable for smaller tasks. It is ideal for validating and preprocessing user input to HTML forms. These client side processing can often reduce bandwidth when sending the results to the server. Processing and storing information about large numbers of visitors using client side JavaScript can reduce load on the server.

Like client side JavaScript, server side JavaScript is written as text embedded in HTML, except that it uses <SERVER> tag instead of <SCRIPT> tag. The text file is sent to the JavaScript compiler, where the file is compiled into byte code and stored on the server. When a client requests the Web page, the HTML and client side JavaScript are sent to the client and interpreted on the client browser. The server side JavaScript, however, is executed on the server. Commands in the server side JavaScript to write information to the client browser generate HTML and send it to the client browser for display as normal.

Server side JavaScript functions can also be divided into three main groups, obtaining information, processing the information and communicating the results to the client, but with additional features. Listed below are some of these features,

- **Obtaining Information:**

1. Server side objects: Server side JavaScript automatically creates a number of objects that are useful in enabling effective interaction between client and server. For example, when a reader requests a Web page from the server, the server creates a

request object relating to that interaction. The request object stores information as follows:

request.ip: The IP address of the client.

request.agent: The name and version of the client browser.

request.imageX: The horizontal position of the mouse when the client clicked an image map.

request.imageY: The vertical position of the mouse when the client clicked an image map.

request.method: The HTTP method associated with the request (POST, GET or HEAD).

request.protocol: The HTTP protocol level supported by the client software.

request.auth_type: The authorization type.

request.auth - user: The name of the local HTTP user of the browser, if HTTP access authorization is active for the URL.

request.query: Information from the request; material that appears after the question mark.

request.url: URL of the request, minus the protocol, host name, and optional port number.

There are also client Object, project Object and server Object. We won't list all the details in this thesis.

2. Storing client information more permanently
3. Creating other objects and variables
4. Accessing files
5. Accessing databases

- **Processing the information**

  Server side JavaScript can use all the client side adding value techniques. It can also be injected into the process of adding value information retrieved from files and databases accessible by the server.

- **Communicating the results to the users**

1. Writing HTML dynamically
2. Writing JavaScript and variable values
3. Redirecting the user
4. Sending email via server account

2.6.2 ASP

Active Server Page technology is a scripting environment for Microsoft Internet Information Server in which you can combine HTML, scripts and reusable ActiveX server components to create dynamic Web pages.

An Active Server Page is a Web page with an extension of ASP instead of HTM. In addition to the HTML contained in the page, the file contains either JavaScript or VBScript. The script in the file is executed by the server when the page is invoked. Typically, the script will perform some processing and generate HTML statements. The only thing passed to the user is the resulting HTML. There is no visibility into the script code, not even if the user looks at the source code with the browser.

An example of the usefulness of ASP is the processing of a form. Instead of invoking a CGI program, the form data can be provided to an ASP file. That file can then process the data, mail it in a formatted message, and generate an acknowledgment page.

There are a lot of advantages to using ASP. It is easy, not requiring much programming expertise. It is easy to maintain. It is interpreted instead of being compiled, which means only a text editor is needed for debugging. Also, ASP support is built into NT server; no extra effort is needed to run ASP.

The largest drawback to ASP currently is that it is only fully supported on Microsoft NT or Windows series servers. A platform-free alternative to ASP is JSP (Java Server Pages),

which will be discussed in the following section and will be used as our e-commerce solution in Chapter four.

2.6.3 Java Servlet & JSP

Java servlet and JSP are based on platform-independent programming language Java. servlet is Java technology's answer to CGI, so is JSP to ASP.

2.6.3.1 Java

The Java programming language was developed by Sun Microsystems. What makes Java unique is the fact that instead of being compiled into machine-dependent code, it is compiled into intermediate bytecode, which is identical from platform to platform. On each platform is a machine-dependent Java Virtual Machine (Java VM) that can run these bytecodes. Java bytecode help make "write once, run anywhere" possible. A programmer can compile his/her program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run without modification, no matter whether the computer is a Windows 2000, a Solaris workstation, or an iMac.

There are a lot of different types of programs written in the Java programming language, such as applets, applications, servlet, JSP, JavaBeans.

An applet is a program that runs within a Java-enabled browser. Originally applets were used for impressive graphics or sophisticated user input, but they have also been used for complex client programs. A disadvantage of applets is that they usually need to be downloaded from a server which means they must be kept very small to minimize download time.

An application is a standalone program that runs directly on the Java platform.

A servlet, as we will discuss in the next section, is a specialized program that runs on the server side. Similar to the way applets run on a browser and extend the browser's capabilities, servlets run on a Java-enabled Web server and extend the server's capabilities. Java servlets are a popular choice for building interactive Web sites, replacing the use of CGI and ASP.

Java Server Pages (JSP) are HTML documents with imbedded tags and Java code. At runtime, the JSP Page is compiled into a Java servlet and executed. JSP will be further discussed in session 2.6.3.3.

JavaBeans are Java programs that follow the JavaBean specification from Sun Microsystems. They are reusable components that can be called from other Java programs.

How does the API (Application Program Interface) support all these kinds of programs? It does so with packages of software components that provide a wide range of functionality. Every full implementation of the Java platform has the following features [28],

- **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by applets.
- **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.
- **Internationalization**: Help for writing programs that can be localized for users worldwide.
- **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **JavaBeans**: Can plug into existing component architectures.

- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC™)**: Provides uniform access to a wide range of relational databases.

2.6.3.2 Java Servlet

Servlets are programs that run on a Web server and build Web pages. The architecture is shown in Figure 2.4.

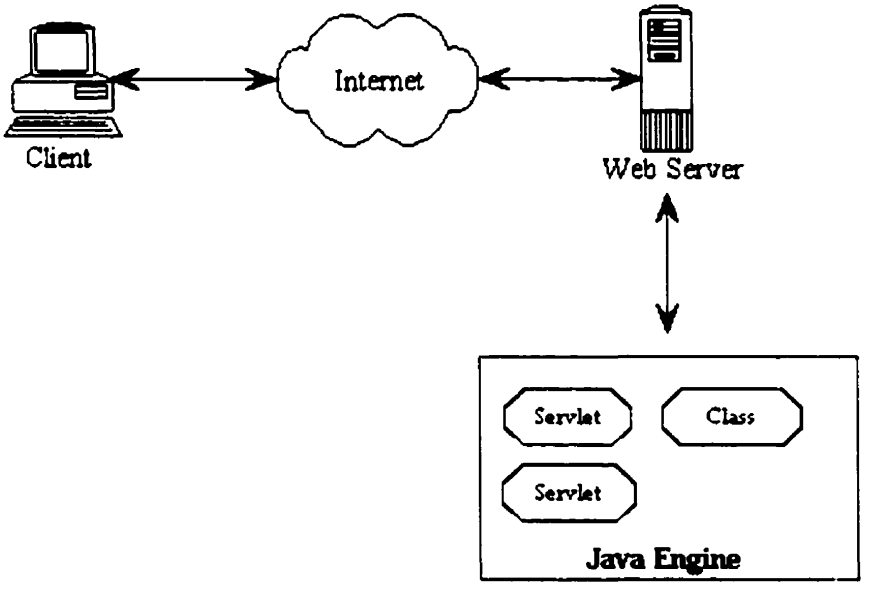

Figure 2.4 Java Servlet

Compared to traditional CGI, Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies.

- **Efficient:** With traditional CGI, a new process is started for each HTTP request. If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time. With servlets, the Java Virtual Machine stays up,

and each request is handled by a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are N simultaneous request to the same CGI program, then the code for the CGI program is loaded into memory N times. With servlets, however, there are N threads but only a single copy of the servlet class. Servlets also have more alternatives than CGI programs for optimizations such as caching previous computations, keeping database connections open, etc.

- **Convenient:** Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.

- **Powerful:** Servlets can talk directly to the Web server. This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.

- **Portable:** Servlets are written in Java. Consequently, servlets written for one server can run virtually unchanged on another. Servlets are supported directly or via a plug-in on almost every major Web server.

- **Inexpensive:** Once there is a Web server, adding servlet support to it (if it doesn't come with the server) is generally free or cheap.

2.6.3.3 JSP

JSP (Java Server Page) allows Web developers to mix regular, static HTML with Java technology. Here is a sample JSP program,

<HTML>

```
<HEAD><TITLE> Hello World! </TITLE> </HEAD>
<BODY>
<H1> Hello,
<!-- User name is "New User" for first-time visitors -->
<% out.println(Utils.getUserNameFromCookie(request)); %>
</H1>
</BODY>
</HTML>
```

As we can see in the above example, JSP separates the user interface from content generation. This enables designers to change the overall page layout without altering the underlying dynamic content.

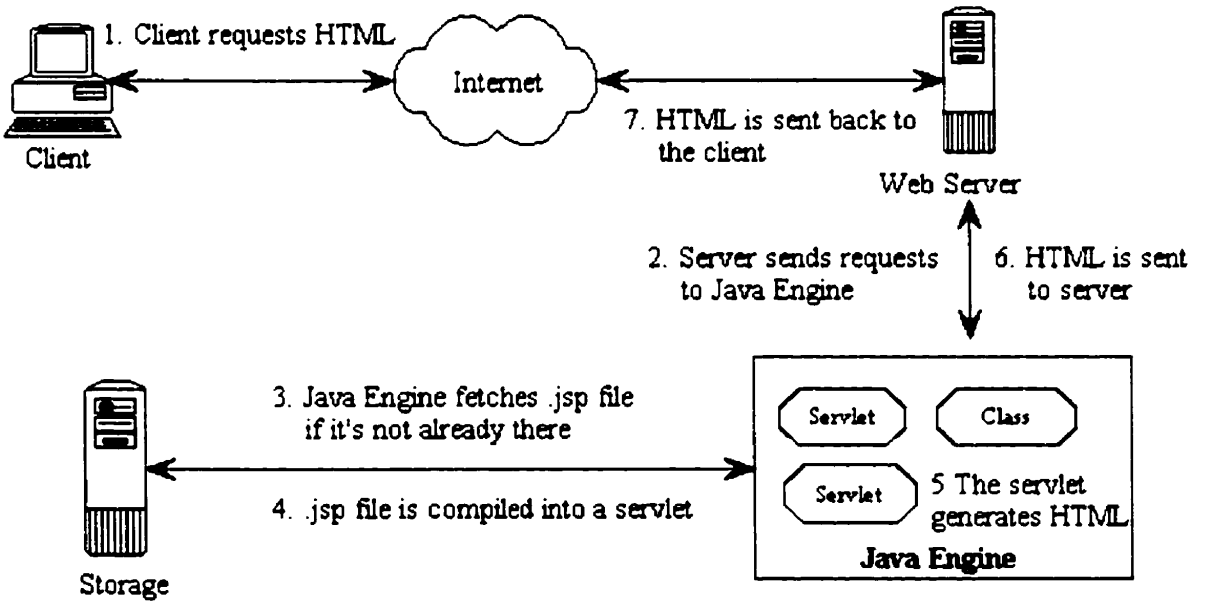

Figure 2.5 Java Server Pages

As part of the Java family, JSP technology enables rapid development of Web-based applications that are platform independent. JSP is an extension of the Java servlet technology. Together, JSP technology and servlets provide an attractive alternative to other types of dynamic Web programming that offers platform independence, enhanced

performance, separation of logic from display, ease of administration, extensibility into the enterprise and most importantly, ease of use. Figure 2.5 shows how JSP and Java servlets work together with Web servers.

Sun has made the JSP specification freely available to the public, so that every Web server and application server will support the JSP interface. JSP pages share the "write once, run anywhere" characteristics of Java technology. It also has a lot of advantages over other Web developing methods,

- **JSP vs. ASP:**
  Firstly, the dynamic part is written in Java, not Visual Basic or other MS-specific language, so it is more powerful and easier to use. Secondly, it is portable to other operating systems and non-Microsoft Web servers.
- **JSP vs. Pure Servlets:**
  JSP cannot do anything that couldn't be done with a servlet. But it is more convenient to write and to modify regular HTML than to have a whole bunch of statements that generate the HTML. Plus, by separating the look from the content, different people can work on different tasks: the Web page designers can build the HTML, leaving places for the servlet programmers to insert the dynamic content.
- **JSP vs. Server-Side Includes (SSI):**
  SSI is a widely supported technology for including externally defined pieces into a static Web page. It only supports simple inclusions. JSP is better because it use servlets instead of a separate program to generate that dynamic part, and it supports more complicated programs that use form data, make database connections, etc.
- **JSP vs. Static HTML:**
  It is obvious that regular HTML cannot contain dynamic information. JSP can be much powerful by only insertion of a small amount of dynamic program.

Chapter 3

WebSphere

WebSphere is an e-commerce software platform developed by IBM. It links together all the applications including fulfillment, logistics, distribution, and accounting processes. It helps businesses build, manage and grow e-commerce sites. The WebSphere software platform for business offers specific solutions for the B2B and B2C markets. For B2C sites, WebSphere has solutions that include catalog and storefront creation, merchandising, relationship marketing and payment processing. For B2B sites, the WebSphere software platform includes sell-side applications to help them sell goods and services to other businesses, exchange key business documents with partners and vendors, and address key areas of content management, such as profiling and personalization. Because of the focal point of this thesis is B2C sites, we will only talk about WebSphere B2C part.

B2C e-commerce is more than putting up a storefront. In addition to catalog and storefront creation, the WebSphere software platform helps ensure a successful e-commerce solution by making the most of payment processing and electronic and marketing techniques that attract and retain the customers. The WebSphere product family consists of multiple integrated applications to assist with building, running and managing complex Web sites. Figure 3.1 shows an overview of the WebSphere product family. We can assort all the tools into three catalogs: build, run and manage. [35]

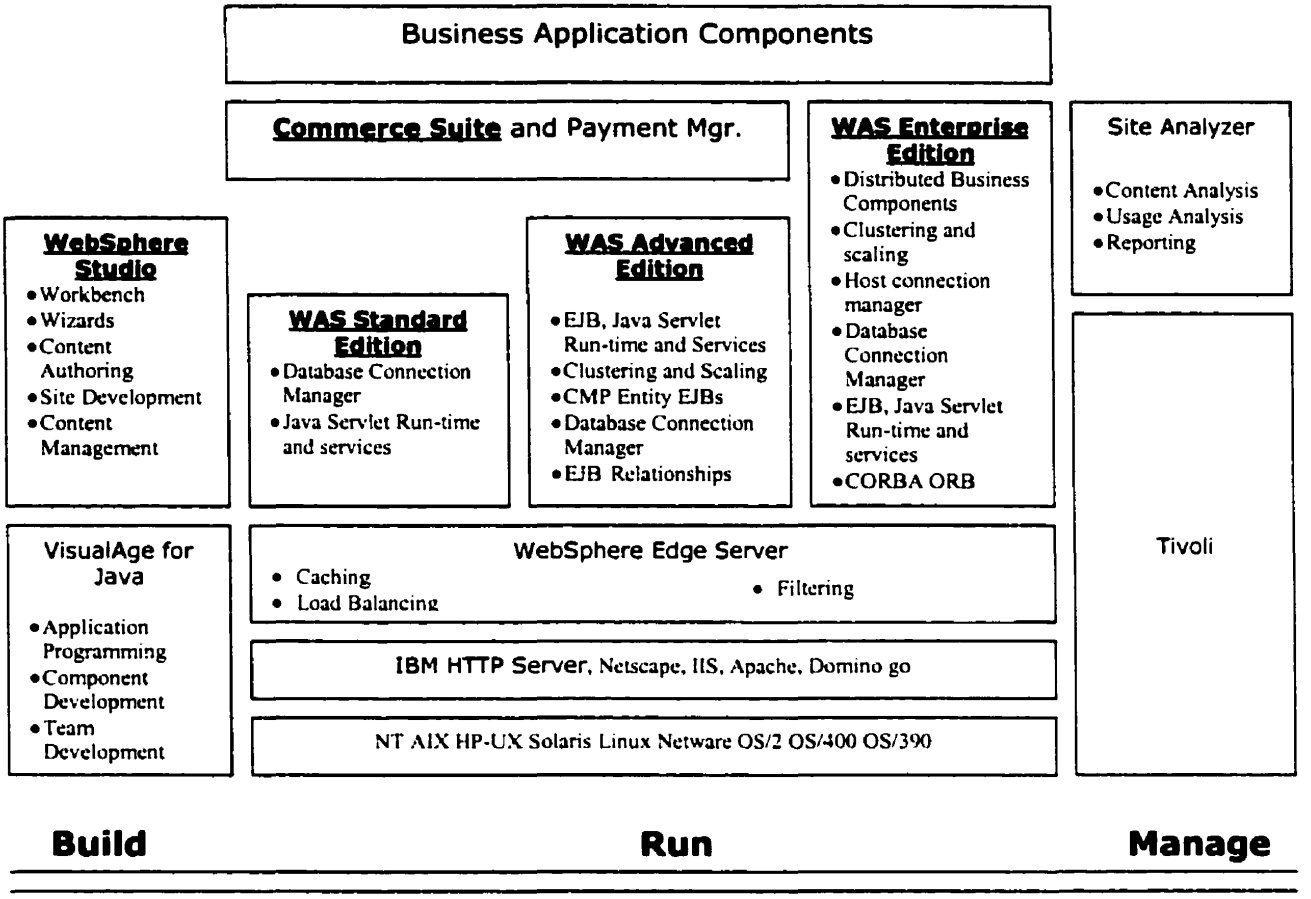

Figure 3.1 WebSphere Product Family

- **Build**

Building tools include WebSphere Studio and VisualAge for Java. WebSphere Studio provides a project-based environment for managing complex Web sites consisting of static and dynamic content. VisualAge for Java provides a single-user or team environment for developing Java applications, servlets, Java Beans and Enterprise Java Beans (EJBs).

- **Run**

Applications for running Web sites include several kinds of servers. WebSphere Application Server provides the infrastructure for deploying enterprise Web applications consisting of servlets, Java Server Pages and Enterprise JavaBeans with connections to

databases and transactional systems. WebSphere Edge Server provides network dispatcher technology and a caching proxy server. IBM HTTP Server combines the popular Apache Web server with performance, security and manageability enhancements. WebSphere Commerce Suite and WebSphere Payment Manager provide the e-commerce business functionality and tools to create and manage standalone or hosted merchant sites.

- **Manage**

WebSphere Site Analyzer provides tools for analyzing Web site content and usage. WebSphere Commerce Suite is Tivoli Ready™, which means Tivoli solutions can be used to manage the e-commerce solution. Tivoli's e-business management products enable businesses to speed deployment, ensure security, maintain availability and optimize performance of the business systems.

In the following sections, we will discuss three essential components of WebSphere: WebSphere Studio, WebSphere Application Server and WebSphere Commerce Suite. They have been highlighted in Figure 3.1. These components are necessary in all WebSphere projects, and are most relevant to our solutions in Chapter four.

3.1 WebSphere Studio

WebSphere Studio is a suite of tools that help teams design, develop, and publish Web applications. WebSphere Studio is the first tool in the industry for visual layout of dynamic Web pages. [39] Studio supports JSPs, full HTML, JavaScript and DHTML, uses wizards (for generating database-driven pages), and updates and corrects links automatically when content changes. WebSphere Studio allows developers to integrate their favorite content creation tools, and provides local and remote debugging with the industry's first JSP debugger. Here listed some of the Studio features:

- Page designers, graphic artists, and programmers all work on the same projects with a common view

- The Web site is composed of independent parts, so it is easier to maintain
- Any tools can plug in, so developers can work with their favorites
- Web development tools include:
  - Page Designer to visually create and edit HTML and JSP files
  - Applet Designer, a visual authoring tool that makes it easy to build new Java applets
  - WebArt Designer for creating images, buttons, and other graphics for the Web page
  - AnimatedGif Designer for assembling GIF animations
- Wizards are provided to import a site into Studio and to create server-side Java programming logic for database access or JavaBeans
- Includes an integrated Debugger to find JSP and Java code problems in real time
- Integration with popular source control management software like IBM VisualAge Team Connection, Rational ClearCase, Microsoft Source Safe, PVCS, and Lotus Domino

WebSphere Studio version 3.0 is fully integrated with IBM VisualAge for Java. They work together to reduce the time and effort in creating, managing, debugging and deploying Web applications. These tools are not only integrated with each other, but also with the new versions of the IBM WebSphere Applications Servers for creating and deploying EJB applications that developers can test and debug fully. Tasks can be done either from within the VisualAge for Java integrated development environment (IDE) or remotely on WebSphere.

A major goal of WebSphere Studio is to make it easy for developers to organize and manage Web development and deployment so they can save time and avoid errors. The Studio has Java servlet and JSP generation wizards for easy creation of dynamic Web pages using Java technology. The Studio workbench provides easy, powerful team-enabled site creation and management. Web page designers, graphic artists, and programmers can all work on the same project with common views of files and relationships.

3.2 WebSphere Application Server

WebSphere Application Server provides the "platform" for Web applications based on Java technologies such as servlets, Java Server Pages and Enterprise JavaBeans. It is the basis of WebSphere application services. Figure 3.2 shows a number of possible relationships among the elements of a Web application. [36]

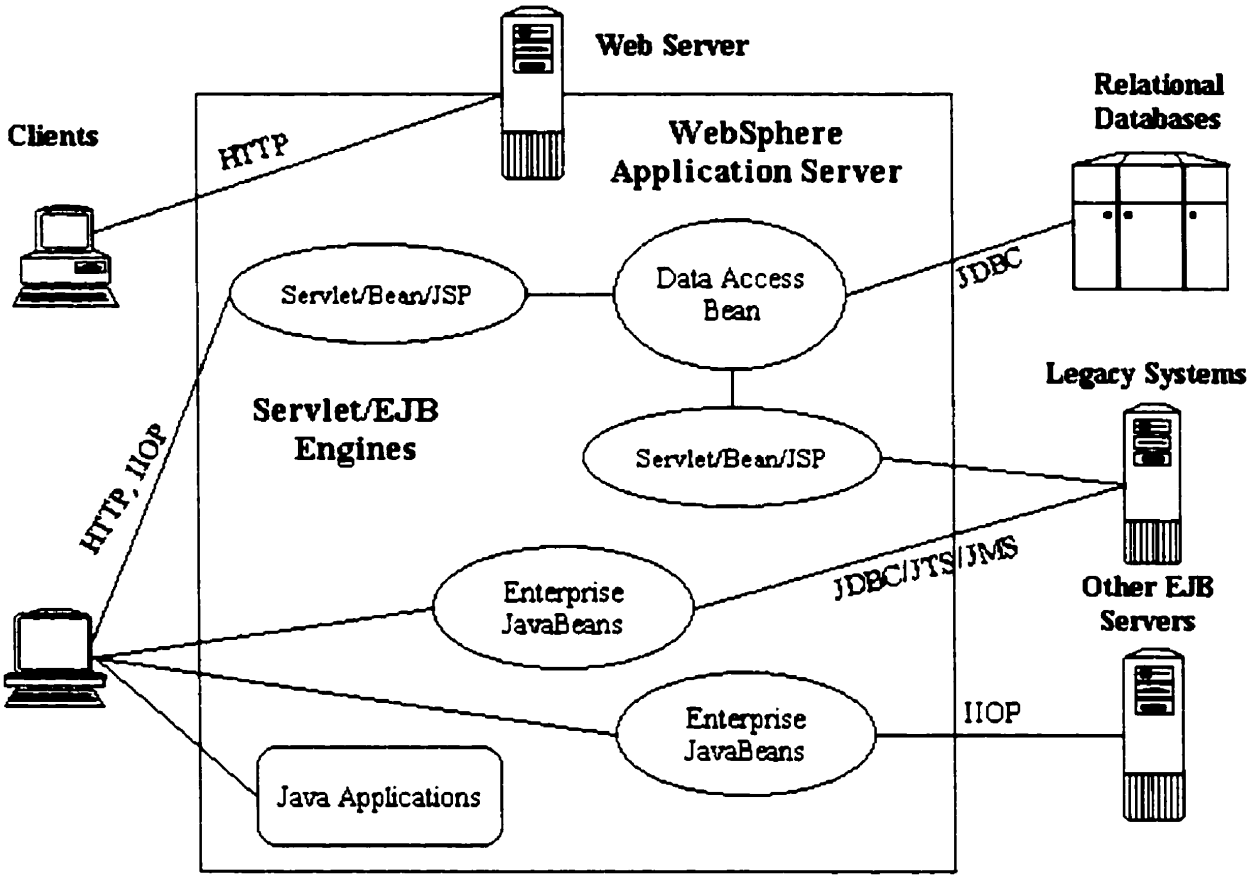

Figure 3.2 WebSphere Application Server

WebSphere Application Server Standard Edition provides support for running non-transactional server side Java applications based on Java servlets and Java Server Pages, incorporating database access, personalization and security. The Standard Edition lets developers use Java servlets, Java Server Pages and XML to quickly transform static

Web sites into vital sources of dynamic Web content. Standard Edition is appropriate for a single node site. Its features include,

- Support for Web applications:
  - Java servlets 2.1 runtime
  - Java Server Pages 0.91 and 1.0 runtime
  - JDBC database access with connection pooling
  - JNDI naming services
  - XML support
  - Integrated security using LDAP
  - HTTP Session and User Profile Support
- Administration tools
  - Java based graphical administration interface
  - Distributed debugging support
- Works with all major Web servers
  - IBM HTTP Server, IIS, Netscape, Apache, Domino

WebSphere Application Server Advanced Edition extends the capabilities of Standard Edition by providing Enterprise JavaBeans support, transaction services and clustering support. The Advanced Edition is a high-performance EJB server for implementing EJB components that incorporate business logic. It has all features of Standard Edition plus,

- Scalable, highly-available transactional runtime
  - Clustering with multiple replicated nodes
  - Workload management
  - Built-in Object Request Broker
  - Supports DB2 transactional JDBC driver
- Enterprise JavaBeans support
  - Full EJB support, including container managed persistence, entity, and session beans
  - EJB security
  - Integrates with DB2, Oracle, and Sybase databases

WebSphere Application Server Enterprise Edition provides support for building distributed, transactional applications that can integrate Web and non-Web systems. The Enterprise Edition integrates EJB and CORBA components to build high-transaction, high-volume e-business applications. Enterprise Edition provides the features of Advanced Edition plus some powerful product components like Component Broker, TXSeries and IBM MQSeries. According to our project requirements, we will be using the Advanced Edition in Chapter four.

3.3 WebSphere Commerce Suite

WebSphere Commerce Suite is actually a new version of Net.commerce. It provides all of the tools and services needed to build scaleable, high functional e-commerce Web sites for businesses. The package also includes a Web server, DB2 database and Payment Server. The WebSphere Commerce Suite functionality includes an extensible server providing the store flow business logic, a comprehensive data model for merchant data, an administration interface including store creation tools, catalog architect for defining product catalog data, Product Advisor for producing intelligent catalog views, some sample stores and backend integration samples.

WebSphere Commerce Suite Start Edition helps businesses develop compelling e-commerce sites that allow customers to set preferences, navigate catalogs, drop items in shopping carts, check out, select payment and shipping methods and pay for purchases by credit card. An administrative interface is provided for managing Web site, stores or malls. Developers can create interactive catalogs that can help sell anything (including hard goods, soft goods and services) and guide customers through efficient shopping flows. Credit card can be processed with IBM WebSphere Payment Manager.

WebSphere Commerce Suite Pro Edition can increase site functionality, accommodate high transaction volumes and leverage the back-end systems. WebSphere Commerce Suite Pro Edition contains all the features and functionality of the Start Edition plus these features:

- Integrate existing middleware and systems using accepted industry standards like Java technology, Java Server Pages, Enterprise JavaBeans and XML.
- Create effective up-sell and cross-sell techniques that drive sales by presenting products complementary to initial purchases or that invite customers to purchase more expensive items than the original selection in the same product category.
- Offer time-based promotions that allow a business to move more products in a specified time or inventory condition.
- Reach global customers by providing information based on preferred language and cultural preference.
- Serve the customers, regardless of location, time or device, by extending the applications to wireless users through m-commerce capabilities.
- Build associations (such as accessories or replacements for out-of-stock items) to increase revenue; create one-price packages at attractive prices; assemble bundles that support convenience and offer guidance to the customers.
- Customize catalogs that cater to specific buying preferences and purchasing patterns through the product advisor feature
- Conduct open-cry, sealed-bid and Dutch auctions, providing a new channel to move excess inventory and discontinued products. Set the rules of the auctions and establish an auction gallery for buyers to view, search and specify bids for items on the auction block. The end result is not only a win for the highest bidder but also a win for the e-business.

3.4 Summary

All successful e-commerce must navigate a set of four phases: quick start, growth and maturation, differentiation, and continuous integration. From a technical perspective, businesses need a platform that supports a wide range of solutions, features, and change management. With all the features discussed in previous sessions, WebSphere software platform stands out among all the similar e-commerce software packages. It has emerged as a flexible and robust platform for strategic e-commerce. Instead of forcing

organizations to assemble pieces, it provides a comprehensive platform. As all good e-commerce platforms should, WebSphere supports key industry standards, including J2EE, XML, LDAP, and WAP.

WebSphere provides Studio and VisualAge development tools for quick start application. It offers a variety of scalability and reliability features required for all e-commerce solutions. WebSphere also includes enhanced and advanced features for those businesses trying to differentiate themselves.

Chapter 4

The Solution

The Web site we are going to set up is named "The Buccaneer". The Buccaneer is a Web-based discount store. Unlike businesses that use the Web as part of a sales and marketing strategy, The Buccaneer will use the Web for its business in general. The Buccaneer mainly focuses on discount car accessories, but it has a potential to extend its products to other categories. As described in the previous chapters, the major phases in a Web site development cycle would be requirements gathering, design and implementation. In this chapter, we will go thought these steps and present two different implementations.

4.1 Requirements

The requirements gathering phase is one of the most significant phases in development. The design is based on the requirements and the construction is based on the design. So the requirement specification is the foundation of the entire project. After an intense communication with The Buccaneer's managers, a close investigation into the project and a comprehensive survey on similar Web sites, we concluded the requirements as listed below,

1. Introduction to The Buccaneer as well as information on using the site.

2. The presentation of an initial list of categories, providing the ability to navigate down through subsequent levels of subcategory.
3. The ability to navigate from subcategories up towards the initial list.
4. A shopping cart for users to check their selections.
5. A customer online registration process.
6. Online shopping abilities such as customer login, price calculation, credit card processing.
7. A wholesale page with password protect, where authorized wholesale buyers could access for wholesale prices and information.
8. An ODBC database on the server to store the customer information and transaction records.
9. A help page provides information on how to shopping online, means of delivery and any other useful guides.
10. An overall design to make The Buccaneer's image consistent.
11. The description of the copyright ownership for the site.
12. The description of both marketing and technical contact information.
13. The development platform should be based on existing hardware conditions: PC Pentium III 500M.
14. A message showing that the Web site is up-to-date.
15. Fast downloading time.
16. A server capacity of about 50 to 1000 visits a day.
17. An assumption that the uses have low to average level of computer knowledge, average speed connection to the Internet, a 15 inch monitor and a resolution of 800*600 or higher.

4.2 High- Level Design

Based on the requirements listed above, we came up with a high level functional design as shown in Figure 4.1.

The "home" page is the first page users would surf and the most important page in a Web site in terms of capturing users' attention. The Home page aimed at browsers should be analogous to magazine covers. The home page is the focal point for the rest of the site. All the links on the home page should point inward, toward pages within the site. A very clear and concise statement of what is in the site that might interest the user will be provided. Menus will be organized in a clear way presenting all six possible choices as drawn in Figure 4.1. Current date information and a description of the copyright ownership will be on the homepage as well.

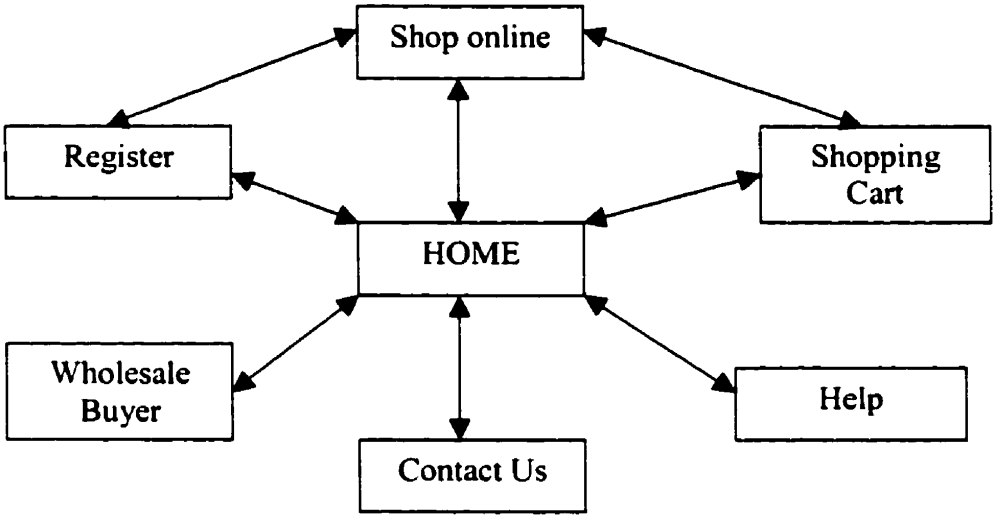

Figure 4.1 The Buccaneer High Level Design

The "register" page will be a form to gather user's personal information. The users will be asked to choose a user name, a password and provide information such as name, address and phone number. The users need to fill out this form before they buy anything online. With the user name and password they chose, they do not need to provide these information when they shopping again. This page can be accessed at anytime from the menu, and it will be launched automatically when users check out from their shopping cart.

The "shop online" page actually includes a series of category pages. All the available product categories will be organized in a hierarchical style. Navigation buttons will be placed on every page for easy access.

The "Shopping cart" page will give users a list of products they chose to buy. They can check their shopping cart at any time during their surf. No registration is needed for accessing the cart. This page also leads to real transaction pages when users click "check out" button. Transaction pages include "login" or "registration", verification of the item list and payment methods, and a confirmation of the transaction result page. In other scenarios, if users click "continue shopping" button, the "shopping cart" page will guide them back to "shop online" page.

For wholesale buyers, the "wholesale" page serves as an information board to list all the up-to-date wholesale products and prices. There is no online transaction function in this page because most wholesales are not paid by credit cards. This page is password protected. Only authorized users can access this page.

The "Contact Us" page provides information on how to contact The Buccaneer's sales and Webmaster. The users can use this information should they have any question about The Buccaneer.

The "Help" page will provide information on how to shop online, how to calculate the price, how to order offline, the delivery regulations, security concerns, FAQ (frequently asked questions) and etc. The objective is to make sure that the users never feel confused and helpless when they shop at The Buccaneer.

To summarize this section, we derived a high-level design from the requirements analysis highlighting the functions that will be available to users. We will skip the layout and art design parts since they are not our main focuses in this thesis. Low-level design and different implementations will be presented in the following sections.

4.3 Building from Scratch

As mentioned in Chapter one, this approach enables the developers to shape the system to the exact specifications. It requires more technical know-how but allows far greater flexibility in the development.

4.3.1 Development Platform

There are many possible ways to set up a Web site from scratch. A different choice of hardware or programming language could lead to the same storefront, however, the cost, performance and maintenance would be very different. Our objective is to find a solution with low cost and high performance.

As described in Chapter two, Java technology (JSP, servlet) has more advantages in Web development than any other possibilities. Therefore, we decided to use Java as the main programming language in our solution. Based on the consideration of cost, performance and compatibility, we require the following development environment,

- Hardware: Pentium III PC
- Operating System: Window 98 (Basic)
- Web Server: Apache (Free)
- Java Runtime Environment: Java Development Kit 1.3 (Free)
- Java Engine: Allaire JRun (Free)
- Java Editor: Allaire JRun Studio (Free)
- HTML editor: Note Pad (Included in Windows98)
- Database: Microsoft Access (Basic)
- Picture editor: Adobe Photo Shop (Basic)
- Browser: Internet Explorer (Free)

As we can see, these software are either free or basic, which means most probably they are pre-installed on the hardware. This development platform ensures a very low start-up

cost. Besides, all the software are well-developed, mature products. We can assume that they are highly reliable.

4.3.2 Application Architecture

Now it is time to break down the high-level design in Section 4.2. A detailed chart of the application architecture is shown in Figure 4.2. In this chart, each page on the Web site and the navigation that leads to the given page are identified. Each box represents a Web page. The bold texts are page titles. The italic texts are the names of the files that support the pages. There are lines connecting pages. The arrowheads on the lines indicate flow direction. A solid line means that the movement is triggered by the users, typically by clicking a hypertext link or submitting a form. A broken line means that the movement to the destination page is a function of the Web site not controlled by the users, which could be a request forwarding or a file being loaded.

The top seven boxes are links on the menu bar that will be on top of every page. Customers can go to these pages at any time, from any page in the Web site. Therefore, every page should have a link to these boxes. However, there would be no point to draw all these trivial details in the chart and so we did not do that. Typically, a Web page contains one file. However, the "shopping cart" page is relatively more complicated and needs the support from several servlets. The database connections are not shown in this chart, but will be discussed later in this chapter. It is worth mentioning that because of the complicated logic in our programs, we decided to use servlets instead of JSP.

Before we begin the real implementation, we need to make sure that the design we have been talking about so far fully supports all the requirement specifications. Here is the list of requirements. In the bracket following each requirement are the supporting Web pages or some specific comments

1. Introduction to The Buccaneer as well as information on using the site. (Home, Help)

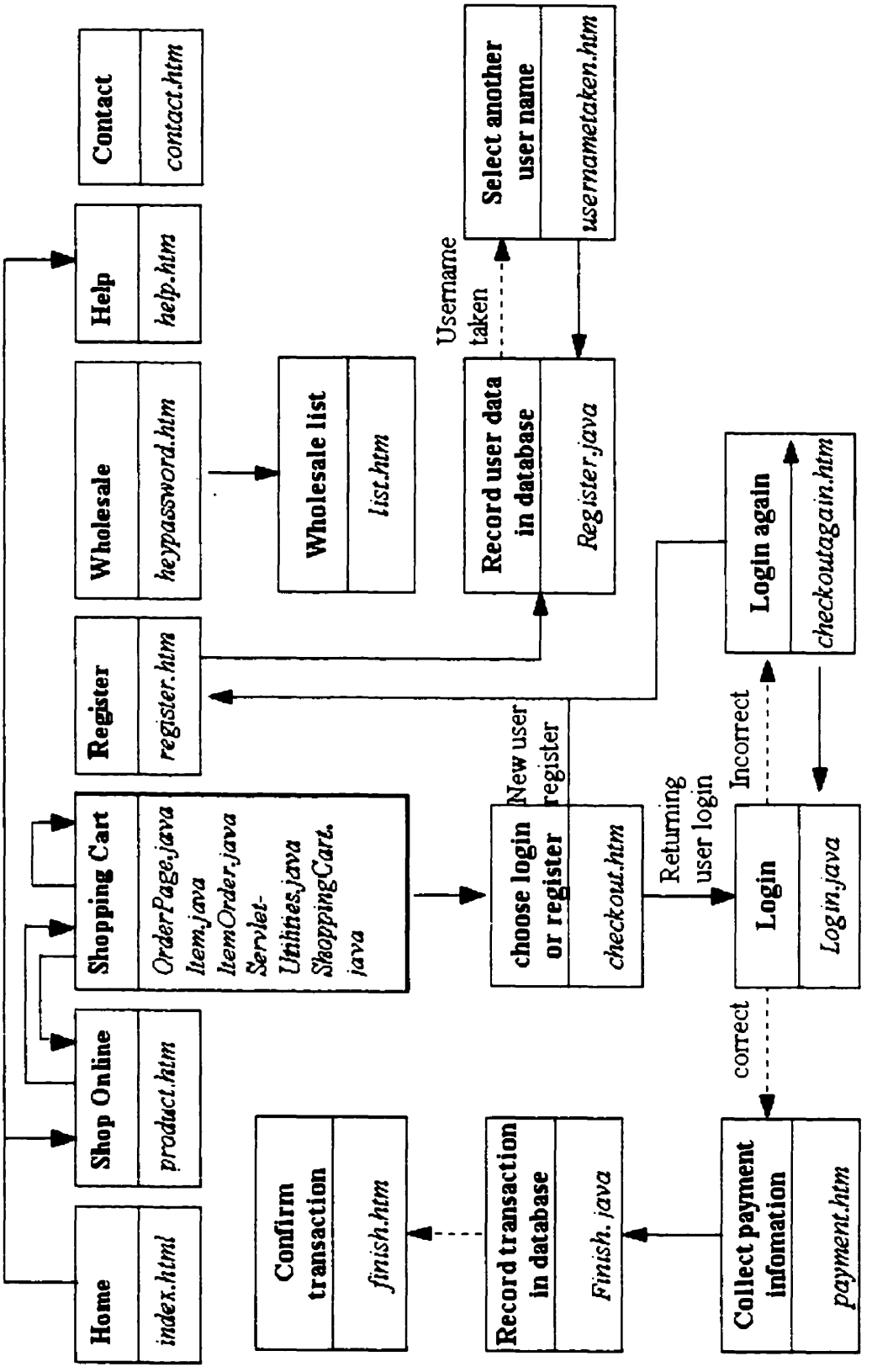

Figure 4.2 Application Architecture

2. The presentation of an initial list of categories, providing the ability to navigate down through subsequent levels of subcategory. (Shop Online)
3. The ability to navigate from subcategories up towards the initial list. (Shop Online)
4. A shopping cart for users to check their selections. (Shopping Cart)
5. A customer online registration process. (Register, Record user data in database, Select another user name)
6. Online shopping abilities such as customer login, price calculation, credit card processing. (Choose login or register, Login, Login again, Collect payment information, Record transaction in database, confirm transaction)
7. A wholesale page with password protect, where authorized wholesale buyers could access for wholesale prices and information. (Wholesale, Wholesale list)
8. An ODBC database on the server to store the customer information and transaction records. (Record user data in database, Login, Record transaction in database)
9. A help page provides information on how to shopping online, means of delivery and any other useful guides. (Help)
10. An overall design to make The Buccaneer's image consistent. (All pages)
11. The description of the copyright ownership for the site. (Home)
12. The description of both marketing and technical contact information. (Contact)
13. The development platform should be based on existing hardware conditions: PC Pentium III 500M. (Specified in the Development Platform)
14. A message showing that the Web site is up-to-date. (Home)
15. Fast downloading time. (All pages)
16. A server capacity of about 50 to 1000 visits a day. (Specified in the Development Platform)
17. An assumption that the uses have low to average level of computer knowledge, average speed connection to the Internet, a 15 inch monitor and a resolution of 800*600 or higher. (All pages)

4.3.3 Implementation

In this section, we will briefly go through the implementation of some important pages in the Web site and the structure of database tables.

4.3.3.1 Home, Help Page, Contact Page and Wholesale Page

The reason we talk about these four pages together is that they are all static pages. They have no database requirement and no Java programming. We programmed these pages in Notepad with HTML and some JavaScript. The JavaScript is used to present the current date on every page and prompt customers for password in the Wholesale page. These pages all look similar, except the texts and some pictures are different. Figure 4.3 to Figure 4.7 show home page, help page, contact page, wholesale login page and wholesale page, respectively.

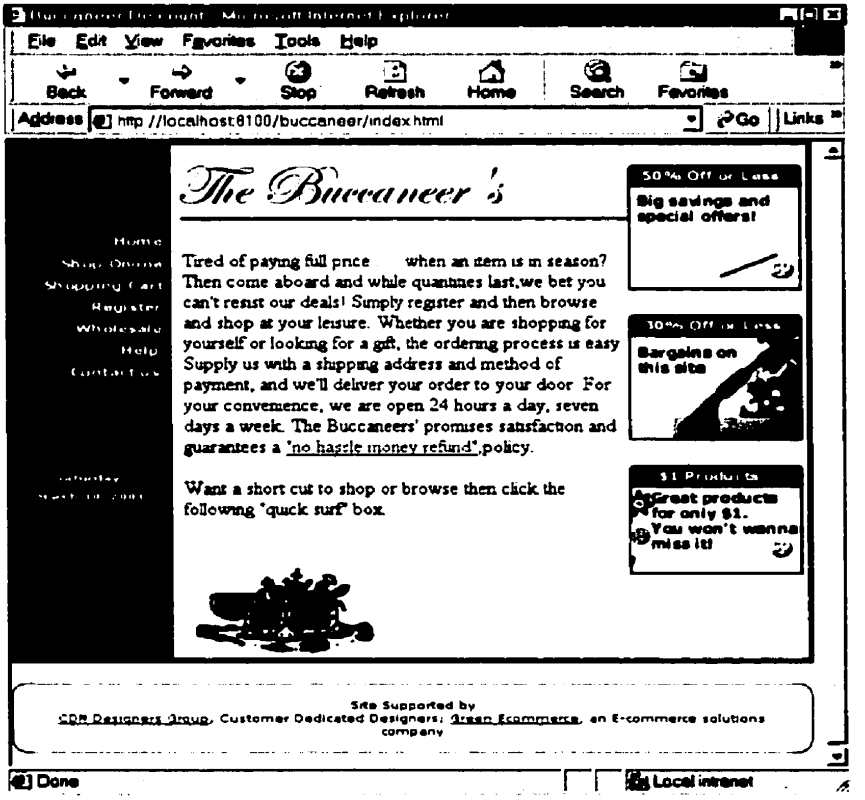

Figure 4.3 Home Page

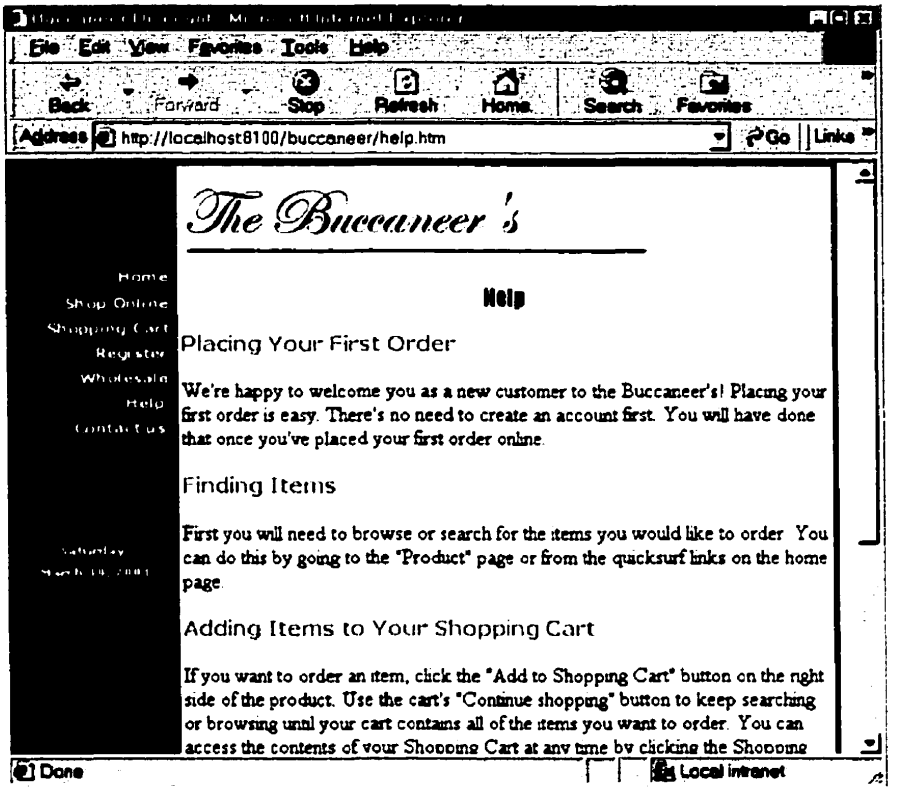

Figure 4.4 Help Page

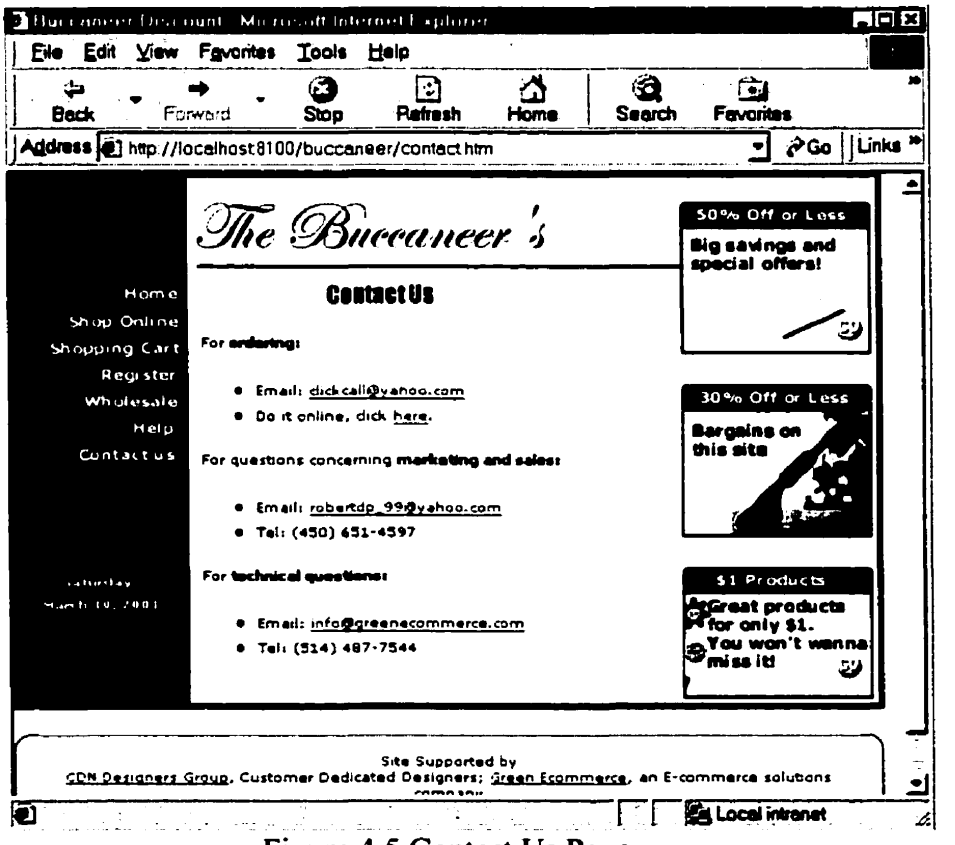

Figure 4.5 Contact Us Page

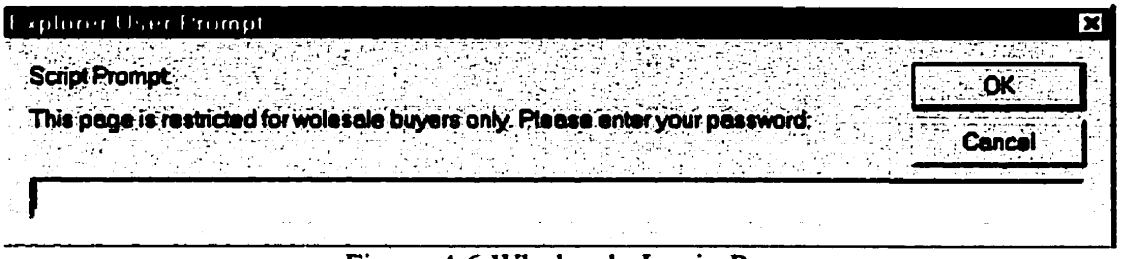

Figure 4.6 Wholesale Login Page

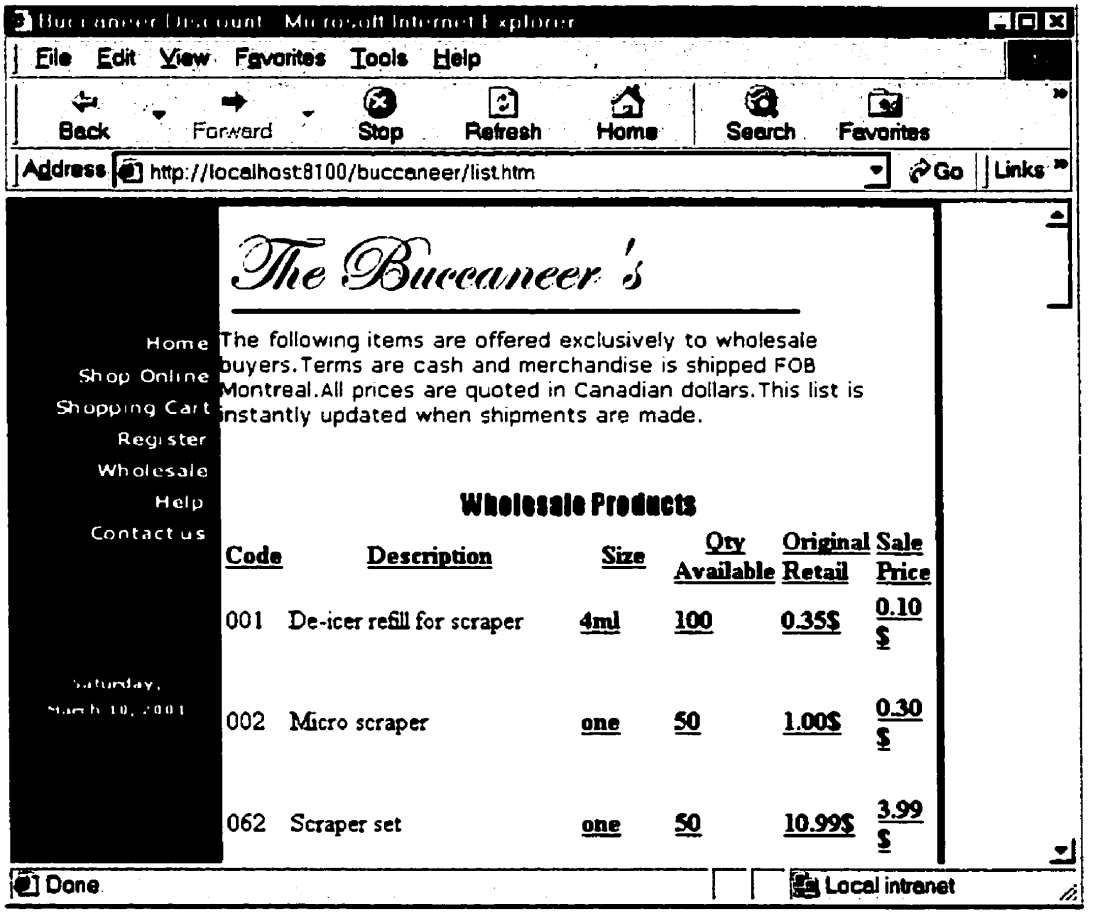

Figure 4.7 Wholesale Page

4.3.3.2 Shop Online

Shop Online page shows customers the product lists. There is a category table at the upper part of the page. Customers can navigate down from these top categories to real

product lists. There are also "Back to top" buttons below every list enabling customers to navigate back to the top categories.

For each product, a product ID, a short description, a regular price and a buccaneer's price are listed. There is an active link of "Add to cart" for each product. Once an "add to cart" link is invoked, a query is sent to the shopping cart programs. For example, for a product named "De-icer buster combo" which costs $4.99 and has a product ID of 150, the query would be href="../servlet/OrderPage?itemID=150&shortDescription=De-icer+ buster+combo& cost=4.99". Figure 4.8 shows the top part of the shop online page.

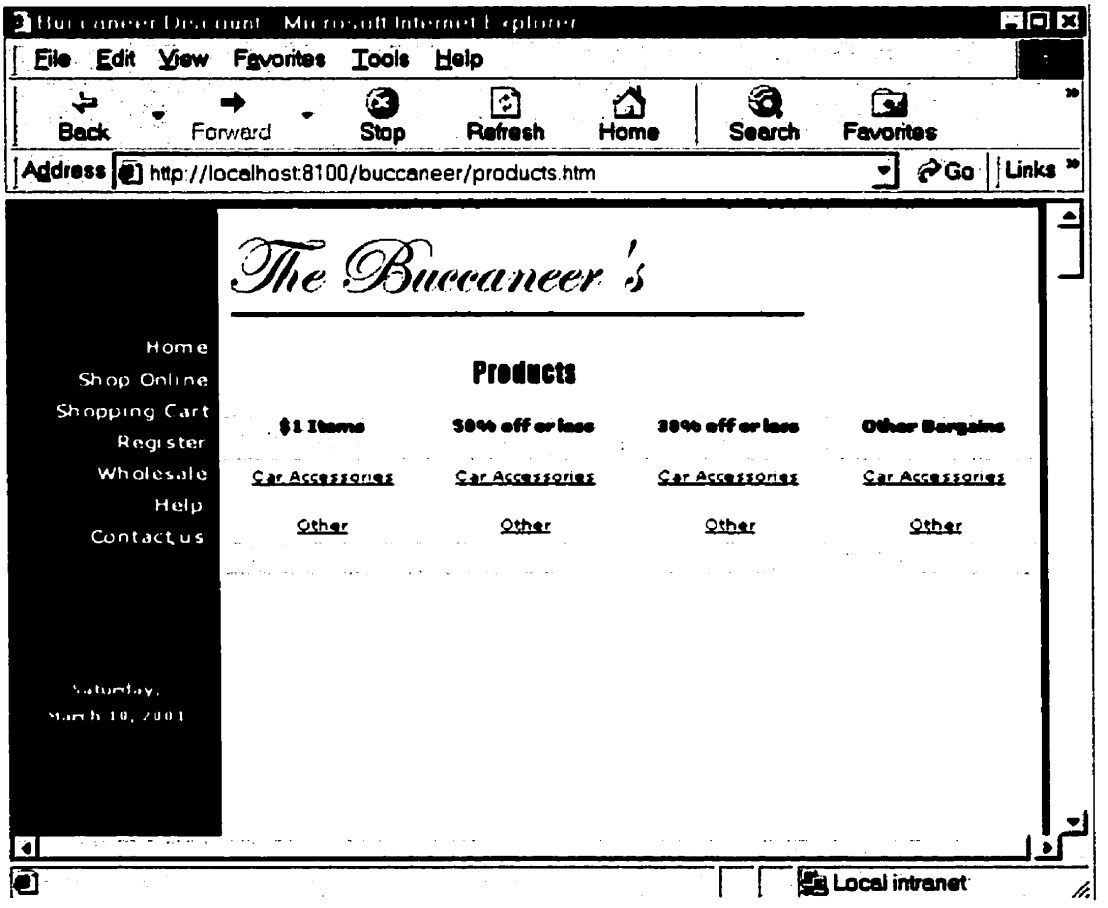

Figure 4.8 Shop Online Page

4.3.3.3 Shopping Cart

The shopping cart is one of the most important parts in this Web site. A shopping cart allows customers to select products, keep track of their previous selections and make changes to their interest lists. A shopping cart can also automatically calculate prices and distinguish different customers. Figure 4.9 is an sample shopping cart.

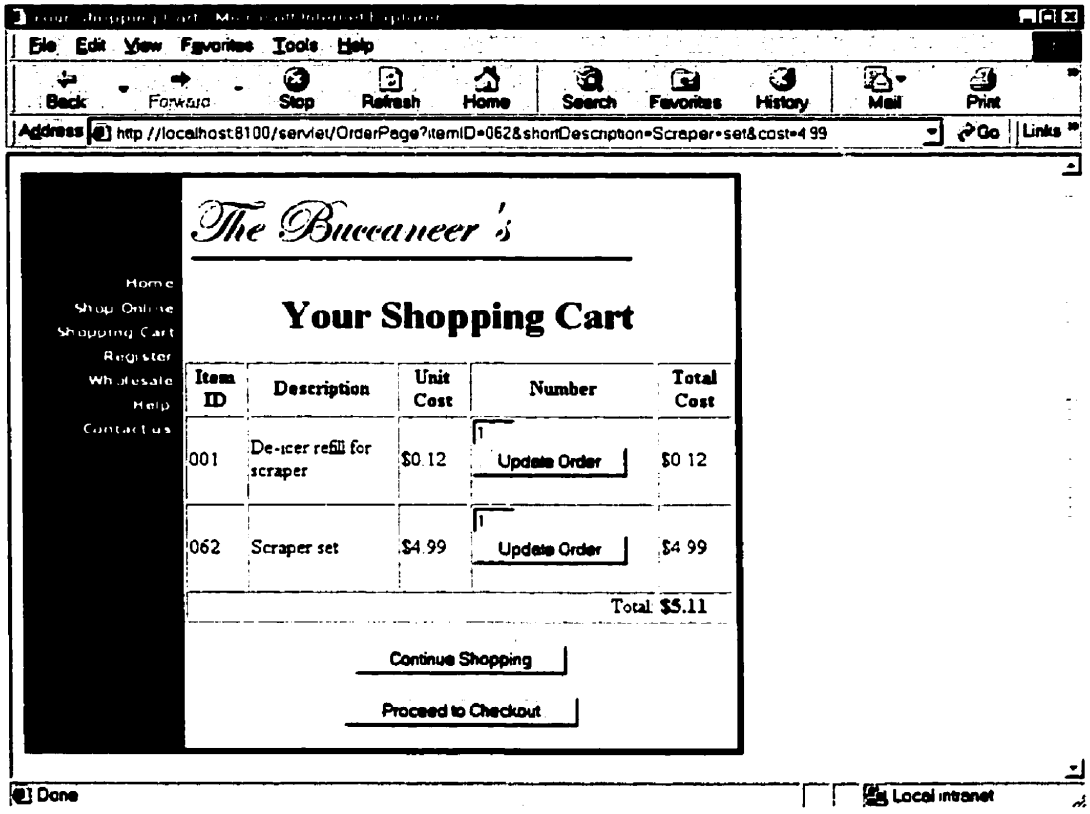

Figure 4.9 Shopping Cart

The most important technology used in Shopping Cart is "Session Tracking". HTTP is a stateless protocol. Every time a client retrieves a Web page, it opens a separate connection to the Web server. The server does not automatically maintain contextual information about clients. This lack of context makes online transaction more difficult. The customers cannot choose multiple products, because when they choose a second one, the information for the first one is lost. When they check out, the server would not know which previously created shopping carts are theirs. To solve this problem, servlets provide the HttpSession API. The API is built on top of HTTP Cookies and URL-

rewriting. Servlet developers do not need to manipulate these details. We can look up an HttpSession by:

```
HttpSession session = request.getSession(true);
```

There are also a lot of predefined methods to get and set information to a session.

The main program for the Shopping Cart is "OrderPage". The program logic is shown in Figure 4.10.

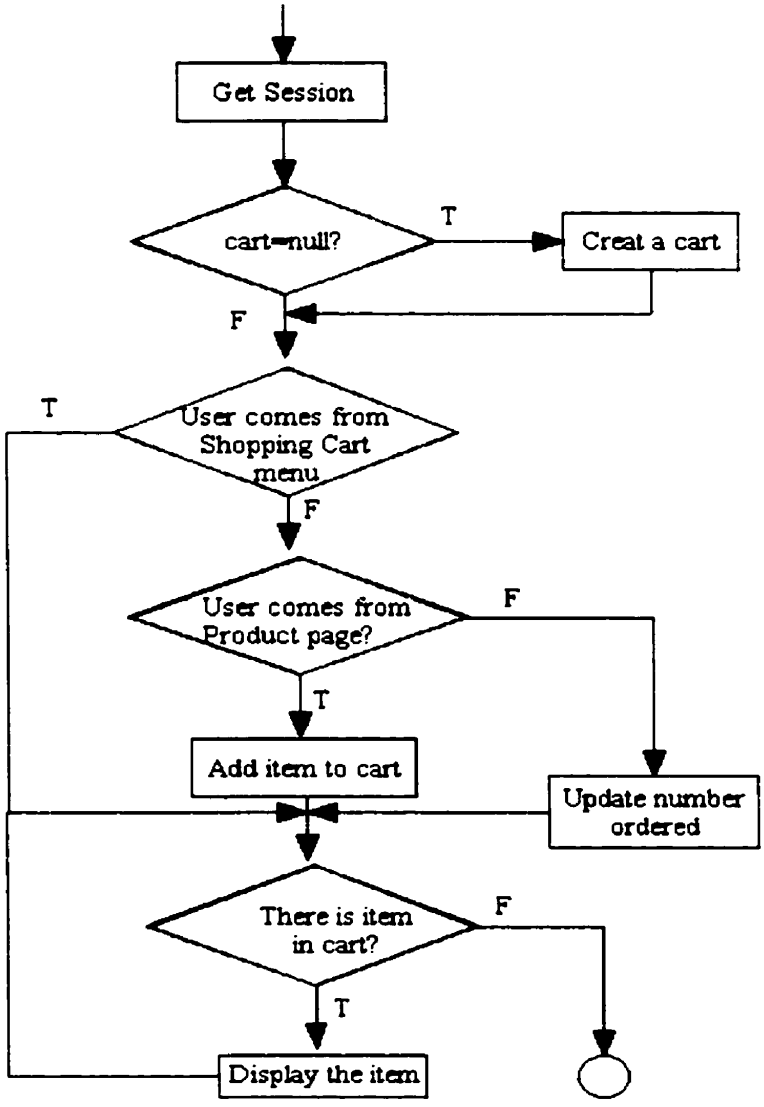

Figure 4.10 OrderPage.java

The whole Shopping Cart structure includes several other classes. Here is a very brief introduction to these classes. Only class names and class variables are listed. For class functions and complete programs, please refer to Appendix A.

```
/* class Item is the prototype of each product item. Every object of class Item is a record
of a real product*/
public class Item {
 private String itemID;
 private String shortDescription;
 private String longDescription;
 private double cost;
}
/* class ItemOrder deals with the item ordered. For a same product, if a customer ordered
three times, there would be three objects of class Item, but only one object of class
ItemOrder*/
public class ItemOrder {
 private Item item;
 private int numItems;
}
/* class ShoppingCart manipulates the shopping cart. Its functions include creating a new
shopping cart, adding items or changing the shopping cart. */
public class ShoppingCart {
 private Vector itemsOrdered;
}
```

4.3.3.4 Register and Login

The main technology involved in these two pages is database manipulation. Take register.java as an example. After getting a user's information from the registration form, we need to check whether the username has been taken. If so, the program will redirect the user to another page to choose an alternative username. Otherwise, the program saves

all the information into the database. Figure 4.11 shows the register page. Figure 4.12 shows the Login page.

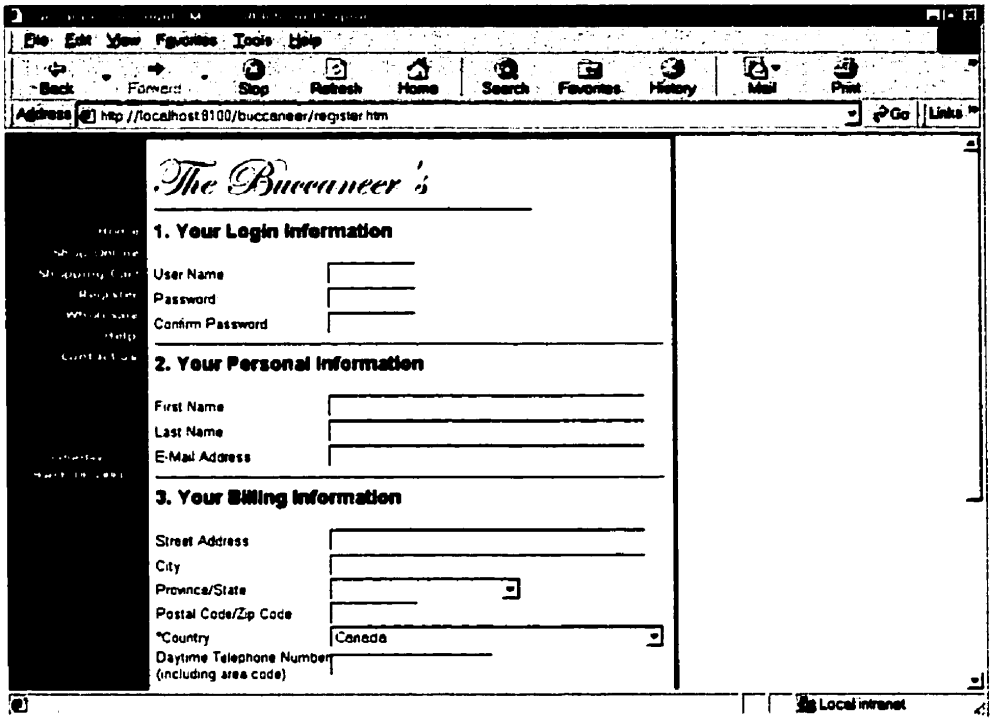

Figure 4.11 Register Page

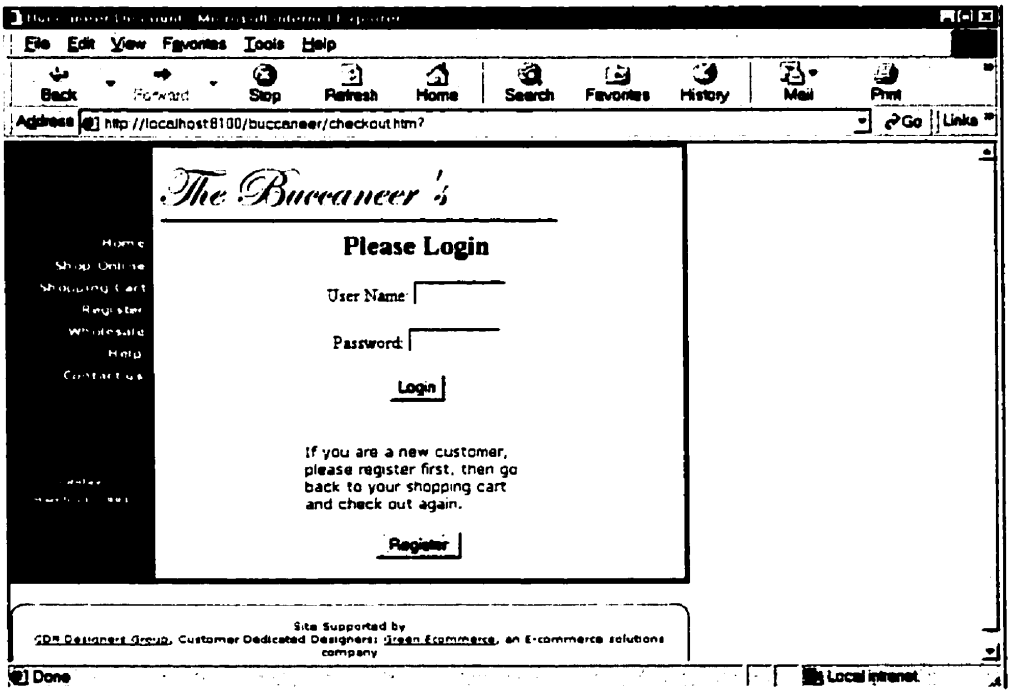

Figure 4.12 Login Page

The database part of the program is shown below:

```
try
{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
catch(ClassNotFoundException cnfe)
{ out.print("Sorry, our database is experiecing some problems. Please come again");}
try
{ String sourceURL = "jdbc:odbc:buccaneer";
  Connection databaseConnection =DriverManager.getConnection (sourceURL);
  Statement statement = databaseConnection.createStatement();
  String query1  = "select * from Customer where UserName= '" + username + "'";
  ResultSet result = statement.executeQuery (query1);
  if(result.next())
  gotoPage("/buccaneer/usernametaken.htm", request, response);
 else
{ String query2 ="insert into Customer " +  "values('" + username + "', '" + password  +
  "', '" +  firstname + "', '" + lastname  + "', '" + email + "', '" + address1 + "', '" + city
  + "', '" + province + "', '" + postalcode + "', '" + country + "', '" + dayphone + "')";
   statement.executeUpdate (query2);
   databaseConnection.close();
}
catch(SQLException e)
{out.print("Sorry, our database is experiecing some problems. Please come again later");}
```

4.3.3.5 Database

According to the requirement specification, an average-performance database is quite enough. To minimize the development cost, we chose Microsoft Access. Access is a part of Microsoft Office, which is available on almost every PC. Besides, the user-friendly interface makes it easier to set up and maintain than many other databases. Although Access uses a technology that makes it a poor choice for large amounts of traffic, The

Buccaneer site isn't expecting a high level of concurrent traffic according to our requirement specification.

We developed two tables in The Buccaneer database. The structure is shown in Figure 4.13. Note that we do not have a product table in the database. Due to the fact that there are less than 50 kinds of products in The Buccaneer's Web site, and most of them are not likely to change, we hard-coded them in the HTML files.

Customer
Username
Password
First Name
Last Name
Email
Address1
City
Province
Postal Code
Country
Day Phone

Orders
Username
Credit Cart Type
Credit Card Num
Card Holder
Exp Date Mon
Exp Date Year
Product ID
Quantity
Total Price

Figure 4.13 Database Tables

4.4 Building with WebSphere

We have spent a whole chapter (Chapter three) on WebSphere. WebSphere is a typical and one of the best e-commerce solution packages on the market. We are going to set up The Buccaneer Web site on WebSphere in this section.

4.4.1 Development Platform

In the previous solution, we needed to choose an appropriate development environment. In WebSphere, however, everything is bundled together. Developers can use them as a whole, or choose some of them to go with each company's requirements. The Buccaneer Web site does not have specific environment requirements, so we will use everything that comes with WebSphere. Let's take a look at the settings:

- Hardware: Pentium III PC (As specified in requirements specification)
- Operating System: Window NT4.0
- Web Server: IBM HTTP Server (based on Apache)
- Java Runtime Environment: Java Development Kit 1.1.7
- Application Server: WebSphere Application Server
- Java Editor: VisualAge for Java
- HTML editor: IBM Homepage Builder
- Database: DB2 Universal Database
- Picture Editor: IBM PerfectPhoto
- Browser: Internet Explorer

Amazingly, all the above listed are IBM software except the operating system, browser and JRE. Therefore, these software can be expected to work together more smoothly and efficiently.

4.4.2 Implementation

There are plenty of ways to set up a Web site on WebSphere. For example, if we work with WebSphere Commerce Suite, we have three methods: using the Store Creator, using the Store Profile Editor or using the Commerce Suite Administrator. As an alternative, we can directly configure IBM HTTP server and WebSphere Application Server. Or, we can even skip HTTP Server, and set up the whole Web site on WebSphere Application Server. In the following sections, we try two of the methods.

4.4.2.1 Using the Store Creator

This is the simplest and quickest way to set up a Web site. A Wizard will guide us through all the necessary steps in setting up a functional Web site. We only need to do whatever we are prompted to do. After the Web site is created, we can modify it using WebSphere Commerce Suite Administrator templates or directly modify the Net.Data files.

Here are the steps we went through in setting up The Buccaneer's Web site:

1. **Going through the store creator wizard**
   - Select a store model. We selected "Retail Store" model. This store model allows users to browse and search the catalog, choose from several registration options, create an address book, view the interest list, view order status, view customer service information.
   - Enter store information, which includes store name, store directory and default currency
   - Enter contact information, such as address, phone number
   - Select a registration type. We selected "Register when ordering". With this type, shoppers can browse without registering, but must register and log on before placing an order.
   - Select payment options. In this step, we enabled The Buccaneer to accept both online and offline payment.
   - Select a store style. We selected "New wave – denim" for our store.
   - Select the location of the navigation bar. We chose to put the navigation bar on the left side.
   - Select images. In this step, we can choose store logo and banner image.
   - Select colors and background images
   - Add sample products, taxes, and shipping

2. **Publish the store**
   - Publish the store in WebSphere Studio. WebSphere automatically produces detailed publishing report.
   - Test the store with the sample products

3. **Modify store settings in the WebSphere Commerce Suite Administrator**
   - Delete sample products
   - Specify shipping services
   - Set tax rates as 14.5%
   - Create product categories in Store Manager, Product Categories. WebSphere organize the categories in a tree structure and number all the categories in its own manner. We need to input all the category names, descriptions, parent category information, and assign display templates.
   - Input product information in Store Manager, Product Information. For each product, there are four forms to fill out: Attributes, Prices, Templates and Parent Categories. It was a long and tedious work.

4. **Modifying Web pages**
   - Modify navigation bar. We deleted some unnecessary links like "Address book". Some of the menu names are changed to go with The Buccaneer's style. For example, we changed "Customer Services" to "Contact Us", "Categories" to "Shop Online". All these changes are made with Net.Data.
   - Modify instructions and confirmation. We modified some instructions on the ordering page and the confirmation after an order is made. All these changes are made for the purpose of serving the shoppers better when they go through the ordering processes.

Figure 4.14 shows the homepage of The Buccaneer's Web site developed by Store Creator. The look and feel of this Web site is very different than the one we developed in the previous section. This is because Store Creator has its own selection of styles. It saves

a lot of time to use one of these existing styles. However, it is difficult to personalize the style. The main menu is not the same as the previous one either due to the same reason. To customize the functions, developers need to manually modify the Net.Data files created by Store Creator. It is a time-consuming task. We talk more about this in Chapter five.

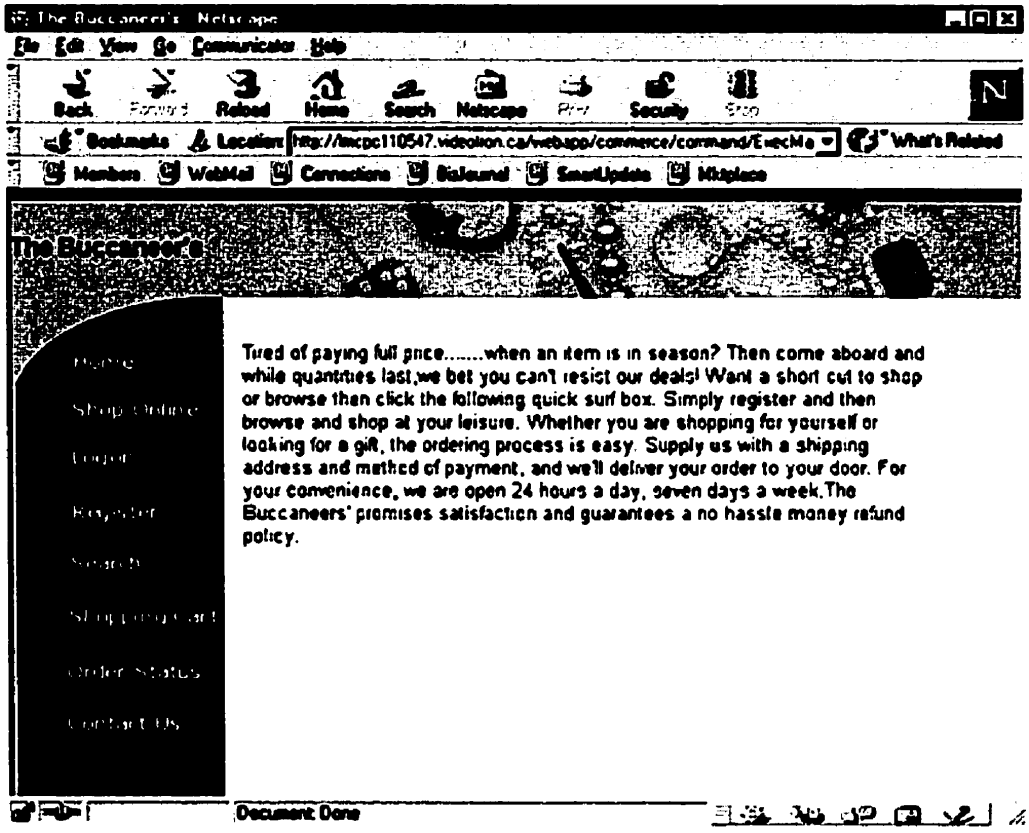

Figure 4.14 Home Page Produced by Store Creator

4.4.4.2 Building on WebSphere Application Server

To set up a Web site on WebSphere Application Server requires higher technical skills, but it provides great flexibility. Here is how we set up the site:

1. Configure a Web application

Firstly, we need to configure a Web application in the WebSphere Advanced Administrative Console as shown in Figure 4.15.

In this task wizard, we went through four steps: One, name the Web application, select system servlets to add to the Web application, select JSP version. Two, choose a servlet engine, which is WCS servlet Engine in our case. Three, specify the virtual host and a unique Web path for invoke the Web application. Four, specify advanced settings such as the servlet context attributes, reload intervals and whether to automatically reload servlet classes that have been modified. After the wizard, the command "WebApplication.create" will be running and a new application is recorded.

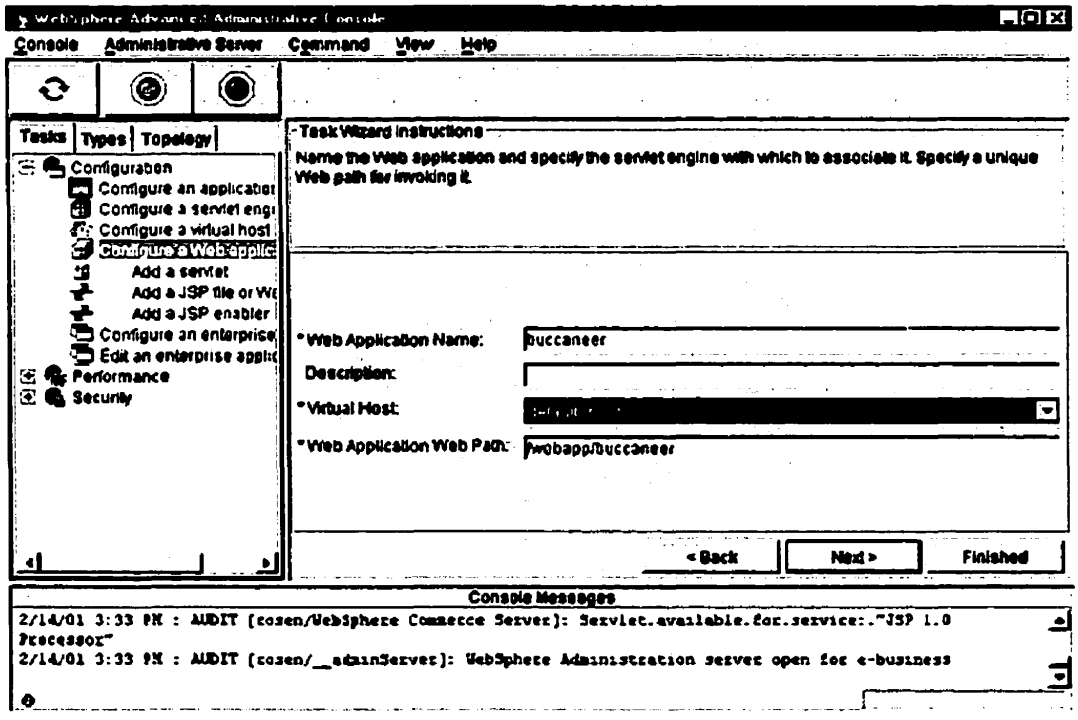

Figure 4.15 Configure a Web Application

2. Add a "file" Servlet

After setting up the application, we need to add a servlet called "File" to the application. The "File" servlet deals with the process of common files such as HTML,

JPG and GIF, communicates with HTTP Server, ensures a correct output to the clients. The servlet can be added in the WebSphere Advanced Administrative Console by specifying its servlet name, class name and Web path list.

3. **Create Web pages**

   In this method, we do not have the help from Store Creator or any other wizards or templates. We need to develop every page by ourselves. We developed the HTML files in Homepage builder and Java files in VisualAge for Java. The application architecture is exactly the same as we described in Section 4.3.2. All the HTML files and Java files are similar to those developed in JRun Studio except the directory addresses and hyperlinks are different.

4. **Save pages according to the application structure**

   To avoid further configuration, we need to store all the pages in a particular directory structure. For The Buccaneer's Web site, all the HTML, JGP and GIF files should be in the directory:

   WAServer Root\hosts\default_host\buccaneer\web

   All the Java and Class files should be in:

   WAServer Root\hosts\default_host\buccaneer\servlets

5. **Database**

   We use DB2 Universal Database in this solution. DB2 is a fully functional database. We can use the DB2 Control Center to set up The Buccaneer's database. The database structure is the same as the one shown in Figure 4.5. We set all the tables as "Low maintenance – system managed space". This will save us from a lot of other configuration.

Now, we have finished the development with WebSphere Application Server. At the client side, the look and feel of this Web site is exactly the same as the one shown in Figure 4.3. Users will not notice any difference between them. However, at the server side, they are two totally different systems with almost nothing in common.

We have developed The Buccaneer's Web site with the two solutions: building from scratch and building with WebSphere. In the latter case, we used two different methods. These implementations provided part of the research basis for the comparison in Chapter five.

Chapter 5

Comparison

For businesses, especially those small businesses that do not have extra money to burn, the biggest challenge is choosing the right e-commerce development solution. Ultimately, the solution they choose will determine whether money is made or lost on the Web. However, to find out the best solution is not an easy task. There are plenty of aspects we should consider. In this chapter, we will compare the two solutions we developed in Chapter four: building from scratch or building with WebSphere.

The comparisons are mainly based on our experience in developing The Buccaneer's Web site using the above-mentioned methods. But only one sample is not convincing. To make the results statistically correct, we surveyed two other e-commerce projects built from scratch. These two projects are similar to the one we did in Chapter four. We also surveyed five WebSphere development groups. All the members in these groups are university students majoring in computer science. Each group developed an e-commerce Web site with WebSphere. Three of the sites require only basic e-commerce functions and minimum personalization. These sites are built by Store Creator. The other two groups developed their own Web pages and configured their applications on WebSphere HTTP Server and WebSphere Application Server. These two sites have their own styles and unique functions.

In the following sections, we will compare the time, complexity, performance and cost of the two solutions. As we mentioned before, some sites have minimum requirements of personalization while some other sites require their own styles or even need to integrate with the existing systems. These two scenarios have very different development processes. Therefore we need to separate them in the following comparisons. Let us call the first one "simple sites", the second one "moderate sites". Surely there are also "comprehensive sites", but this thesis mainly focuses on small businesses. We will not cover those comprehensive sites in this comparison.

5.1 Time

Time is an objective factor in comparing the solutions. We will compare the time involved in the whole development process as well as the time in upgrade and maintenance periods.

Apparently, experienced programmers will spend less time than entry-level programmers; Web developers will spend less time than normal programmers. To make the results objective, we assume the developers are familiar with Java, database and basic Web design, but do not have experience in servlet or WebSphere, neither do they have previous experience in e-commerce system programming. Thus the "time" we are comparing here includes not only development time, but time to learn the new technologies.

For those simple sites, if we develop them from scratch, we need to learn Java servlet and JSP, which will cost about 3-5 days. To select, learn and configure a development platform probably requires 4-5 days. Requirement collection and design normally cost 2 weeks. Implementation costs 3-5 days. That is a total of 21-25 days. It will not costs a lot of time to maintain and upgrade these sites, because no new technology is necessary. The "upgrade" we were talking about here only includes those within the "simple sites" definition, like enlarge the database or add more links. If a simple Web site is becoming

so successful that it needs an upgrade into a major fully functional Web site, then that is another story.

An estimation of the time is shown in Figure 5.1. The vertical axis shows the update and maintenance time. The horizontal axis shows the development time. The positions of the four boxes roughly indicate the time required by these solutions.

If we develop these simple sites with WebSphere, we still need 2 weeks for requirement collection and design. Learning and installing WebSphere will cost 4-6 weeks. Implementation with Store Creator costs 10 minutes. Some simple personalization costs 1-2 days. That is a total of about 31-42 days. To maintain and upgrade these sites will cost lots of time because all the files generated by Store Creator are written in Net.Data language. Net.Data is IBM's solution to JSP. However, it is exclusively used in WebSphere. Developers need to spend extra time in learning Net.Data when they upgrade those sites.

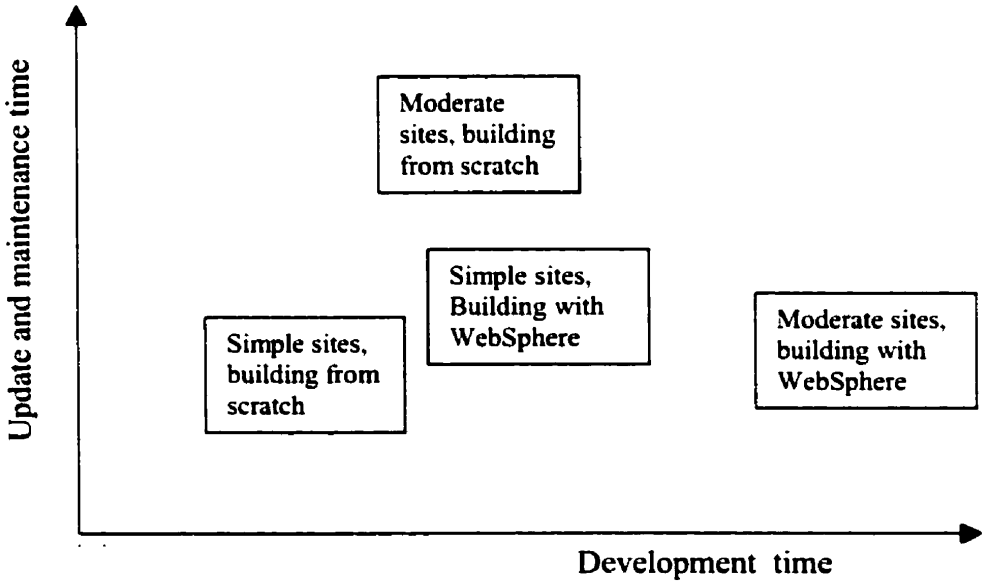

Figure 5.1 Time Comparison

Likewise, for moderate sites, if we develop them from scratch, we also need to learn Java servlet and JSP, which will cost about 3-5 days. To select, learn and configure a development platform probably need 4-5 days. Requirement collection and design will cost a little bit longer than those simple sites, normally about 3 weeks. Implementation costs 8-10 days. That is a total of 30-35 days. To maintain and upgrade these sites sometimes is not an easy task. Some functions like Product Advisor, marketing administrator are very complicated and time consuming. We gave out our estimation in Figure 5.1.

To develop a moderate site with WebSphere, we still need about 3 weeks for requirement collection and design. Learning and installing WebSphere will cost 8-10 weeks. Implementation with WebSphere Application costs 8-10 days. That is a total of about 63-75 days. It seems that to start with WebSphere takes more time than to build from scratch. However, to maintain and to upgrade these sites are relatively easier because a lot of the e-commerce functions are already included in WebSphere. Developers only need to integrate them into the system.

5.2 Complexity

According to computer dictionaries, "Complexity" normally indicates the level in difficulty in solving mathematically posed problems as measured by the time, number of steps or arithmetic operations, or memory space required. Those are called time complexity, computational complexity, and space complexity, respectively. The "complexity" we are talking about in this section is the level of difficulty in developing a Web site, which includes the following issues: installation and implementation, administration and maintenance, catalog and product development, personalization and customer relations, marketing and merchandising. They are measured by development time, technologies used, programming involved, and experiences needed.

Some solutions are easy to install and implement, but may be hard to personalize. The site we did with WebSphere Store Creator was an example for that. Some solutions are

more difficult to start up, but they become easier when the Web site grows, for example the WAS solution we presented for moderate Web sites. The complexity of a Web site normally decides what kind of developers are needed, entry level or experienced. Figure 5.2 shows the complexity of each solution in developing Web sites. Please note, the small sites building from scratch is not applicable for the category "marketing and merchandising". We only compared the other three in this category.

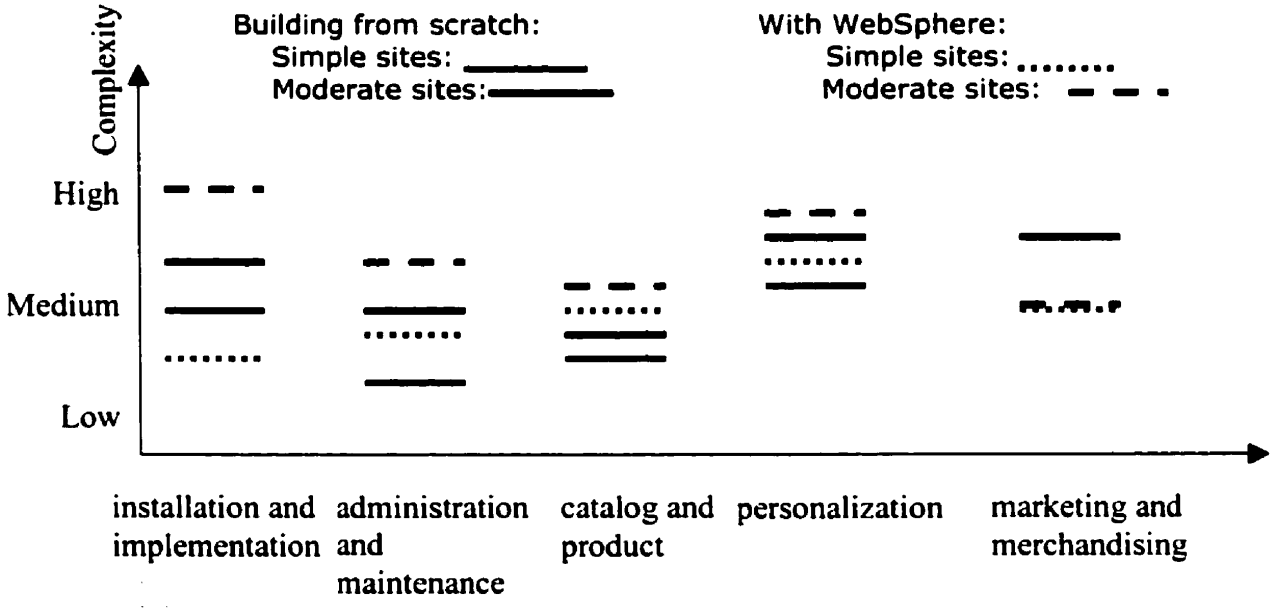

Figure 5.2 Complexity

5.3 Performance

The word "Performance" has different meanings in different circumstances. In the Web development domain, performance means the fulfillment of the request specification, the ability to perform efficiently, and the potential to perform better.

In terms of the fulfillment of the request specification, both solutions are satisfying. They may vary in time, cost and effort in achieving that goal, as we described in other sections, but at least both of them can provide the functions that they are supposed to provide.

The ability to perform efficiently, however, is different with the two solutions. The ability to perform efficiently includes various aspects such as multiple platform support, easy administration and functionalities. Those that supported multiple platform and databases were awarded extra points. We took points away for limited platform support, difficulty during installation. In order to get a score of very good in the administration, a solution should be easy to conduct administrative functions such as monitoring or reporting. We took points away for solution that has poor functionality, such as poor database performance in terms of dealing with large amounts of traffic.

For the first solution, building from scratch, we chose the development environment as economically as possible. As a result, the overall performance is not as good as WebSphere. As long as the Web site is small and the transactions online are moderate, the performance problem is not a big issue. However, as the Web site grows, the difference between the two solutions will become more and more apparent.

To attract new customers and business partners, a simple shopping cart system for the Web storefront will eventually become insufficient. For a strong e-commerce presence, we should drive shoppers to the site via sophisticated site analysis, offer shoppers personalization features, and keep those shoppers coming back with a competitive and intuitive design, one-to-one marketing features, and Web-based customer service. The ability to adopt all these functions is the so-called "potential to perform better". WebSphere has the great advantage in this area. Almost all the above-mentioned functions are well integrated into WebSphere. For example, WCS Pro let the developers intuitively create discount and promotional rates, shopper groups, and product auctions; for shopper personalization, it includes an interactive catalog builder that lets shoppers have a more personalized shopping experience. WebSphere is definitely the winner in the long run in terms of the performance.

5.4 Cost

Cost is the outlay or expenditure made to achieve an objective. The reason that we put this topic last is that all the above-mentioned factors eventually lead to the difference of costs. For example, the more complicated a solution is, the more experienced developers must be to develop the site. More experienced developers normally have higher salaries. Thus the cost of the whole development cost will be higher. Development time is also a big factor in determining the cost. The longer the time is, the higher the cost will be. Performance seems to be irrelevant with the development cost. However, that is not true. If a solution fails to meet the requirement specification, the cost for redesign could be significantly high. Furthermore, performance is closely related with the maintenance and upgrade costs. Therefore, when evaluating these various solutions, we should not only consider the cost of the package but the considerable amount of money on e-commerce development environment, necessary additional hardware, the salary for the developers, the cost to integrate the site with existing systems, the cost to customize it to suit any individual needs. Often what looks like an inexpensive setup at the beginning can end up being a big budget when trying to add new features or redesign.

The start up price for our first solution is pretty low. Costs of all the software and hardware add up to a total of no more than $5,000. For simple sites, the development cost is calculated on an assumption of $280 a day (a typical entry-level contractor's salary). For a moderate site, the development cost is calculated as $400 per day (a typical medium-level contractor's salary). The WebSphere Pro edition is sold at $38,000, which is reasonably priced for a software package of its kind. Most of the similar packages on the market are priced in the same range. The start up cost is the package price plus hardware price plus development price. The development price is also calculated as $280 per day for simple sites and $400 per day for moderate sites. Table 5.1 shows how we calculated the total development cost of each methods.

	Simple sites		Moderate sites	
	Building from scratch	**Building with WebSphere**	**Building from scratch**	**Building with WebSphere**
Software cost	$1000	$38,000	$1000	$38,000
Hardware cost	$2,000	$2,000	$2,000	$2,000
Developer's salary	$6,160 ($280 per day for 22 days)	$9,800 ($280 per day for 35 days)	$12,800 ($400 per day for 32 days)	$28,000 ($400 per day for 70 days)
Total	$9,160	$49,800	$15,800	$68,000

Table 5.1 Development Cost Comparisons

The upgrade and maintenance costs for every solution are closely related to their performance because the ability to adopt new functions (as we described in the "performance" section) is the main factor in determining these costs. Figure 5.3 shows a rough comparison of the costs for the two solutions.

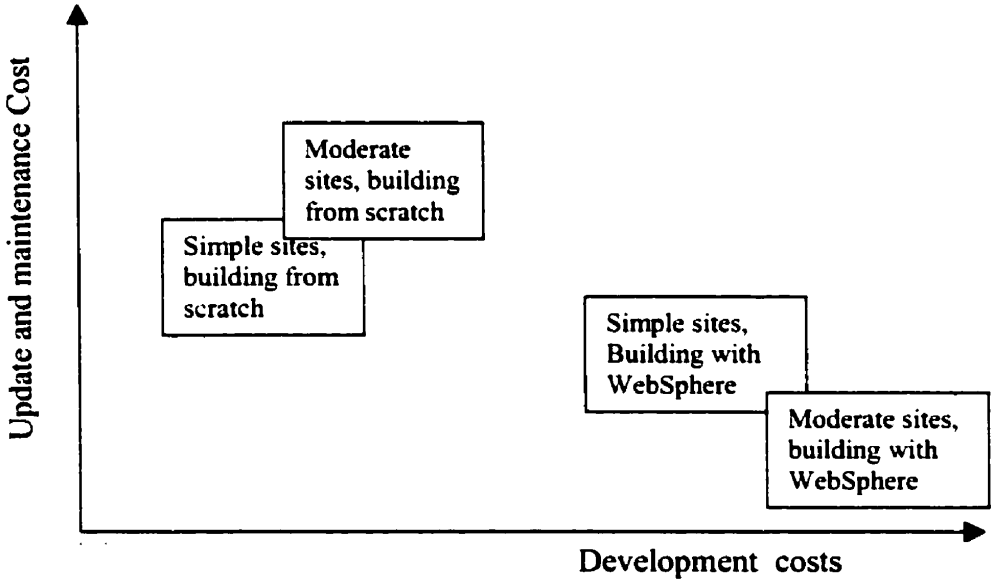

Figure 5.3 Costs

5.5 Summary

As more and more companies become involved in e-commerce, choosing the right e-commerce solution becomes increasingly important. Our goal is to find out the best solution for small companies. We looked for solutions that provide an environment for developing a competitive Web site and give companies room for growth and integration with an existing infrastructure. We presented and compared two e-commerce solutions that fit the goal: building from scratch with Java servlet on Apache and JRun, and building with IBM WebSphere.

We installed them on appropriate hardware platforms, operating systems, and databases. To test our solutions, we created an online discount store: The Buccaneer. With the first solution, we programmed everything from scratch and linked to the Access database. For the second solution, we accessed Store Creator wizards and sample templates to complete our online store. We modified store code using Net.Data to evaluate the skills a user would need to customize a site. We also developed another version of the store using WebSphere Application Server. To make our comparisons more accurate, we surveyed two e-commerce projects building from scratch and five WebSphere projects. We consulted their developers for the development time, development plans, developers' experiences, programs, costs, and project performance. We used these data in the comparison.

We evaluated the time spent, the complexity, the performance and costs to develop the Web sites with each solution. All the solutions made a strong showing with impressive features and a good degree of scalability. Building from scratch with Java servlets is economical, efficient and flexible. It is a really good solution for small companies who need a standard-functioning Web site to extend their selling channels. WebSphere is functionally stronger, but the start up cost is way too much for some of the small businesses according to Figure 5.3. Also, if what these businesses need is only shopping cart and online transaction systems, they may never need most of the fancy functions in

WebSphere. It would be a waste of time and money to buy a huge package like WebSphere.

On the other hand, WebSphere offers the widest array of tools out of the box, including the Payment Manager transaction server and a remarkably easy-to-use auction-building feature; it also makes site customization a smooth process. It assembles IBM products that are already mature, including DB2 Universal Database, so components are well integrated and easy to manage. Those medium companies who are concerned with setting up competitive Web sites, streamline their business processes and paving a road to a successful future should definitely select WebSphere. They will have to spend a considerable amount of money on starting up, but they will see that WebSphere is worth every penny they spent. According to Figure 5.3, we can clearly see that in the maintenance and upgrade period, the cost for the WebSphere solution will eventually turn out to be lower than the other solution.

Chapter 6

Conclusion and Future Work

This thesis mainly focused on finding a good e-commerce solution for small businesses. We introduced some commonly used technologies in developing e-commerce systems. The topics covered from Web standards and protocols to Web planning and design, from Web servers to server side programming. We also introduced an e-commerce software package WebSphere. Using these technologies, we found out two good solutions: building from scratch with mainly Java technologies and building with IBM's WebSphere. We developed an online store with each of the solutions. In order to find out the advantages and disadvantages of these solutions, we made a comprehensive comparison. Developing time, complexity, performance and cost are evaluated in our comparison. From these comparisons, we concluded that building from scratch with Java servlets is economical, efficient and flexible. It is a really good solution for small companies who need a standard-functioning Web site to extend their selling channels. On the other hand, WebSphere offers the widest array of tools. Those medium companies who are concerned with setting up competitive Web sites, streamline their business processes and paving a road the successful future should definitely select WebSphere. We hope this conclusion is helpful to businesses.

There are plenty of other e-commerce solutions available. We only compared two of them. It is definitely possible that there are better solutions than ours. Future research should be conducted to include more possible solutions and to test more functions. Solutions for comprehensive Web sites should also be studied.

E-commerce is developing so fast that all kinds of new tools and software packages are emerging every day. A TV may still look like a TV in twenty years, but e-commerce could become a totally different thing in ten years. No matter whether we are business people or Web developers, we should always be aware of the fast development of e-commerce technology in order to provide the clients with the best services.

References

1. Web Security, Amrit Tiwana, Digital Press 1999
2. Using HTML 4, Java 1.1 and JavaScript 1.2, Second Edition, Eric Ladd and Jim O'Donnell, Que Corporation, 1998
3. Survival Guide to Web Site Development, Mary Haggard, Microsoft Press, 1998
4. Mastering JavaScript, James Jaworski, Sybex Inc., 1997
5. Active Server Pages Bible, Eric A. Smith, IDG Books Worldwide, Inc., 2000
6. Building Professional Web Sites with the Rifht Tools, Jeff Greenberg, J.R. Lakeland, Hewlett-Packard Company, 1999
7. Bulid a Web Site, The Programmer's Guide to Create, Building, and Maintaining a Web Presence, Net.Genesis and Devra Hall, Net.Genesis 1995
8. A Windows NT Guide to The Web, Richard Raucci, Springer-Verlag New York, Inc., 1997
9. Developing Java Beans, Robert Englander, O'Reilly & Associates, Inc., 1997
10. Guide to Building Intelligent Websites with JavaScript, Nigel Ford, John Wiley & Sons, Inc., 1998
11. How to Set Up and Maintain a World Wide Web Site, The Guide for Information Providers, Lincoln D. Stein, Addison-Wesley Publishing Company, 1995
12. How to Set Up and Maintain a Web Site – Second Edition, Lincoln D. Stein, Addison-Wesley Longman, Inc., 1997
13. Java How to Program – Second Edition, H.M. Deitel, P.J. Deitel, Prentice-Hall, Inc., 1998
14. http://www.online-commerce.com/tutorial.html
15. http://hotwired.lycos.com/webmonkey/e-business/tutorials/tutorial3.html
16. Core Java Web Server, Chris Taylor, Tim Kimmet, Prentice Hall PRT, 1999
17. UNIX Web Server, second edition, R.Douglas Matthews, Paul Jones, Jonathan Magid, Donald A. Ball, JR., Michael J. Hammel, Ventana, 1997

18. Web Site Engineering, Thomas A. Powell, David L. Jones, Dominique C. Cutts, Prentice Hall PTR, 1998
19. Programming.java An Introduction to Programming Using Java, Rick Decker, Stuart Hirshfield, PWS Publishing Company, 1998
20. HTML 4.0 Sourcebook, Ian S. Graham, John Wiley&Sons, Inc, 1998
21. The Windows NT Web Server Handbook, Tom Sheldon, McGraw-Hill, Inc., 1996
22. The Non-Designer's Web Book, Robin Williams, John Tollett, Peachpit Press, 1998
23. Introduction to E-Commerce, Tan Zheng, pptph.com, 2000
24. Security Electronic Transaction, http://www.setco.org
25. On-line computer dictionary, http://wombat.doc.ic.ac.uk
26. RFC Hypertext Transfer Protocol, http://src.doc.ic.ac.uk/computing/internet/rfc/rfc2068.txt
27. Interface Design in E-commerce, Yu Xing, McGill University, 2000
28. Java, http://java.sun.com/
29. Collaborative Web Development, Strategies and Best Practices for Web Teams, Jessaca Burdman, Addison Wesley Longman, Inc., 1999
30. The World Wide Web Complete Reference, Rick Stout, McGraw-Hill, Inc., 1996
31. Web Server Technology, The Advanced Guide for World Wide Web Information Providers, Nancy J. Yeager, Robert E. McGrath, Morgan Kaufmann Publishers, Inc., 1996
32. Programming Web Components, Reaz Hoque, Tarun Sharma, The McGrau-Hill Companies, Inc., 1998
33. Database Driven Web Sites, Jesse Feiler, Academic Press, 1999
34. Developing an E-business Application for the WebSphere Application Server, John Akerley, Murtuza Hashim, Alexander Koutsoumbos, Angelo Maffione, International Technical Support Organization, http://www.redbooks.ibm.com
35. Creating a Store, IBM WebSphere Commerce Suite documentation
36. WebShpere Application Server standard edition, Getting Started, IBM WebSphere Commerce Suite documentation

37. WebSphere Application Servers: Standard and Advanced Editions, Barry Nusbaum, Matias Djunatan, Wakako Jinno, Peter Kelley, International Technical Support Organization, http://www.redbooks.ibm.com
38. Web-Based Specification and Integration of Legacy Services, Ying Zou, Kostas Kontogiannis, Dept. of Electrical & Computer Engineering, University of Waterloo, November, 2000
39. IBM WebSphere Commerce Suite Fundamentals, International Business Machines Corporation, 2000
40. IBM WebSphere Commerce Suite Pro Edition for Windows NT Installation Guide, International Business Machines Corporation 1996, 2000
41. IBM WebSphere Performance Pack Usage and Administration, Marco Pistoia, Vincenzo Iovine, Stefano Pischedda, International Technical Support Organization, http://www.redbooks.ibm.com
42. NCSA World Wide Web Server: Design and Performance, R.E.McGrath, D.A.Reed, IEEE Computer, Volume 28, Number 11, November 1995
43. Designing Multinational Online Stores: Vhallenges, Implementation Techniques and Experience, Yumman Chan, IBM Toronto Lab, CASCON 2000, November, 2000
44. End-to-end E-commerce Application Development Based on XML Tools, W.KU et.al., IEEE Data Engineering, Vol. 23, No.1
45. Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as Used in the World-Wide Web, Berners-Lee, T. RFC 1630, CERN June, 1994
46. The World Wide Web, Berners-Lee, T., T. Cailiar, H.F.Nielsen, Communications of the ACM, Volume 37, Number 8, August, 1994
47. The Secure Hypertext Transfer Protocol, Rescorla,E., A.Schiffman, Internet Draft, July 1995
48. Witan Web and the Software Engineering of Web-based Applications, J Howard Johnson and Stephen A. Mackay, Institute for Information Technology, National Research Council, November, 2000

49. Security models for Web-based applications, James B. D. Joshi, Walid G. Aref, Arif Ghafoor and Eugene H. Spafford, Communications of the ACM, Volume 44, No. 3, March 2001

50. Conversational interfaces for e-commerce applications, Mark Lucente, Communications of the ACM, Volume 43 , No. 12, December 2000

51. A cost and performance model for Web service investment, Kai R. T. Larsen and Peter A. Bloniarz, Communications of the ACM, Volume 43, No. 2, February 2000

APPENDIX A – Source Code

The source codes listed here are the java files used in Section 4.3.3. They are part of the implementation of our first solution: building from scratch. We also wrote many HTML files in that solution, but because they are lengthy and simple, we will not list them in the appendix. The Java files written for the WebSphere solution are similar to the files listed here. All the files are sorted in alphabetic order.

1. Finish.java

```
/* Finish.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.text.NumberFormat;
import java.lang.*;
import java.lang.String.*;
import java.lang.Double.*;

public class Finish extends HttpServlet
{
 public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
   { PrintWriter out = response.getWriter();
    HttpSession session = request.getSession(true);
    String username = (String)session.getValue("username");
    ShoppingCart cart;
    cart = (ShoppingCart)session.getValue("shoppingCart");

    if (cart != null)
    {
      String type = request.getParameter("type");
      String cardholder = request.getParameter("cardHolder");
```

```
    String cardnumber = request.getParameter("cardNumber");

   String expdatemonth = request.getParameter("expiryDateMonth");
   String expdateyear = request.getParameter("expiryDateYear");
   Vector itemsOrdered = cart.getItemsOrdered();
   ItemOrder order;

   /* Database connection*/
    try {Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
    catch(ClassNotFoundException cnfe){ out.print("Sorry, our database is experiecing
some problems. Please come again");}

    try
    { String sourceURL = "jdbc:odbc:buccaneer";
      Connection databaseConnection =DriverManager.getConnection (sourceURL);
       Statement statement = databaseConnection.createStatement();
        out.print("database ready. ");

/* save orders in database*/
        for(int i=0; i<itemsOrdered.size(); i++)
        { order = (ItemOrder)itemsOrdered.elementAt(i);
          String query ="insert into Orders " +
          "values('" + username
                     + "', '" +  type
                     + "', '" + cardnumber
                     + "', '" + cardholder
                     + "', '" + expdatemonth
                     + "', '" + expdateyear
                     + "', '" + order.getItemID()
                     + "', '" + order.getNumItems()
                     + "', '" + order.getTotalCost()
                     + "')";
         statement.executeUpdate (query);
        }
          databaseConnection.close();
          session.invalidate();
          gotoPage("/buccaneer/finish.htm", request, response);

       }
       catch(SQLException e){out.print("Sorry, our database is experiecing some
problems.  Please come again later");}

   } else out.print("no shopping cart");

 }
```

```
 private void gotoPage (String address, HttpServletRequest request, HttpServletResponse
response)
 throws ServletException, IOException
 {
 RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
 dispatcher.forward(request, response);
 }


/* POST and GET requests handled identically. */

  public void doPost(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException
     {
      doGet(request, response);
     }
}
```

2. Item.java

```
/* Item.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

public class Item
{
 private String itemID;
 private String shortDescription;
 private String longDescription;
 private double cost;

 public Item(String itemID, String shortDescription, String longDescription, double cost)
{
   setItemID(itemID);
   setShortDescription(shortDescription);
   setLongDescription(longDescription);
   setCost(cost);
  }

 public String getItemID()
{
   return(itemID);
  }

 protected void setItemID(String itemID)
{
   this.itemID = itemID;
  }

 public String getShortDescription()
{
   return(shortDescription);
  }

 protected void setShortDescription(String shortDescription)
{
   this.shortDescription = shortDescription;
  }

 public String getLongDescription()
{
   return(longDescription);
```

```
 }

 protected void setLongDescription(String longDescription)
{
  this.longDescription = longDescription;
 }

 public double getCost()
{
  return(cost);
 }

 protected void setCost(double cost)
{
  this.cost = cost;
 }
}
```

3. ItemOrder.java

```
/* ItemOrder.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */


public class ItemOrder
{
 private Item item;
 private int numItems;

 public ItemOrder(Item item)
{
   setItem(item);
   setNumItems(1);
 }

 public Item getItem()
{
   return(item);
 }

 protected void setItem(Item item)
{
   this.item = item;
 }

 public String getItemID()
{
   return(getItem().getItemID());
 }

 public String getShortDescription()
{
   return(getItem().getShortDescription());
 }

 public String getLongDescription()
{
   return(getItem().getLongDescription());
 }
```

```
 public double getUnitCost()
{
  return(getItem().getCost());
 }

 public int getNumItems()
{
  return(numItems);
 }

 public void setNumItems(int n)
{
  this.numItems = n;
 }

 public void incrementNumItems()
{
  setNumItems(getNumItems() + 1);
 }

 public void cancelOrder()
{
  setNumItems(0);
 }

 public double getTotalCost()
{
  return(getNumItems() * getUnitCost());
 }
}
```

4. Login.java

```
/* Login.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.text.NumberFormat;
import java.lang.*;
import java.lang.String.*;
import java.lang.Double.*;

public class Login extends HttpServlet
{
 public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
 { PrintWriter out = response.getWriter();
   String username = request.getParameter("userName");
   String password = request.getParameter("password");

   if ((username != null) && (password != null ))
   {
    /* check whether the user name and password pair is valid*/

    try{ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
       catch(ClassNotFoundException cnfe){};
     try {  String sourceURL = "jdbc:odbc:buccaneer";
      Connection databaseConnection =DriverManager.getConnection (sourceURL);
      Statement statement = databaseConnection.createStatement();
      String query = "select * from Customer where UserName = '"
                                              + username + "' and Password = '"
                                              + password + "' ";
      ResultSet result = statement.executeQuery (query);

       if(result.next())
       {
         HttpSession session = request.getSession();
         session.putValue("username", username);
         gotoPage("/buccaneer/payment.htm", request, response);
```

```
            databaseConnection.close();}
          else
          {
            gotoPage("/buccaneer/checkoutagain.htm", request, response);
            databaseConnection.close();}
        }
        catch(SQLException e){};
      }
      else
      {
        out.print("no input");
        gotoPage("/buccaneer/checkoutagain.htm", request, response);}
}

private void gotoPage (String address, HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
  RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
  dispatcher.forward(request, response);
}

/* POST and GET requests handled identically. */

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
      doGet(request, response);
    }
}
```

5. OrderPage.java

```
/* OrderPage.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.text.NumberFormat;
import java.lang.*;
import java.lang.String.*;
import java.lang.Double.*;

public class OrderPage extends HttpServlet
{  public void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException
     {
      HttpSession session = request.getSession(true);
      ShoppingCart cart;
      synchronized(session)
      {
        cart = (ShoppingCart)session.getValue("shoppingCart");
         if (cart == null)
         {
           cart = new ShoppingCart();
           session.putValue("shoppingCart", cart);
          }
         String itemID = request.getParameter("itemID");

         String shortDescription =  request.getParameter("shortDescription");
         String longDescription = "";
         String itemcost = request.getParameter("cost");
         double cost = 0 ;
          if (itemcost != null)
          { cost = (Double.valueOf(itemcost)).doubleValue();}
          Item newitem;
          newitem = new Item(itemID,shortDescription, longDescription, cost);

           if (itemID != null)
           {
             String numItemsString = request.getParameter("numItems");

           /* Shoppers come from "add to cart" button"*/
```

```
            if (numItemsString == null)
            {
              cart.addItem(newitem);
             }

           /* Shopper came from "update order" button*/
            else
             {
              int numItems;
               try
               {
                  numItems = Integer.parseInt(numItemsString);
               } catch(NumberFormatException nfe) {
               numItems = 1;
              }
            cart.setNumOrdered(newitem, numItems);
          }
        }
      }

      /* Output*/
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Your Shopping Cart";
      out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=\"#FFFFFF\">\n" +
            "<table width=\"70%\" border=0 cellspacing=0 cellpadding=0>" +
            "<tr><td height=\"3\" bgcolor=\"#3F388F\"></td></tr>" +
                              "<tr><td>" +
            "<table border=0 width=100% cellspacing=0 cellpadding=0> <tr>" +
            " <td width=\"2\" bgcolor=\"#3F388F\" rowspan=\"2\"> </td>" +
            " <td width=\"120\" bgcolor=\"#321B5F\" align=\"right\" valign=\"top\">" +
            " <table width=\"120\" border=0 cellspacing=0 cellpadding=0>" +
            "<tr><td height=80> </td></tr>" +
            "<tr><td align=\"right\"><a href=\"../buccaneer/index.html\">
             <img src=\"../buccaneer/home.jpg\" border=0></a></td></tr>" +
            "<tr><td align=\"right\"><a href=\"../buccaneer/products.htm\">
             <img src=\"../buccaneer/shop.jpg\" border=0></a></td></tr>" +
            "<tr><td align=\"right\"><img src=\"../buccaneer/cart.jpg\" border=0>
             </td></tr>" +
            "<tr><td align=\"right\"><a href=\"../buccaneer/register.htm\">
             <img src=\"../buccaneer/register.jpg\" border=0></a></td></tr>" +
            "<tr><td align=\"right\">
             <a href=\"../buccaneer/heypassword.htm\"
onMouseOver=\"window.status='none' ;return true\" >
```

```
            <img src=\"../buccaneer/wholesale.jpg\" border=0></a></td></tr>" +
           "<tr><td align=\"right\"><a href=\"../buccaneer/help.htm\">
            <img src=\"../buccaneer/help.jpg\" border=0></a></td></tr>" +
           "<tr><td align=\"right\"><a href=\"../buccaneer/contact.htm\">
            <img src=\"../buccaneer/contact.jpg\" border=0></a></td></tr>" +
           "<tr><td height=\"80\"> </td></tr></table></td>" +
           " <td bgcolor=\"#000000\" width=\"2\"> </td>" +
           " <td align=left valign=top height=100%>" +

           "<table border=\"0\" width=100% height=100%
background=\"../buccaneer/bg_map.jpg\" cellspacing=\"0\" cellpadding=\"0\">\n" +
           "<tr><td>\n" +
           " <br>  <img src=\"../buccaneer/buccaneer.gif\">\n" +
           " <br>  <img src=\"../buccaneer/line.jpg\">\n" +
           "<P><H1 ALIGN=\"CENTER\">" + title + "</H1>");
      synchronized(session)
     {
      Vector itemsOrdered = cart.getItemsOrdered();
       if (itemsOrdered.size() == 0)
       {
         out.println("<center><H2><I>No items in your cart...</I></H2>" +
                     "<form action=\"../buccaneer/products.htm\">" +
                     "<input type=submit name=back value=\"Continue Shopping\">
                      </form></center>");
        }
        else
        {

        /* Show every item in the cart*/
            out.println
            ("<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
             "<TR BGCOLOR=\"#FFFFCE\">\n" +
             "<TH>Item ID<TH>Description\n" +
             "<TH>Unit Cost<TH>Number<TH>Total Cost");
              ItemOrder order;

              NumberFormat formatter =
              NumberFormat.getCurrencyInstance();

              String formURL ="/servlet/OrderPage";
              formURL = response.encodeURL(formURL);
             double totalcost = 0;


              for(int i=0; i<itemsOrdered.size(); i++) {
              order = (ItemOrder)itemsOrdered.elementAt(i);
```

```
            out.println
             ("<TR>\n" +
              "  <TD>" + order.getItemID() + "\n" +
              "  <TD>" + order.getShortDescription() + "\n" +
              "  <TD>" +
               formatter.format(order.getUnitCost()) + "\n" +
              "  <TD>" +
              "<FORM ACTION=\"" + formURL + "\">\n" +
              "<INPUT TYPE=\"HIDDEN\" NAME=\"itemID\"\n" +
              "       VALUE=\"" + order.getItemID() + "\">\n" +
              "<INPUT TYPE=\"TEXT\" NAME=\"numItems\"\n" +
              "       SIZE=3 VALUE=\"" +
               order.getNumItems() + "\">\n" +
              "<SMALL>\n" +
              "<INPUT TYPE=\"SUBMIT\"\n "+
              "       VALUE=\"Update Order\">\n" +
              "</SMALL>\n" +
              "</FORM>\n" +
              "  <TD>" +
             formatter.format(order.getTotalCost()));
             totalcost=totalcost + order.getTotalCost();
        }
      out.println( "<tr><td colspan=4 align=right> Total: </td><td><b>" +
       formatter.format(totalcost) + "</b></td></tr>");
       String checkoutURL =  response.encodeURL("../buccaneer/checkout.htm");

       out.println
        ("</TABLE>\n" +
                  "<FORM ACTION= ../buccaneer/products.htm>\n" +
        "<BIG><CENTER>\n" +
                   "<INPUT TYPE=\"SUBMIT\"\n" +
        "       VALUE=\"Continue Shopping\">\n" +
        "</CENTER></BIG></FORM>" +
        "<FORM ACTION=\"" + checkoutURL + "\">\n" +
        "<BIG><CENTER>\n" +
                   "<INPUT TYPE=\"SUBMIT\"\n" +
        "       VALUE=\"Proceed to Checkout\">\n" +
        "</CENTER></BIG></FORM>");
       }
        out.println
         ("</td><td width=1 align=left bgcolor=\"#3F388F\"> </td> </tr></table>"+
          "</td></tr> <tr>
          <td bgcolor=\"#3F388F\" height=\"4\" valign=center colspan=\"4\"></td></tr>" +
          "</table></td></tr></table> </BODY></HTML>");
    }
  }
```

```
/* POST and GET requests handled identically. */

 public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {   doGet(request, response);
    }
}
```

6. Payment.java

```
/* Payment.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.text.NumberFormat;
import java.lang.*;
import java.lang.String.*;
import java.lang.Double.*;

public class Payment extends HttpServlet
{
 public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
 {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String title = "Payment method";
  out.println(ServletUtilities.headWithTitle(title) +
        "<BODY BGCOLOR=\"#FFFFFF\">\n" +
        "<table width=\"70%\" border=0 cellspacing=0 cellpadding=0>" +
        "<tr><td height=\"3\" bgcolor=\"#3F388F\"></td></tr>" +
        "<tr><td>" +
        "<table border=0 width=100% cellspacing=0 cellpadding=0> <tr>" +
        " <td width=\"2\" bgcolor=\"#3F388F\" rowspan=\"2\"> </td>" +
        " <td width=\"120\" bgcolor=\"#321B5F\" align=\"right\" valign=\"top\">" +
        " <table width=\"120\" border=0 cellspacing=0 cellpadding=0>" +
        "<tr><td height=80> </td></tr>" +
        "<tr><td align=\"right\"><a href=\"../buccaneer/index.html\">
        <img src=\"../buccaneer/home.jpg\" border=0></a></td></tr>" +
        "<tr><td align=\"right\"><a href=\"../buccaneer/products.htm\">
        <img src=\"../buccaneer/shop.jpg\" border=0></a></td></tr>" +
        "<tr><td align=\"right\"><img src=\"../buccaneer/cart.jpg\" border=0></td>
        </tr>"+
        "<tr><td align=\"right\"><a href=\"../buccaneer/register.htm\">
        <img src=\"../buccaneer/register.jpg\" border=0></a></td></tr>" +
        "<tr><td align=\"right\">
```

```
        <a href=\"../buccaneer/heypassword.htm\" onMouseOver=\"window.status='none'
;return true\" >
        <img src=\"../buccaneer/wholesale.jpg\" border=0></a></td></tr>" +
          "<tr><td align=\"right\"><a href=\"../buccaneer/help.htm\">
          <img src=\"../buccaneer/help.jpg\" border=0></a></td></tr>" +
          "<tr><td align=\"right\"><a href=\"../buccaneer/contact.htm\">
          <img src=\"../buccaneer/contact.jpg\" border=0></a></td></tr>" +
          "<tr><td height=\"80\"> </td></tr></table></td>" +
          " <td bgcolor=\"#000000\" width=\"2\"> </td>" +
          " <td align=left valign=top height=100%>" +
          "<table border=\"0\" width=100% height=100%
background=\"../buccaneer/bg_map.jpg\" cellspacing=\"0\" cellpadding=\"0\">\n" +
          "<tr><td>\n" +
          " <br>  <img src=\"../buccaneer/buccaneer.gif\">\n" +
          " <br>  <img src=\"../buccaneer/line.jpg\">\n" +
          "</td></tr></table>\n" +
          "<table border=0 align=center bordercolor=\"#321B5F\" cellspacing=0
cellpadding=0>\n" +
          " <Form action=\"gotoPage(/servlet/Finish\" method=\"Post\"><tr>
           <td Colspan=2 align=left>\n" +
          "<font face=arial, geneva, helvetica, sans-serif size=4 color=Maroon><B>" +
          "Your Credit Card Information</B></font><br>\n" +
          " <font face=\"arial, geneva, helvetica, sans-serif\" size=2>
           All information are required </font><br><br></td>\n" +
          "</tr> <tr bgcolor=\"#FFFFCC\"> <td>
           <font face=\"arial, geneva, helvetica, sans-serif\" size=2>Type</font>
          </td>\n" +
          "<td><select name='type' value=''><option value=''>
          <option value=\"1\">Visa</option><option value=\"2\">MasterCard</option>
          <option value=\"3\">American Express</option>\n" +
          "</select></td></tr>\n" +
          "<tr bgcolor=\"#FFFFCC\"><td>
          <font face=\"arial, geneva, helvetica, sans-serif\" size=2>Credit Card Number:
          </font></td>\n" +
          "<td> <input name=\"cardNumber\" size=40 MAXLENGTH=20 value=\"\" >
           </td></tr>\n" +
          " <tr bgcolor=\"#FFFFCC\"> <td>
           <font face=\"arial, geneva, helvetica, sans-serif\" size=2>
            Card Holder</font></td>\n" +
          "<td> <input name=\"cardHolder\" size=40 MAXLENGTH=40 value=\"\" >
           </td></tr>\n" +
          "<tr bgcolor=\"#FFFFCC\"> <td>
          <font face=\"arial, geneva, helvetica, sans-serif\" size=2>Expiry Date
          </font></td>\n" +
          "<td Valign=top>
```

```
            <input size=2 MAXLENGTH=2 name=\"expiryDateMonth\" VALUE=\"\"
>/\n" +
            "<input size=2 MAXLENGTH=4 name=\"expiryDateYear\" value=\"\" >
             </td></tr></table></center>\n" +
             "<CENTER> <table border=0 align=center width=\"80%\" cellspacing=0
cellpadding=2>\n" +
            "<tr> <td align=center>
             <br><input type=\"submit\" value=\"Continue\" name=\"Submit\"
                  alt=\"Submit\">    \n" +
            "<input type=\"reset\" value=\"CLEAR\" name=\"reset\" alt=\"reset\">
             </td></tr> </table> </form> </CENTER>\n" +
            "</td> <td width=1 align=left valign=top bgcolor=\"#3F388F\"> 
             </td></tr></table>\n" +
            " </td></tr><tr>
              <td bgcolor="#3F388F" height="4" valign="center" colspan="4">
              </td></tr></table></td></tr></table>" );

      }

 private void gotoPage (String address, HttpServletRequest request, HttpServletResponse
response)
 throws ServletException, IOException
 { RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
 dispatcher.forward(request, response);
 }

  /* POST and GET requests handled identically. */

  public void doPost(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException
     {
      doGet(request, response);
      }
}
```

7. Register.java

```
/* Register.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.text.NumberFormat;
import java.lang.*;
import java.lang.String.*;
import java.lang.Double.*;

public class Register extends HttpServlet
{
 public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
 { PrintWriter out = response.getWriter();
   String username = request.getParameter("userName");
   String password = request.getParameter("password");
   String firstname = request.getParameter("firstName");
   String lastname = request.getParameter("lastName");
   String email = request.getParameter("email");
   String address1 = request.getParameter("address1");
   String city = request.getParameter("city");
   String province = request.getParameter("province");
   String postalcode = request.getParameter("postalCode");
   String country = request.getParameter("country");
   String dayphone = request.getParameter("dayPhone");


   try {Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}
   catch(ClassNotFoundException cnfe)
   { out.print("Sorry, our database is experiecing some problems. Please come again");}
   try
   {  String sourceURL = "jdbc:odbc:buccaneer";
      Connection databaseConnection =DriverManager.getConnection (sourceURL);
       Statement statement = databaseConnection.createStatement();
       String query1  = "select * from Customer where UserName= '" + username + "'";
       ResultSet result = statement.executeQuery (query1);
```

```
    if(result.next()) gotoPage("/buccaneer/usernametaken.htm", request, response);
    else
   {
       String query2 ="insert into Customer " +
                                    "values('" + username
                                       + "', '" +  password
                                        + "', '" +  firstname
                                        + "', '" +  lastname
                                        + "', '" +  email
                                        + "', '" +  address1
                                        + "', '" +  city
                                        + "', '" +  province
                                        + "', '" +  postalcode
                                        + "', '" +  country
                                        + "', '" +  dayphone
                                        + "')";

       statement.executeUpdate (query2);
       databaseConnection.close();
       response.setContentType("text/html");
       String htmltitle = "Register";
       out.println(ServletUtilities.headWithTitle(htmltitle) +
         "<BODY BGCOLOR=\"#FFFFFF\">\n" +
         "<table width=\"70%\" border=0 cellspacing=0 cellpadding=0>" +
         "<tr><td height=\"3\" bgcolor=\"#3F388F\"></td></tr>" +
         "<tr><td>" +
         "<table border=0 width=100% cellspacing=0 cellpadding=0> <tr>" +
         " <td width=\"2\" bgcolor=\"#3F388F\" rowspan=\"2\"> </td>" +
         " <td width=\"120\" bgcolor=\"#321B5F\" align=\"right\" valign=\"top\">" +
         " <table width=\"120\" border=0 cellspacing=0 cellpadding=0>" +
         "<tr><td height=80> </td></tr>" +
         "<tr><td align=\"right\"><a href=\"../buccaneer/index.html\">
          <img src=\"../buccaneer/home.jpg\" border=0></a></td></tr>" +
         "<tr><td align=\"right\"><a href=\"../buccaneer/products.htm\">
          <img src=\"../buccaneer/shop.jpg\" border=0></a></td></tr>" +
         "<tr><td align=\"right\"><a href=\"../servlet/OrderPage\">
          <img src=\"../buccaneer/cart.jpg\" border=0></a></td></tr>" +
         "<tr><td align=\"right\"><a href=\"../buccaneer/register.htm\">
           <img src=\"../buccaneer/register.jpg\" border=0></a></td></tr>" +
         "<tr><td align=\"right\">
          <a href=\"../buccaneer/heypassword.htm\"
onMouseOver=\"window.status='none' ;return true\" >
          <img src=\"../buccaneer/wholesale.jpg\" border=0></a></td></tr>" +
         "<tr><td align=\"right\"><a href=\"../buccaneer/help.htm\">
          <img src=\"../buccaneer/help.jpg\" border=0></a></td></tr>" +
         "<tr><td align=\"right\"><a href=\"../buccaneer/contact.htm\">
```

```
            <img src=\"../buccaneer/contact.jpg\" border=0></a></td></tr>" +
           "<tr><td height=\"80\"> </td></tr></table></td>" +
           " <td bgcolor=\"#000000\" width=\"2\"> </td>" +
           " <td align=left valign=top height=100%>" +



           "<table border=\"0\" width=100% height=100%
background=\"../buccaneer/bg_map.jpg\" cellspacing=\"0\" cellpadding=\"0\">\n" +
           "<tr><td valign=top>\n" +
           " <br>  <img src=\"../buccaneer/buccaneer.gif\">\n" +
           " <br>  <img src=\"../buccaneer/line.jpg\">\n" +
           "<P><font size=3 ALIGN=\"CENTER\">  
            <b>You are successfully registered!</b><P>
            <b> Please remember your username:<I>"+ username+ "
            </I>  and password:<I>" + password + "</b></font>");

        out.println
          ("</td><td width=1 align=left bgcolor=\"#3F388F\"> </td>
            </tr></table>" +
           "</td></tr> <tr><td bgcolor=\"#3F388F\" height=\"4\" valign=center
colspan=\"4\"></td></tr>" +
           "</table></td></tr></table> </BODY></HTML>");
      }
  }
  catch(SQLException e){out.print("Sorry, our database is experiecing some problems.
Please come again later");}


 }

 private void gotoPage (String address, HttpServletRequest request, HttpServletResponse
response)
 throws ServletException, IOException
 { RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
 dispatcher.forward(request, response);
 }

 /* POST and GET requests handled identically. */

 public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
      doGet(request, response);
    }
}
```

8. ServletUtilities.java

```
/* ServletUtilities.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

import javax.servlet.*;
import javax.servlet.http.*;

/* Some simple time savers*/
public class ServletUtilities
{
 public static final String DOCTYPE =
  "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
  "Transitional//EN\">";

 public static String headWithTitle(String title)
{
  return(DOCTYPE + "\n" +
       "<HTML>\n" +
       "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
 }

 public static int getIntParameter(HttpServletRequest request,String paramName,
                                                    int defaultValue)
 {
  String paramString = request.getParameter(paramName);
  int paramValue;
  try {
   paramValue = Integer.parseInt(paramString);
    } catch(NumberFormatException nfe)
    { // null or bad format
   paramValue = defaultValue;
  }
  return(paramValue);
 }

 public static String getCookieValue(Cookie[] cookies, String cookieName,
                                                    String defaultValue)
{
     if (cookies != null)
     {
        for(int i=0; i<cookies.length; i++)
        {
     Cookie cookie = cookies[i];
```

```
      if (cookieName.equals(cookie.getName()))
        return(cookie.getValue());
      }
    }
    return(defaultValue);
  }
    public static Cookie getCookie(Cookie[] cookies, String cookieName)
{
      if (cookies != null)
      {
        for(int i=0; i<cookies.length; i++)
        {
      Cookie cookie = cookies[i];
      if (cookieName.equals(cookie.getName()))
        return(cookie);
      }
    }
    return(null);
  }
  public static String filter(String input)
{
    StringBuffer filtered = new StringBuffer(input.length());
    char c;
      for(int i=0; i<input.length(); i++)
      {
     c = input.charAt(i);
     if (c == '<')
     {
      filtered.append("&lt;");
     } else if (c == '>')
     {
      filtered.append("&gt;");
     } else if (c == '"')
     {
      filtered.append("&quot;");
     } else if (c == '&')
     {
      filtered.append("&amp;");
     } else
     {
      filtered.append(c);
     }
    }
    return(filtered.toString());
  }
}
```

9. ShoppingCart.java

```
/* ShoppingCart.java
 * Project:  Buccaneer
 * Author:  Yu Xing
 * Data: 20, Jan. 2001 */

import java.util.*;

public class ShoppingCart
{
 private Vector itemsOrdered;


 public ShoppingCart()
{
   itemsOrdered = new Vector();
 }

 public Vector getItemsOrdered()
{
   return(itemsOrdered);
 }


 public synchronized void addItem(Item newitem)
{
   ItemOrder order;
   for(int i=0; i<itemsOrdered.size(); i++)
   {
    order = (ItemOrder)itemsOrdered.elementAt(i);
    if (order.getItemID().equals(newitem.getItemID()))
     {
      order.incrementNumItems();
      return;
     }
   }
   ItemOrder newOrder = new ItemOrder(newitem);
   itemsOrdered.addElement(newOrder);
 }
```

```
 public synchronized void setNumOrdered(Item newitem, int numOrdered)
{
  ItemOrder order;
  for(int i=0; i<itemsOrdered.size(); i++)
  {
   order = (ItemOrder)itemsOrdered.elementAt(i);
   if (order.getItemID().equals(newitem.getItemID()))
    {
     if (numOrdered <= 0)
     {
      itemsOrdered.removeElementAt(i);
     }
     else
     {
      order.setNumItems(numOrdered);
     }
     return;
    }
  }
  ItemOrder newOrder =
   new ItemOrder(newitem);
  itemsOrdered.addElement(newOrder);
 }
}
```