E-Mail Supported, Agent-Based Portal for Automobile Sales

by

Gabriel Komakech

B.Sc (Makerere University), Microsoft Certified Professional

Department of Information Systems

e-mail: gkomakech@dicts.mak.ac.ug, tel.: +256-772-212422

A Project Report Submitted to School of Graduate Studies in Partial Fulfilment for the Award of Master of Science in Computer Science Degree of Makerere University

February, 2006
Declaration

I Gabriel Komakech do hereby declare that this Project Report is original and has not been published and/or submitted for any other degree award to any other University before.

Signed ................................................................. Date........................................
Gabriel Komakech
B.Sc (Computer Science, Chemistry), Microsoft Certified Professional
Approval

This Project Report has been submitted for Examination with the approval of the following supervisor/s.

Signed ..........................................................................Date........................................
Dr. V. Baryamureeba
Department of Computer Science
Faculty of Computing and Information Technology
Makerere University
Dedication

To my wife, son and daughter: Lucy, Nicholas and Karen. Thank you for the encouragement, patience and putting up with my late hours of reading and coming home. You are a special trio.
Acknowledgment

1. To God, through him the impossible can be made possible.

2. To my supervisor Dr. V. Baryamureeba for his time, guidance and advice throughout the course of this project.

3. Faculty of Computing and Information Technology, for waiving my tuition fees.

4. Special thanks to Sida/SAREC for paying for the extension fees.

5. To my work colleagues at DICTS - Thank you for the support and encouragement you gave me!

6. To my Parents, brothers and sisters for your support and prayers.

7. To all our friends and family, for the encouragement, support and prayers.

8. To my pioneer class-mates: we have made it!! Good luck.
## Table of Contents

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>1.1</td>
<td>Background to the Study</td>
</tr>
<tr>
<td>1.2</td>
<td>Statement of the problem</td>
</tr>
<tr>
<td>1.3</td>
<td>Aim and Objectives of the Study</td>
</tr>
<tr>
<td>1.4</td>
<td>Justification of the Study</td>
</tr>
<tr>
<td>1.5</td>
<td>Scope</td>
</tr>
<tr>
<td>2</td>
<td>Literature Review</td>
</tr>
<tr>
<td>2.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2.2</td>
<td>Agent Communication</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Sockets</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Shared Memory</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Semaphores</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Message queues</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Named pipes</td>
</tr>
<tr>
<td>2.2.6</td>
<td>Remote Procedure Calls (RPC)</td>
</tr>
<tr>
<td>2.2.7</td>
<td>Mail slots</td>
</tr>
<tr>
<td>2.2.8</td>
<td>Memory mapped files</td>
</tr>
<tr>
<td>2.3</td>
<td>Internet Technology</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Simple Mail Transfer Protocol (SMTP)</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Post Office Protocol (POP)</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Internet Mail Access Protocol (IMAP)</td>
</tr>
<tr>
<td>2.3.4</td>
<td>E-mail Models</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Internet message format - Message Headers</td>
</tr>
<tr>
<td>2.3.6</td>
<td>Internet message format - Message body</td>
</tr>
<tr>
<td>2.3.7</td>
<td>HyperText Transfer Protocol (HTTP)</td>
</tr>
<tr>
<td>2.3.8</td>
<td>Web Servers</td>
</tr>
<tr>
<td>2.4</td>
<td>Agent related studies</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Kasbah</td>
</tr>
<tr>
<td>2.4.2</td>
<td>MAGMA</td>
</tr>
<tr>
<td>2.4.3</td>
<td>OFFER</td>
</tr>
<tr>
<td>2.4.4</td>
<td>EMP</td>
</tr>
<tr>
<td>2.4.5</td>
<td>Agent-based workflow</td>
</tr>
<tr>
<td>2.5</td>
<td>E-Commerce Models</td>
</tr>
</tbody>
</table>
3 Methodology

3.1 Introduction ................................................. 22

3.2 The inception phase ........................................ 23
   3.2.1 Review of existing online stores ...................... 23

3.3 The Elaboration phase ..................................... 23

3.4 The construction phase .................................... 24

3.5 Tools ......................................................... 24

4 The Inception Phase .......................................... 25

4.1 Introduction ................................................ 25

4.2 Existing Online Stores .................................... 25
   4.2.1 Cars Uganda ............................................ 25
   4.2.2 Trust Company Limited (TCL) ......................... 26
   4.2.3 eBay ................................................... 27

4.3 Domain model context diagram for the automobile store .... 29
   4.3.1 Domain actors .......................................... 29
   4.3.2 Use Cases .............................................. 30
   4.3.3 Use Case Diagram ..................................... 32

4.4 Detailed Requirements Specification ........................ 32

5 The Elaboration Phase ........................................ 34

5.1 Introduction ................................................ 34

5.2 Deployment view .......................................... 35
   5.2.1 Description of each component ....................... 35

5.3 Logical View ............................................... 37
   5.3.1 The eStore Classes .................................... 40

5.4 The data view .............................................. 41
   5.4.1 The ACCOUNT Class and the ACCOUNT RELATION ... 42
   5.4.2 The WEBSITE Class and the REQUEST RELATION ... 42
   5.4.3 The MATCH Class and the MATCH RELATION ........ 42
   5.4.4 The AUTOMOBILE Class and the AUTOMOBILE RELATION 43

6 The Construction Phase ...................................... 44

6.1 Introduction ............................................... 44

6.2 Implementing the User Interface Layer ..................... 44
   6.2.1 The Log In Form - index.php ......................... 45
   6.2.2 The search form - search.php ......................... 47
   6.2.3 The Account maintenance form - account.php ........ 51

6.3 Implementing the Business Layer .......................... 53

6.4 Implementing the Data Layer ............................... 53

7 Conclusion, Recommendations and Future work ................ 56

7.1 Conclusion ................................................. 56

7.2 Recommendation .......................................... 57

7.3 Future Work ............................................... 57

Bibliography ..................................................... 58
List of Tables

4.1 eStore Actors ........................................... 29
4.2 eStore Use Case ......................................... 30
4.3 Requirement Specification ............................... 33
List of Figures

2.1 SMTP Client-Server model .................................................. 12
4.1 Domain Model context diagram ............................................ 29
4.2 UC-002000 Process flow .................................................... 31
4.3 UC-003000 Use Case Diagram ............................................. 32
5.1 eStore deployment configuration ......................................... 35
5.2 eStore Logical configuration ............................................... 37
5.3 eStore Class diagram ....................................................... 40
5.4 eStore database schema diagram ......................................... 41
6.1 eStore Logon Screen ....................................................... 45
6.2 eStore search criteria Screen ............................................ 47
6.3 Results generated by clicking Go to My Account .................... 49
6.4 Results generated by search.php form .................................. 49
6.5 Search Result sent to customer’s e-mail ............................... 50
6.6 eStore account maintenance Screen ..................................... 51
6.7 Output for creation of new account .................................... 52
6.8 Data Layer code snippet ................................................... 54
6.9 Data Layer code snippet - Cont......................................... 54
6.10 Automobile Catalog ....................................................... 55
7.1 process.php ........................................................................ 61
7.2 process.php cont.............................................................. 62
7.3 svrSearch.pl ..................................................................... 63
7.4 svSearch.pl cont.............................................................. 64
7.5 svSearch.pl cont.............................................................. 65
7.6 svrAccount.pl ................................................................. 66
**LIST OF ACRONYMS**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Agent Communication Language</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>B2B</td>
<td>Business-to-Business</td>
</tr>
<tr>
<td>B2C</td>
<td>Business-to-Consumer</td>
</tr>
<tr>
<td>C2C</td>
<td>Consumer-to-Consumer</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>E-Mails</td>
<td>Electronic Mails</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile communication</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communication Technology</td>
</tr>
<tr>
<td>IMAP</td>
<td>Internet Message Access Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter Process Communication</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>KQML</td>
<td>Knowledge Query and Manipulation Language</td>
</tr>
<tr>
<td>MAGMA</td>
<td>Minnesota Agent Marketplace Architecture</td>
</tr>
<tr>
<td>MTA</td>
<td>Mail Transfer Agent</td>
</tr>
<tr>
<td>MIDL</td>
<td>Microsoft Interface Definition Language</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>POP</td>
<td>Post Office Protocol</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
</tr>
<tr>
<td>RUP</td>
<td>Rotational Unified Process</td>
</tr>
<tr>
<td>SASL</td>
<td>Socket Authentication Security Level</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>TCL</td>
<td>Trust Company Limited</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>XNS</td>
<td>Xerox Network Systems</td>
</tr>
</tbody>
</table>
Abstract

Most organizations today have web presence as a way of quickly reaching out to a diverse clientele community, spread across the universe. Unlike in the early days of the Internet, the current Internet technology and infrastructure has matured to not only support display of static contents, but also support for dynamic contents that allows interactivity with the end-user. It also supports a mix of technology, such as integration with e-mail services. Although the environment offers more, very few organizations have taken advantage of its full potential.

This project realizes a 3-tier, web-based application for searching automobile portal. It was designed and developed to operate in asynchronous mode, where a web client logs on, sets search criteria and logs off. The system then searches for the client’s request and informs him/her using electronic mail (e-mail). The underlying technology deployed relied on component-based approach, with three independent software agents developed to handle the three key operations: Account management, Dispatching e-mail messages and searching for a client’s request. Unified Modeling Language (UML) was used as a modeling language, and Rotational Unified Process (RUP) as a software engineering methodology.

Web developers interested in building 3-tier web systems may find this work a good starting point.
Chapter 1

Introduction

1.1 Background to the Study

The Internet has dramatically changed how people sell and buy goods. In recent years we have seen the emergence of electronic marketplaces that leverage information technology to create more efficient markets. One of the greatest potentials of electronic marketplaces is in its ability to bring together buyers and sellers with greater effectiveness and at a massive scale. In deed, real-world examples of online marketplaces such as eBay (www.ebay.com), a popular online market, offers over 3 million items for sale at any given time. The ability to bring together buyers and sellers at such a massive scale may be both a blessing and curse for electronic marketplaces. On the one hand, it is a blessing since buyers reap the benefits of greater product diversity with potentially lower prices and sellers are able to reach a greater pool of potential buyers. However, a curse it may be since buyers have to put in extensive efforts in searching through an enormous amount of products; and sellers have to compete with a great number of sellers in order to effectively reach the potential buyers. This project sets out to develop software agents that help buyers by searching for products on their behalf, and later presenting them with matching results, via their e-mail addresses.

In the physical world, the use of agents or brokers in areas like real estates, recruitment and automobiles is very common. When an individual wants to buy a house for
example, a real estates agent is contacted and given details and specifications. The Agent uses these details to locate a matching house for its client.

Agents rely on fellow agents to effectively search and locate a matching product in a given domain. In this example, the agent given the task would go asking fellow agents "do you have a two-bed roomed house for rent?" In other situations, agents know who amongst them deals in a specific type of products. In this case, queries for that product would be directed only to those agents.

Searching for a product at existing online marketplaces requires that the user remains online until when results of the search are displayed, yet the search is done on backend web servers and web databases at far away locations with no additional involvement required from the user until when the result of the search is displayed, and yet the user remains online during the entire process, which may take from several minutes to hours of waiting. This can be very expensive for poor users, who pay colossal amounts to access the Internet. There is a big disadvantage with this method, especially in third world countries where existing Internet links are very unstable due to a combination of factors: -

- unreliable electricity supply;
- low service provision from Internet Service Providers (ISPs);
- use of inferior technology for Internet access (e.g. wireless infrastructure that is affected by interference in the unlicensed bands - 2.4GHz).

If the user’s internet link breaks down when a lengthy search request was about to be displayed, every thing will be lost and the user will have to start the lengthy procedures anew after the link is restored.
In Uganda, for example, most organizations, Internet Cafes and individuals can only afford a few kbps of bandwidth because of the high bandwidth cost in the country. In addition, the links often time-out in the middle of a search, thus frustrating the user who had issued the search and already waited for a long time for the results.

The software agents to be developed in this project will be capable of directing search results to a designated e-mail address. This is very useful especially in a situation when a user’s connection breaks off before results of the search could be displayed. The software agent to be developed presents Internet users with a powerful tool to efficiently tap the potential of electronic commerce and circumvent the bottleneck of high bandwidth cost in third world countries.

1.2 Statement of the problem

Internet users waste a lot of time locating matching products from online stores. Existing techniques require users to be online during the entire duration of the search.

This presents a big problem because users often spend too much time following dead or incorrect links, and at times unwanted links such as links to pornographic sites.
1.3 Aim and Objectives of the Study

The overall aim is to increase users productivity by ensuring that users spend less time searching the Internet. The specific objectives are:

(i) To study and analyze the operations and processes of existing online stores, including whether similar solution to the proposed does exist;

(ii) To design a model for an agent-based marketplace that sends search results to buyer’s e-mail address(es);

(iii) To develop a software tool, based on the model in (ii) above.

1.4 Justification of the Study

Time wasting: Users spend a lot of their time searching on the net from one link to the other. This is unnecessary waste of time. Computers can be tasked to carry out this work on behalf of the users, as computers are best at dealing with repetitive tasks.

Development in Mobile Phone Technology: Mobile phone technology is rapidly taking root in every corner of Uganda, and other third world countries. Mobile phone users can now access the Internet directly from their mobile phones or by using their mobile phones in conjunction with their computers to dial up their ISPs and gain access to the Internet. Because of the high charges levied by Mobile telephone service providers for these services, it is not economical for users to be online for extended durations. This is a good application area for the output of this research; providing mobile telephone users with a cheap and convenient way of accessing online stores. Users connect to the Internet, assign to an agent the task of locating a given product and then close down the connection. Upon completion of the assigned task, the agent then provide the feedback to the user’s designated e-mail address.
E-mail density: Comparatively, most third world countries have greater access to e-mail services than access to full Internet. By integrating e-mail services as an access path to the numerous online stores, disadvantaged users with limited Internet access, but with access to e-mail can benefit from the study output. Such users would for example, request their friends to schedule a search on their behalf.

1.5 Scope

The agents developed in this project were limited to only the automobile products.
Chapter 2

Literature Review

2.1 Introduction

In this chapter, related work about agent technology as well as the supporting technology were studied and analyzed. The following interprocess communication techniques were studied and the most appropriate one selected for the project:

- sockets;
- message queues;
- semaphores;
- shared memory;
- named pipes;
- mailslots;
- memory mapped files.

2.2 Agent Communication

The manner in which software agents communicate with one another is a fundamental detail that must be understood in order for one to be able to develop software agents. Inter-process Communication (IPC) mechanism is one way by which processes can exchange information. Some IPC mechanisms only support data exchange between
processes running on the same machine, while other IPCs connect processes running
on geographically dispersed machines. IPC is achieved through a variety of methods:-

2.2.1 Sockets

Steven (1999)[19] defines sockets as a method for communication between a client
program and a server program in a network. A socket is defined as ”the endpoint in
a connection”. When you create a socket, the operating system sets up internal data
structures to manage the socket which is referenced through the socket descriptor,
that gives you access to the socket. Sockets provide support for different kinds of
communications protocols and data streams. Example, Transmission Control Pro-
tocol/ Internet Protocol (TCP/IP), Xerox Network Systems (XNS) protocols and
many more. Of the various forms of IPC, sockets are by far the most popular. This is
because of their cross-platform communication capabilities. This kind of IPC was a
good consideration for the agent in this project because of the need for multi-platform
operatability.

2.2.2 Shared Memory

This is an interprocess communications technique in which the same memory is ac-
cessed by more than one program running in a multitasking operating system. Ac-
cording to Stevens (1999)[19], what is normally required, however, is some form of
synchronization between the processes that are storing and fetching information to
and from the shared memory region to prevent the applications from ”colliding” or
trying to update the same information at the same time. Synchronization is achieved
through: mutexes, condition variables, readwrite locks, record locks, and semaphores.
This technique is not directly implemented by Java and thus was not used in the
project since the software agents were developed using a mix of programming lan-
guages, including Java.
### 2.2.3 Semaphores

When a work process locks a resource, it sets a semaphore. Another work process that also wants to access it must then wait (Richardson, 1999)[19]. Consider a stretch of railroad in which there is a single track over which only one train at a time is allowed. Guarding this track is a semaphore. A train must wait before entering the single track until the semaphore is in a state that permits travel. When the train enters the track, the semaphore changes state to prevent other trains from entering the track. A train that is leaving this section of track must again change the state of the semaphore to allow another train to enter. In the computer version, a semaphore appears to be a simple integer. A process (or a thread) waits for permission to proceed by waiting for the integer to become 0, and in order to gain control of the resource, the process first increments the semaphores by 1. When it is finished, the process changes the semaphore’s value by subtracting 1 from it. Although semaphores represent one of the fastest means to facilitate interprocess communication, its use is restricted for processes running on a single machine. For this particular project, we did not use semaphores to handle the inter-agent communication because the agents developed in this study interact with one another across multiple computers.

### 2.2.4 Message queues.

According to Marshall (1999) [15], two or more processes can exchange information via access to a common system message queue. The sending process places via some operating system message-passing module a message onto a queue which can be read by another process. Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place (subject to other permissions). When a message is sent, its text is copied to the message queue. The sending and receiving of the message can be performed either as blocking or non-blocking operations. Non-blocking operations allow for asynchronous message transfer – the process is not suspended
as a result of sending or receiving a message. In blocking or synchronous message passing, the sending process cannot continue until the message has been transferred or has even been acknowledged by the recipient. IPC signal and other mechanisms can be employed to implement such transfer. A blocked message operation remains suspended until when the call succeeds or the process receives a signal or when the queue is removed.

2.2.5 Named pipes

This technique allows programs on different computers to establish a reliable 2-way communications link (Russinovich, 1999)[18]. A named pipe server calls a low level networking API function to define a pipe on the local computer, and a named pipe client opens a connection to the server’s pipe by specifying the pipe’s name using the same API function. With the established connection, the client and server can then use the low level read and write standard I/O routines, to exchange data through the pipe connection. Named pipes are not implemented in Windows 98 and Windows 95, but only in later version of Windows operating systems. Due to this limitation, it was not suitable for the agents developed in this project.

2.2.6 Remote Procedure Calls (RPC)

Programmers developing an RPC client/server application define functions that the server implements (Russinovich, 1999)[18]. The developer writes the client program as if the program can invoke the server functions directly. What happens under the hood, however, is that a client-side library takes the parameters specified by the client functions into a message. This message is transmitted to the server, which also uses a server-side library to unpack the message and pass them to the server function. When the server function finishes, any return parameters go through the same process as the server sends them back to the client. The fact that the parameter packing and unpacking (called marshalling and unmarshalling) and the message transmission are hidden from the application developer makes RPC an attractive API. RPC in
Windows is really a programming API layer that rides on top of named pipes, Winsock or Message Queues. The Microsoft Interface Definition Language (MIDL) Compiler automatically builds client and server libraries for a developer, based on the function definitions the developer provides.

### 2.2.7 Mail slots

Mail slots are part of the Windows IPC services. They allow one computer to send messages to another or to broadcast messages to several other computers (Russinovich, 1999)[18]. Messages smaller than 425 bytes are sent as datagrams and can be broadcasted. The ability to broadcast a message over the network to any number of computers is very desirable in many cases. Using datagrams means that no open connection is needed to the remote computer(s). The sequence for setting up a Mail Slot is easy. One computer creates the Mail Slot, and can only read data. A remote computer can open the Mail Slot and write data. Mail Slots are identified by names. You also supply a computer name for the target computer - or use the * notation to broadcast to all Mail Slots with a given name, on any computer. Since Mailslots is implemented only in MS Windows, it was found unsuitable for the agents developed in this study.

### 2.2.8 Memory mapped files

Memory mapped files is used as a way of accessing a large file, without necessarily loading the file into memory (Marshall, 1999)[15]. By using file mapping, a whole file can be thought of as already loaded into memory and accessed through a memory pointer. File mapping is also used as a means to share data between processes. Using file mapping in this way, there is no actual file involved. It is more like a reserved memory block that every process can see. Memory mapped files have to implement processes and thread synchronization or else the implementing application will crash in a very short time. Once a memory mapped file is created, its size cannot be changed during that session. So, file mapping is great for read-only files or file operations that
do not affect the file size. This technique was unsuitable for agent communication since agents do not need to be constraint by current size status of its accessed file.

2.3 Internet Technology

This project outputs are a collection of Internet based client-server tools, called agents. The agents interact predominantly in the Internet domain; example using web servers, web browsers and e-mail client software. These technologies were studied, and the researcher thus gained better understanding of the problem domain, that enabled proper conceptual understanding of agents interactions. This was done through literature review on the following internet technologies:

2.3.1 Simple Mail Transfer Protocol (SMTP)

SMTP is the main protocol used to send e-mail on the Internet. It consists of a set of rules for how a program sending e-mail and a program receiving e-mail should interact. The SMTP design is based on the following model of communication (Postel, 1982)[16]: A user SMTP program requests the server for the user's e-mail by establishing a two-way transmission channel to the server-SMTP. The server-SMTP may be either the ultimate destination or an intermediate. SMTP commands are generated by the user-SMTP and sent to the server-SMTP. SMTP replies are sent from the server-SMTP to the user-SMTP in response to the commands. The responses are pre-defined by the SMTP Protocol, thus easing development of SMTP client applications. Figure 2.1 illustrates a typical SMTP client-server interaction.

11
2.3.2 Post Office Protocol (POP)

POP is a protocol that allows many e-mail programs to retrieve messages from a mail server. E-mail is usually delivered on the Internet to a mail server, and an e-mail program running on a personal computer retrieves that e-mail from the server using POP. POP is one of the most simplified protocol used for accessing e-mail boxes based on the download-and-delete model (Gellens et al., 1998)[10]. Various extensions to the protocol exist through which e-mail administrators can set policy decisions: The TOP capability indicates that the optional TOP command is available, The USER capability indicates that the USER and PASS commands are supported (although they may not be available to all users), The SASL capability indicates that the POP server supports the SASL authentication mechanism, The RESP-CODES capability is used to identity the server’s extended response codes, The LOGIN-DELAY capability determines how often the server accepts e-mail request from a given client, the PIPELINING capability determines whether the server is capable of accepting multiple commands at a time, the EXPIRE capability allows the server to inform the client as to the policy in effect. The argument to the EXPIRE capability indicates the minimum server retention period, in days, for messages on the server. EXPIRE 0, for example indicates that the client is not permitted to leave e-mail on the server.
A POP mail-box can be treated as a network resource accessible using Uniform Re-
source Locator (URL) notation (Gellens, 1998)[9]. By being able to specify a POP
mailbox as a URL allows for flexibility and ability to tune many programs and pro-
tocols to gain access to a POP mailbox.

2.3.3 Internet Mail Access Protocol (IMAP)

IMAP is a protocol allowing a client to access and manipulate e-mail messages on a
server. It permits manipulation of remote message folders (mailboxes), in a way that
is functionally equivalent to local mailboxes. IMAP includes operations for creating,
deleting, and renaming mailboxes; checking for new messages; permanently removing
messages; searching; and selective fetching of message attributes, texts and portions
thereof (Chrispin, 1994)[7]. IMAP is designed to transmit message data on demand,
and to provide the facilities necessary for a client to decide what data it needs at any
particular time. With IMAP, there is generally no need to do a wholesale transfer of
an entire mailbox or even of the complete text of a message. This makes a difference
in situations where the mailbox is large, or when the link to the server is slow. For
example, a user connected to an IMAP server via a dialup link can determine that a
message has a 2000 byte text segment and a 40 megabyte video segment, and select
to fetch only the text segment.

2.3.4 E-mail Models

E-mails are messages passed from one computer user to another through computer
networks. There are three fundamental models of client/server e-mail (Crispin, 1994)[7]:
offline, online, and disconnected use.

Offline model is the most familiar form of client/server e-mail and is used by proto-
cols such as POP. In this model, a client application periodically connects to a server,
downloads all the pending messages to the client machine and deletes them from the
server (depending on the client’s settings). Thereafter, all e-mail processing is local
to the client. This model is store-and-forward; it moves e-mail on demand from an
intermediate server to a single destination machine.

**Online model** is most commonly used with remote filesystem protocols. In this
model, a client application manipulates mailbox data on a server machine. A connec-
tion to the server is maintained throughout the session. No mailbox data are kept on
the client; the client retrieves data from the server as is needed. A good example is
IMAP.

**Disconnected** model is a hybrid of the offline and online models. In this model,
a client downloads some set of messages from the server, manipulates them offline,
then at some later time uploads the changes. The server remains the authoritative
repository of the messages. The problems of synchronization (particularly when mul-
tiple clients are involved) are handled through the means of unique identifiers for each
message.

### 2.3.5 Internet message format - Message Headers

Request for Comments (RFC822) defines a general ”memo” framework for syntax of
how message headers are formatted. According to Crocker (1982)[8], the syntax that
distinguishes between header fields is specified separately from the internal syntax
for particular fields. This separation is intended to allow simple parsers to operate
on the general structure of messages, without concern for the detailed structure of
individual header fields. Example of syntax rules are given as:

- **RULE1 ”/” RULE2: ALTERNATIVES**: Elements separated by slash (”/”) are
  alternatives. Therefore ”foo / bar” will accept foo or bar;

- **(RULE1 RULE2): LOCAL ALTERNATIVES**: Elements enclosed in parenthe-
  ses are treated as a single element. Thus, ”(elem (foo / bar) elem)” allows the
token sequences ”elem foo elem” and ”elem bar elem”;

14
• **RULE: REPETITION**: The character "*" preceding an element indicates repetition. The full form is: \( j_{l}^{*} m_{r} \text{element} \) indicating at least \( j_{l} \) and at most \( m_{r} \) occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows one or two;

• **[RULE]: OPTIONAL**: Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)";

• **NRULE: SPECIFIC REPETITION**: "\( n_{l} \text{(element)} \)" is equivalent to "\( n_{l}^{*} n_{m} \text{(element)} \)"; that is, exactly \( n_{l} \) occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters;

• **#RULE: LISTS**: A construct "#" is defined, similar to "*", as follows: \( j_{l}^{#} j_{m} \text{element} \).

### 2.3.6 Internet message format - Message body

Bellcore (1993)[3] defines, in RFC 1521, the format of message bodies to allow multi-part textual and non-textual messages to be represented and exchanged without loss of information. Multiple objects can thus be included in a single message to represent body text messages in multiple character sets, multi-fonts and non-textual material such as images, audio and video fragments. The message type is specified in the content header field of each e-mail using type and subtype tags to specify the type of data in the body of a message and contains enough information to make it easy for the receiving user agent to pick an appropriate agent or mechanism to present the data to the user and how to deal with the data.
2.3.7 HyperText Transfer Protocol (HTTP)

HTTP is the language used by web browsers, web servers and all related web applications to talk to each other (Gourley and Totty, 2002)[12]. HTTP communication or transaction consists of a request command sent from the client to the server, and a response result sent from the server to the client. The communication happens with formatted blocks of data called *HTTP messages*. The formats of HTTP request and response messages are similar and contained three blocks: *start line* indicating what to do for a request or what happened for a response; *Header fields* consisting of a named and a value, separated by colon; *Body* containing any kind of data. Request body carry data to the web server, while response body carry data back to the client. Each HTTP request message must have a method. The method tells the server what action to perform. Example of common HTTP methods are:-

- **GET** Send the named resource from the server to the client.
- **PUT** Store data from the client into a named server resource.
- **DELETE** Delete the named resource from the server.
- **POST** Send client data into a server gateway application.
- **HEAD** Send just the HTTP headers from the response for the named resource.

Because HTTP is very modular and simple, it was quiet easy to develop software agents that understood the HTTP language and thus could provide web service to its users.
2.3.8 Web Servers

Web contents/resources live on web servers. A web resource is the source of web content. An example is a static file on the web server’s filesystem. This file can contain anything: text, HTML, Microsoft Word, Adobe Acrobat, images, movies or any other format you can think of. Web servers speak the HTTP protocol, so they are often called HTTP Servers (Gourley and Totty, 2002)[12]. The primary business of a web server is to translate a request message either into a filename, and then send that file back over the Internet, or into a program name, and then run that program and send its output back (Laurie and Laurie, 2002)[14]. Of the over 75 available web servers in the market, Apache is the most popular and demands the highest market share of over 60%, mainly because it is free and open source, but more importantly, Apache is extensible through an established API, which many people have used to introduce new features.

2.4 Agent related studies

2.4.1 Kasbah

One of the early marketplaces is Kasbah (Chavez and Maes, 2001)[5]. Kasbah is a Web-based system where users create autonomous agents that buy and sell goods or services on their behalf. Selling agents try to sell themselves by going into the marketplace contacting interested buying agents and negotiating with them to find the best deal. The job of the marketplace is to facilitate interaction between the agents. In Kasbah, a specific communication language is used since their agents are locally built and can communicate via a predefined set of methods. Kasbah was designed to specifically assist users in negotiation, which makes Kabash quiet different from the agents developed in this study.
2.4.2 MAGMA

The Minnesota AGent Marketplace Architecture (MAGMA) (Gini, et al., 1997)[11] provides for marketplace architecture that includes the infrastructure required for conducting commerce on the Internet. MAGMA supports communication among agents and allows for various forms of automated and human-controlled transactions. There are several trader agents, an Advertising server and a Bank in MAGMA. Trader agents are responsible for buying and selling goods. The Advertisement server provides a classified advertisement that includes search and retrieval of ads by category. Finally, the Bank provides a set of basic banking services such as checking accounts and electronic cash. All agents communicate with each other through sockets.

2.4.3 OFFER

OFFER (Bichler and Segev, 1999)[2] is an electronic brokering architecture which uses Object Management Group (OMG)’s Common Object Request Broker Architecture (CORBA) as a distribution infrastructure. There are three main components: suppliers, customers and e-brokers. A customer can search for a service either directly in the e-catalog of the supplier or use the e-broker to search all the e-catalogs of all the suppliers, which are registered with this broker. CORBA is chosen as the communication infrastructure to solve the interoperability problem. As the negotiation mechanism e-brokers employ simple auction implementations.

2.4.4 EMP

EMP (Boll et al., 1999)[4] is a marketplace that has a Database Management System (DBMS) based architecture. Business transactions within the electronic market are realized by a set of modular market services like offering, buying, registration and authentication. Product data are stored in a DBMS and accessed by the market server through Java Database Connectivity (JDBC) interface.
2.4.5 Agent-based workflow

Arpinar et al., (1998)[1] modeled the commerce processes in the marketplace as adaptable agent-based workflows. The design provided by the workflow technology makes the customization of electronic commerce processes for different users possible. Agent-based implementation, on the other hand, provides for highly reusable component-based workflow architecture as well as negotiation ability and the capability to adapt to dynamic changes in the environment. Agents communication is handled through Knowledge Query and Manipulation Language (KQML). A workflow-based architecture also makes it possible for complete modeling of electronic commerce processes by allowing involved parties to be able to invoke already existing applications or to define new tasks and to restructure the control and data flow among the tasks to create custom built process definitions.

2.5 E-Commerce Models

In the most basic sense, a business model is the method of doing business by which a company can sustain itself – that is, generate revenue (Rappa, 2006)[17]. The business model spells-out how a company makes money by specifying where it is positioned in the value chain. Different types of e-commerce models exist:-

**Brokerage Model**

Brokers are market-makers: they bring buyers and sellers together and facilitate transactions. Brokers play a frequent role in business-to-business (B2B), business-to-consumer (B2C) or, consumer-to-consumer (C2C) markets. Usually a broker charges a fee or commission for each transaction it enables.

**Advertising Model**

The web advertising model is an extension of the traditional media broadcast model. The broadcaster, in this case, a web site, provides content (usually, but not necessarily, for free) and services (like email) mixed with advertising messages in the form of banner
ads. The banner ads may be the major or sole source of revenue for the broadcaster. The broadcaster may be a content creator or a distributor of content created elsewhere. The advertising model works best when the volume of viewer traffic is large or highly specialized.

**Infomediary Model**
Data about consumers and their consumption habits are valuable, especially when that information is carefully analyzed and used to target marketing campaigns. Independently collected data about producers and their products are useful to consumers when considering a purchase. Some firms function as information intermediaries assisting buyers and/or sellers understand a given market.

**Merchant Model**
Wholesalers and retailers of goods and services. Sales may be made based on list prices or through auction.

**Manufacturer Model**
With this model, a manufacturer reaches buyers directly and thereby compress the distribution channel. The manufacturer model can be based on efficiency, improved customer service, and a better understanding of customer preferences.

**Affiliate Model**
The affiliate model provides purchase opportunities wherever people may be surfing. It does this by offering financial incentives to affiliated partner sites. The affiliates provide purchase-point click-through to the merchant. It is a pay-for-performance model. If an affiliate does not generate sales, it represents no cost to the merchant.
The affiliate model is inherently well-suited to the web, which explains its popularity. Variations include, banner exchange, pay-per-click, and revenue sharing programs.

**Community Model**
The viability of the community model is based on user loyalty. Users have a high investment in both time and emotion. Revenue can be based on the sale of ancillary products and services or voluntary contributions; or revenue may be tied to contextual advertising and
subscriptions for premium services.

**Subscription Model** Users are charged a periodic – daily, monthly or annual – fee to subscribe to a service.

**Utility Model** The utility or ”on-demand” model is based on metering usage, or a ”pay as you go” approach. Unlike subscriber services, metered services are based on actual usage rates.

Chen *et al.* (2001)[6] studied the emerging models of e-commerce Web site design and categorized them based on two Web site design strategies: *information/communication* design strategy and *on-line/transaction* design strategy.

**Information/Communication** This strategy mainly focuses on using the web to market an organization’s businesses, with no or little support for transactions. Web site design models under this category are:- brand awareness and image building, cost saving, promotion and info-mediary.

**on-line/transaction** This strategy invariably provides an electronic catalog of product for sale. Visitors can browse through the catalog and order products on-line. Design models falling under this category include:- brokerage, retail, mall, advertising, subscription, community, manufacturer and customization.

The study by both Rappa and Chen suggested that businesses tend to use a combination of e-commerce models for their Web sites. In this research, the on-line/transaction strategy was used and the specific models used were: brokerage and merchant models.
Chapter 3

Methodology

3.1 Introduction

Unified Modeling Language (UML) is a language that provides a convenient way to model software systems so that the finished product is of high quality (Jason, 2003)[13]. UML is just a notation language and not a way to design systems. To use UML, an appropriate design method should be applied to it. Rotational Unified Process (RUP) is the most common design method in use today. RUP supports an iterative approach to development that addresses the highest risk items at every stage in the life cycle, significantly reducing a project’s risk profile. Due to sophistication of agents technology, it is not possible to sequentially first define the entire problem, design the entire solution, build the software and then test the product at the end. An iterative approach is required that allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution over multiple iterations. This study used UML as a modeling language, and RUP as a software engineering methodology in trying to achieve the set objectives. RUP is a four phased design processes: inception, elaboration, construction and transition phases. Only the first three phases were used due to the academic nature of this study.
3.2 The inception phase

Jason (2003)[13] defines the inception phase of RUP as the stage where initial system analysis is performed to identify the business requirements. The requirements are refined and modeled into use case diagrams. In this study, the output of the inception phase were:-

(i) A vision document: a general vision of the core project’s requirements, key features, and main constraints;

(ii) An initial use-case model, which was about 10%-20% complete.

3.2.1 Review of existing online stores

A preliminary study on some existing online stores were carried out to gain an initial idea on their operations. In each store, the researcher identified techniques used to specify, catalogue and classify automobile products. The online stores studied were:-

- www.ebay.com;
- www.cars.co.ug;

3.3 The Elaboration phase

The design of the various agents were done at the elaboration phase. The Use cases discovered during the inception phase were iteratively modeled into design documents for the various agents, their subsystems and related business objects. Class diagrams were used to represent the different underlying pieces, their relationship to one another and which subsystems they belong to. This effort was to address the critical use cases identified in the inception phase, and that typically exposes the major technical risks of the project. One exploratory, throw-away agent prototype was developed to
mitigate specific risks such as design/requirements trade-offs. The following outputs were achieved at the end of this phase.

(i) A use-case model (at least 80% complete) - all use cases and actors have been identified, and most use-case descriptions have been developed;

(ii) Supplementary requirements capturing the non functional requirements and any requirements that were not associated with a specific use case;

(iii) An executable architectural prototype.

3.4 The construction phase

It was at this stage that we built the final product from the design of the system. The development work was done in incremental iterative process, in portion that were manageable.

3.5 Tools

The following tools were used in the study:-

- Microsoft Visio for developing use cases;

- Java, Perl and PHP programming languages for writing the program codes;

- JBuilder, an Integrated Java Development Environment;

- Latex for all documentation, including the final project report.
Chapter 4

The Inception Phase

4.1 Introduction

In this chapter, systems analysis was conducted to identify the initial business requirements. Three existing online stores were studied, and their relevant processes documented. The first requirement identified was the domain model, which described "what was the business problem". By defining the domain model, the researcher was able to effectively communicate the business problem. This also provided an initial starting point for the Business and Use Case models for the final product.

4.2 Existing Online Stores

4.2.1 Cars Uganda

Cars Uganda (http://www.cars.co.ug) is an Internet-based portal operated in Uganda, and provides avenues for car sellers to upload their vehicles details for online viewing by potential buyers. The site provides facility for sellers to upload pictures and details of the vehicles such as make, condition of the vehicle, type, location, model, price, full description, gear box type, fuel type, year of manufacture, chassis, doors, color, registration number, sales status, cubic capacity, mileage, extras, other features and government requirements. Sellers are however, required to sign-up to the site as a member before they can be able to upload their vehicle details. Sellers are charged
25,000 Uganda shillings for each vehicle displayed on the site.
Cars Uganda, is not itself a car-selling firm, but only helps to link sellers to potential buyers. This is a many-to-many models, where there are multiple sellers and multiple buyers. When a buyer identifies a matching vehicle, he/she contacts the car seller directly and they enter into direct negotiations. If the two parties come to an agreement and the buyer buys off the vehicle, the seller must adjust the vehicle status indicating that it is already sold.

4.2.2 Trust Company Limited (TCL)

TCL is a Japan (Nagoya) based online car exporting company. The company buys cheap second hand vehicles from auctions and dealers in Japan and exports them to many third world countries, including Uganda. A complete listing of all the vehicles available for sale are browsable from the company’s online portal at http://www.japanesevehicles.com.

The online marketplace is based on one-to-many model, with only one seller (TCL) and multiple buyers.

The procedure for buying vehicles from TCL is simple:-

1. A buyer identifies a vehicle of his/her choice. This is done by browsing the product lists in two different ways:-
   - Customer navigates through the catalog, by vehicle category, then by products within each category;
   - Customer searches for required vehicle using keyword search. The search tool allows users to specify what vehicle type, year of manufacture or model they require.

2. The buyer then completes an Order Form;

3. TCL sends an invoice to the buyer either by fax and/or e-mail;
4. The buyer makes payment on the invoice and faxes a copy of the bank receipt along with the invoice back to TCL;

5. Shipping confirmation is sent to the buyer with the details of the ship, port of destination and expected arrival date;

6. The vehicle is shipped and shipping documents sent to the buyer via courier to the address supplied on the original order form.

4.2.3 eBay

eBay (http://www.ebay.com) is The World’s Online Marketplace for the sale of goods and services by a diverse community of individuals and small businesses. Unlike Cars Uganda and TCL which specialize in selling only motor vehicles, eBay is a true market and offers a variety of items in thousands of categories from collectibles like trading cards, antiques, dolls, and housewares to practical items like used cars, clothing, books, CDs, and electronics. Buyers have the option to purchase items in an auction-style format or items can be purchased at a fixed price through a feature called Buy It Now.

The selling process has three steps:

1. Prepare - eBay provides a tool, which allows potential buyers to first research similar items that have already been sold successfully on eBay. The tool enables the buyer to find useful information on pricing, features, and key "buzz words" that attract buyers;

2. List - To be able to sell a product on eBay, buyers must list their products. eBay provides a set of tools that helps users to list their products in a professional manner. Listing a given product is done in six steps:-
   
   - Choose a format: Once you choose a format, you’re ready to begin;
• Select a category: The suggested categories tool can help you choose the best category for your item. You can also look at other items currently on eBay to find out which categories other sellers chose for their items;

• Title and description: Be clear and descriptive to create interest. Remember that buyers search for items based upon the words in the item titles, so be sure to include words that buyers would search for. Make sure to include the condition of your item and any other details that a buyer should know;

• Picture and details: Choose the price at which you would like to start your item. eBay Picture Services then lets you add pictures right from the selling form. Adding pictures to your listings will help buyers know what they are bidding on and remember, the first picture you add is free! In order to make your items stand out on eBay, you can also choose to add listing features like Gallery or Bold;

• Payment and shipping: PayPal is the fast, easy and secure way to accept payments. Include a flat fee for shipping and handling costs or add a shipping calculator based on your buyer’s mailing address.

• Review and submit: Make sure that you have checked all spelling and details of your item and you are ready to submit your listing.

3. Get Paid - After your listing ends successfully, you will receive payment from your buyer.
4.3 Domain model context diagram for the automobile store

Figure 4.1 illustrates the contextual model for the online automobile store, here referred as eStore, developed in this project.

![Diagram](image)

Figure 4.1: Domain Model context diagram

4.3.1 Domain actors

This section describes each of the actors participating in the domain model context diagram, and their roles in the business domain.

<table>
<thead>
<tr>
<th>Actor</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>A customer is a person who searches the online store for automobiles</td>
</tr>
</tbody>
</table>
4.3.2 Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC-001000</td>
<td>customer identifies automobile</td>
</tr>
<tr>
<td>UC-002000</td>
<td>customer specifies product</td>
</tr>
<tr>
<td>UC-003000</td>
<td>customer maintains account</td>
</tr>
</tbody>
</table>

Description of Use Cases

*Customer identifies automobile (UC-001000):* This Use case represents sets of activities carried out by the main actor (Customer) in identifying matching products from the online market. The customer goes through the following steps:-

- Customer logs onto the eStore and sets search criteria;

- Customer scrutinizes the automobile specifications sent to his/her e-mail address;

- Customer extracts the contacts details of the automobile seller (part of the specification sent to his/her e-mail) and contacts the seller to negotiate the transaction.

*Customer specifies product (UC-002000):* This Use Case represents the major set of activities carried out by the customer when logged onto the eStore. The process flow for this Use case is presented in Figure 4.2.
To specify the product, a customer sets values for the following variables to his or her preference: make, transmission type, model, price range, number of doors, cubic capacity and mileage.

Customer maintains account (UC-003000): Customer uses eStore to create a new account for his/her self. The customer’s e-mail ID is used to uniquely identify the customer. Once authenticated, the customer may update her account information.
4.3.3 Use Case Diagram

Figure 4.3 below presents Use Case diagram discovered in this project.

![Use Case Diagram](image)

Figure 4.3: UC-003000 Use Case Diagram

4.4 Detailed Requirements Specification

The Detailed Requirements Specification contains the functional features of the system. As such, it is structured and used as a reference manual. The requirements are grouped into two major categories - functional requirements and proficiency requirements. Functional requirements are written in terms of - “the system shall do X”. Proficiency requirements specify ”how well the system does function X”. Each requirement is assigned a unique identifier, using the following naming convention:

- **DR-ss-nn-mmm**: Data formatting requirement
- **BR-ss-nn-mmm**: Business rule requirement
- **FR-ss-nn-mmm**: Functional requirement
**IR-ss-nn-mmm**  Interface requirement

**PR-ss-nn-mmm**  Proficiency requirement (eg, security, performance, etc.)

Where:

- \( ss \), the section number in which the requirement appears;
- \( nn \), a sequential numbering of the requirements within the section;
- \( mmm \), inserts requirements without forcing a renumbering of any adjacent requirements.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR-40-001-000</td>
<td>A customer account is created only if the e-mail address supplied is valid and exists.</td>
</tr>
<tr>
<td>FR-41-001-000</td>
<td>All required fields are indicated as such with a '*' next to the label.</td>
</tr>
<tr>
<td>FR-41-002-000</td>
<td>The eStore catalog is administered from the backend using SQL Scripts. Administering of all reference data: make, type, model, price,</td>
</tr>
<tr>
<td></td>
<td>gear box type, fuel type, government condition is done from the backend using SQL scripts. No user interface required.</td>
</tr>
<tr>
<td>FR-41-004-000</td>
<td>Backups are to be performed while the system is operational.</td>
</tr>
<tr>
<td>FR-41-005-000</td>
<td>Due to relatively small amount of data at startup, archival of data shall not be supported.</td>
</tr>
<tr>
<td>PR-42-001-000</td>
<td>The system must uniquely identify a user with a User ID/password combination before accepting search request or updating an account. The UserID must be the person’s e-mail address.</td>
</tr>
<tr>
<td>PR-42-002-000</td>
<td>Once the user has logged in, he/she will be automatically logged out if the session is idle for 10 minutes. Subsequent requests to protected pages will force re-authentication.</td>
</tr>
<tr>
<td>PR-42-003-000</td>
<td>A user may only view and update the account in which they are the owner.</td>
</tr>
<tr>
<td>PR-42-004-000</td>
<td>Passwords must be encrypted when stored on disk.</td>
</tr>
<tr>
<td>PR-42-005-000</td>
<td>The system must not be affected by the following list of exploits: URL Changing, Hidden Field Hacks, Spoofing.</td>
</tr>
<tr>
<td>PR-42-006-000</td>
<td>eStore must audit the last date/time when any data is added, modified, or deleted and track the last user ID that added, modified, or deleted the data.</td>
</tr>
<tr>
<td>PR-42-007-000</td>
<td>eStore must support UK English and UK date and time formats.</td>
</tr>
<tr>
<td>PR-42-008-000</td>
<td>All functionality of the system shall work with the following web browser platforms: Internet Explorer 5.0 SP1 or greater, Netscape Navigator 6.0 or greater.</td>
</tr>
</tbody>
</table>
Chapter 5

The Elaboration Phase

5.1 Introduction

This chapter captures all the internal decisions required to implement the eStore system. The approach used was to examine the system from six different perspectives, each providing a different "view". This was intended to provide the researcher with ability to capture all critical system features. A brief description of the six architecture views is provided as follows:

- **Deployment View**: Documents the physical topology of the system, including the software and hardware requirement for each component.
- **Logical View**: Documents the design model, which defines the layers of the eStore application and the primary classes within each layer.
- **Data View**: Describes the database schema and describes how the classes in the logical view are mapped onto the database structures on disk.
- **Process View**: Documents and describes how independent threads of execution within the eStore application communicate.
- **Implementation**: This view maps the classes in the Logical View to JAVA and/or PHP and/or PERL and/or HTML source files and combines the files into deployable components. The view also tracks the dependencies among the components.
- **Security View**: Focuses on how the system identifies end users, how authorization
is granted to end users based on their identity, and how integrity of the system and data are enforced by properly tracking and auditing of system activities.

5.2 Deployment view

Figure 5.1 below shows the physical configuration of the eStore system.

Figure 5.1: eStore deployment configuration

5.2.1 Description of each component

The eStore system requires web and database services to be running before it can be deployed. This section describes the constituent elements for each of the component specified in Figure 5.1 above.
eStore Database Server

The requirement for the eStore database server are:–

- Operating system - Linux, Unixware;
- DBMS software - MySQL.

eStore Web Server

The requirement for the eStore web server:-

- Operating system - Linux;
- Apache Web server;
- PHP software libraries;
- PERL software libraries;
- eStore Agent software libraries.

SMTP Server

The eStore application utilizes existing SMTP servers to deliver search results to the customer’s e-mail address. The application works with any RFC821 compliant SMTP server.

Client/Customer Workstation

Client workstations must meet the following minimum requirements:-

- Web browsers - Internet Explorer (Version 5 and above), Netscape communicator, Firefox, Opera.
5.3 Logical View

Figure 5.2 below shows the logical configuration of the eStore system. Its a 3-tier architecture, comprising the User-Interface (UI) layer, the Business Layer (BL) and the Data Layer (DL).

![Logical Configuration Diagram]

Figure 5.2: eStore Logical configuration

The SMTP and Web Server are standard packages that come with most operating systems. The specific details of the requirement for this research are already detailed in Section 5.2 above.

ACC Manager

The Account Manager (ACC) is the first of three main libraries for the eStore system. This library provides all of the facilitates for the end user to maintain/manage their eStore accounts. It was implemented as a server background process, listening on port 9090. The following services are provided:

- Account Creation: Provides the necessary libraries and interface for self-registration to the eStore portal. Users must be in possession of a valid e-mail account;
• View Account: Provides the necessary libraries and interface that allows users to view their account information;

• Update Account: Provides the necessary libraries and interface that allows users to perform updates on their individual eStore accounts depending on the authorization level;

• Delete Account: Provides the necessary libraries and interface that allows users to remove themselves from the eStore system.

The ACC Manager provide the above services based on the request sent to it by a corresponding PHP-based client software libraries. The client software gathers the customer’s form input and formats them into a message that the ACC Manager understands and is able to process. This is sent to the ACC Manager for processing.

**Dispatch Agent**

The Dispatch Agent is the second of the three main software libraries for the eStore system. It runs as a server background process. Every so often, it wakes up and traverses the database for entries that should be dispatched to customers. It then formats and sends them as smtp messages, including a picture of the matched vehicle, as an attachment. It provides the following services:-

• e-mail formatting: Before sending an e-mail to a given customer’s address, the Dispatch Agent ensures that the message is formatted in a way consistent with the requirements of the Mail Transfer Agent (MTA);

• sending e-mail: The dispatch agent implements the client-side SMTP protocol and communicates directly with a SMTP server when sending e-mail messages.
Search Agent

The Search Agent is the third of the three main software libraries for the eStore system. It was implemented as a background server process listening on port 8080. It provides the following services:

- Receives Requests: it receives and records search requests from customers, for specific automobiles;

- Performs search: Searches the eStore database for automobiles matching the customer’s specification.

The search agent does two things: firstly, It receives messages from a PHP-based client routine detailing search criteria that was set up by an eStore customer. These messages are specially formatted so that the Search Agent is able to differentiate fields boundaries and thus can extract individual fields for further processing. The Search Agent saves the criteria into the Request table in the eStore database. Secondly, every so often, the Search Agent searches the automobile table using criterion got from the request table. Every match made is recorded in the match table. To implement this scenario, there was need to create two independent threads or processes, one handling recording of request and the other handling searching of the eStore domain for matching automobile.
5.3.1 The eStore Classes

The main classes for the eStore systems are presented in the class diagrams presented in Figure 5.3 below:-

![Figure 5.3: eStore Class diagram]
5.4 The data view

This section gives a description of the database schema and elaborates how the classes defined above are mapped onto the database structures on disk. This is presented in Figure 5.4 below.

Figure 5.4: eStore database schema diagram
5.4.1 The ACCOUNT Class and the ACCOUNT RELATION

The ACCOUNT relation is a physical schema which supports the ACCOUNT class, by providing a persistent way of storing results of its operations. The class offers methods such as NEW_ACC(), READ_ACC(), UPDATE_ACC() and DELETE_ACC() that are used by the ACC Manager sub-system to create, display, update and delete user’s accounts information. Each of these methods whenever executed will result into changes to the physical objects stored in the ACCOUNT relation. For example, the NEW_ACC() when executed successfully will create a tuple in the ACCOUNT relation.

5.4.2 The WEBSITE Class and the REQUEST RELATION

The REQUEST relation is a physical schema which supports the WEBSITE class, by providing a persistent way of storing results of its operations. The class offers two methods: LOGON() that is used by the customer to gain access to the eStore portal, and SET_CRITERIA() which the logged on user evokes in order to set his/her search criteria. All that the REQUEST relation is required to do is to record the search criteria set up by the user. The criteria is a string representation of SQL statements generated dynamically as the user sets his/her vehicle preference using the provided interface.

5.4.3 The MATCH Class and the MATCH RELATION

The MATCH relation is a physical schema which supports both the MATCH and the REQUEST classes. It stores physical and persistent objects that are regularly accessed by these two classes. The REQUEST class upon a successful execution of the SEARCH() method, records the vehicle registration number and e-mail address of the requester, in the MATCH relation. The MATCH class also uses the relation in conjunction with its FORMAT() and SEND() methods to format and dispatch
via e-mail, vehicles which have matched the set criteria and whose SENT property is FALSE.

5.4.4 The AUTOMOBILE Class and the AUTOMOBILE RELATION

The AUTOMOBILE relation is a physical schema which supports three classes: The AUTOMOBILE, MATCH and REQUEST classes. It stores physical and persistent objects that are regularly accessed by these classes. The FINDBYCRITERIA() method of the AUTOMOBILE class iterates through tuples in the relation to find vehicles matching the set criteria. The way in which the FINDBYCRITERIA() method was used in the research was to evoke it as part of the SEARCH() method of the REQUEST class.
Chapter 6
The Construction Phase

6.1 Introduction

This chapter details the techniques used to implement the eStore design described in the previous chapters. This includes the various screens of the User Interface Layer; the code snippets for the business layer; and sub-routines for creating the database objects.

6.2 Implementing the User Interface Layer

The user interface layer was implemented in PHP language using three internet based forms:-

- The Log In Form - index.php;
- The Search Criteria Screen - search.php;
- The Account Management Screen - account.php.
6.2.1 The Log In Form - index.php

The Log In form is presented diagrammatically below:

![EStore Logon Screen](image)

**Figure 6.1: eStore Logon Screen**

**Implementation snapshot**

There are two scenarios that have been implemented by this Log In form (index.php):

- **Scenario #1**: when a new eStore customer has to create an eStore account;
- **Scenario #2**: when an eStore customer with a valid account logs on.

The first scenario is pretty straightforward: The *index.php* form just passes control to the *account.php* form. The detailed implementation of the *account.php* was discussed in Sections 6.2.3 of this manuscript. The results of the operation performed
The second scenario was implemented as follows:-

- `index.php` passes control to `process.php` (Appendix A-I) and instructs it to log in the given user;

- `process.php` connects to the database and validates the user’s credentials as provided. If all is fine, the user is logged in and `process.php` transfers control to `search.php`, otherwise, control is given back to `index.php` and the user is asked to try again.

**Results**

Results for the two scenarios are presented below:-

- **Scenario #1**: when the user presses the `sign up` link, the outcome is the `account.php` form presented in Figure 6.3, and contains account creation details for user gkomakech@dicts.mak.ac.ug;

- **Scenario #2**: whenever a customer successfully logs in - by pressing the `Sign In` button, the outcome is the `search.php` form which is presented in Figure 6.2 and contains criterion specified by user gkomakech@dicts.mak.ac.ug for Toyoto Corolla, automatic transmission, mileage limit of 80.000, 4 doors, 18CC, price range of 8-10 million and model 1998. The outcome when the user presses the `Submit` button was discussed in Section 6.2.2.
6.2.2 The search form - search.php

The `search.php` form is presented diagrammatically below:

![Figure 6.2: eStore search criteria Screen](image_url)

**Implementation snapshot**

There are two scenarios that have been implemented by the `search.php` form:

- Scenario #1: when a user clicks on the *go to My Account* link to update his/her account details;

- Scenario #2: when an eStore customer sets search criteria for the kind of automobile he or she wants.

The first scenario is pretty straightforward: *Search.php* passes control to the `account.php` form and instructs it to retrieve and display accounts details for the user.
logged in. The outcome with user gkomakech@dicts.mak.ac.ug logged in was presented in the Results Section on this page.

The second scenario was implemented as follows:

- *Search.php* performs form level validation, to check all input fields entered by the customer. For example, enforcing numerical entries for the Mileage, Price Range, number of doors, CC and Model fields;

- *search.php* passes control to *process.php* (Appendix A-I) and instructs it to process the user’s input;

- *process.php* transforms the user’s input into valid SQL statements, formats it and sends the message to the Search Agent (*svrSearch.pl*) listening on port 8080. Code snippets for *svrSearch.pl* is in Appendix A-II;

- Search Agent signals *process.php* to inform the user that his/her search request has been received and that responses will be sent to his/her e-mail address.

**Results**

Results for the two scenarios are presented below:

- **Scenario #1:** when the user clicks on the *go to My Account* link to update his/her account details; the outcome is presented in Figure 6.3 for user gkomakech@dicts.mak.ac.ug;

- **Scenario #2:** Figure 6.4 shows the output sent and processed by *svrSearch.pl* for user gkomakech@dicts.mak.ac.ug after pressing the *Submit* button with search criterion set to Toyoto Corolla, automatic transmission, mileage limit of 80.000, 4 doors, 18CC, price range of 8-10 million and model 1998. Figure 6.5 meanwhile shows what was sent to the user’s e-mail address.
Figure 6.3: Results generated by clicking Go to My Account

Figure 6.4: Results generated by search.php form
Figure 6.5: Search Result sent to customer’s e-mail
6.2.3 The Account maintenance form - account.php

The *account.php* form is presented diagrammatically below.

![Account Management Screen](image)

Figure 6.6: eStore account maintenance Screen

**Implementation snapshot**

There are two scenarios that have been implemented by the *account.php* form:

- Scenario #1: when an existing eStore user is updating his/her account details;
- Scenario #2: when a new eStore customer is creating an account.

All the two scenarios were implemented in similar ways:
• *account.php* performs form level validation, to check all input fields entered by the customer. For example, enforcing that the user supplied all the mandatory fields: e-mail and password fields and enforcing input mask: e.g ensuring that the e-mail field is entered in a compatible e-mail notation format;

• *account.php* passes control to *process.php* (Appendix A-I) and instructs it to process the user’s input;

• *process.php* formats the user’s input in a way that the ACC Manager understands, and then connects to it (*svrAccount.pl*) on port 9090. The complete code snippet for *svrSearch.pl* is in Appendix A-III. The message sent to svrAccount.pl has flags to signal to ACC Manager whether the information should be used to to create a new account or update an existing account;

• *svrAccount.pl* unpacks the message, and based on its content, creates or updates an eStore account.

**Results**

Figure 6.7 below shows the resulting record created for user gkomakech@dicts.mak.ac.ug.

<table>
<thead>
<tr>
<th>EMAIL</th>
<th>PASSWORD</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>PHONE</th>
<th>MODIFIED_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:sunds@yahoo.com">sunds@yahoo.com</a></td>
<td>89ijh637</td>
<td>SUNDA</td>
<td>SUNDA</td>
<td>0772290007</td>
<td>21/01/2006</td>
</tr>
<tr>
<td><a href="mailto:sofim@dicts.mak.ac.ug">sofim@dicts.mak.ac.ug</a></td>
<td>iue9121</td>
<td>SAMUEL</td>
<td>OTIM</td>
<td>0712123456</td>
<td>20/01/2006</td>
</tr>
<tr>
<td><a href="mailto:gkomakech@dicts.mak.ac.ug">gkomakech@dicts.mak.ac.ug</a></td>
<td>9103h1321</td>
<td>GABRIEL</td>
<td>KONAKECHI</td>
<td>0772212477</td>
<td>20/01/2006</td>
</tr>
</tbody>
</table>

Figure 6.7: Output for creation of new account
6.3 Implementing the Business Layer

The server threads for the business layer were implemented in Perl programming language. The three server processes that were implemented are:-

- The Search Agent - svrSearch.pl;
- The ACC Manager Agent - srvAccount.pl;
- The Dispatch Agent - svrDispatch.pl.

svrSearch.pl and srvAccount.pl are socket-based server processes, whilst svrDispatch.pl (Appendix A-IV) was a non-socket-based server process. svrSearch.pl listens on TCP port 8080 and srvAccount.pl on TCP port 9090 of the eStore webserver machine. Complete code snippets for the three agents are in the Appendix.

6.4 Implementing the Data Layer

The data layer was implemented in MySQL. The code snippet in Figure 6.8 and 6.9 was used to create the physical objects of the data layer: Tables and fields. Figure 6.10 shows extract of the existing catalog in the Automobile relation/table.
-- MySQL dump 10.9 -- Host: localhost Database: estore --
                       --------------- ---------------
  4.1.11

/*!40101 SET CHARACTER_SET_CLIENT=utf8 */;

/*!40101 SET CHARACTER_SET_RESULTS=utf8 */;

/*!40101 SET COLLATION_CONNECTION=utf8 */;

/*!40101 SET NAMES utf8 */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;

/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;

/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;

DROP TABLE IF EXISTS `automobile`;
CREATE TABLE `automobile` (  `reg_no` varchar(250) NOT NULL default '',  `make` varchar(250)  NOT NULL default '',  `type` varchar(250) NOT NULL default '',  `condition` varchar(250) NOT NULL default '',  `location` varchar(250),  `model` varchar(10),  `price` bigint(20) unsigned default '0',  `full_desc` varchar(250) default '',  `year_manu` varchar(5) default '',  `chassis ` varchar(100) default '',  `no_of_doors` bigint(20) unsigned default '0',  `color` varchar(20) ,  `cc` varchar(20) ,  `mileage` varchar(20) ,  `extras` varchar(20) ,  `taxes` varchar(100) ,  `other` varchar(100) ,  `picture_loc` varchar(250) ,  `modified_time` date NOT NULL default '0000-00-00',  `modified_by` varchar(100) NOT NULL default 'admin@estore.com',  PRIMARY KEY  (`reg_no`) ) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

-- Table structure for table `ACCOUNT` --
DROP TABLE IF EXISTS `account`;
CREATE TABLE `account` (  `e_mail` varchar(100) NOT NULL,  `password` varchar(100) collate latin1_general_ci NOT NULL default '',  `first_name` varchar(250) collate latin1_general_ci NOT NULL default '',  `last_name` varchar(250) collate latin1_general_ci NOT NULL default '',  `phone` varchar(100) collate latin1_general_ci NOT NULL default '',  `modified_time` date NOT NULL default '0000-00-00',  `modified_by` varchar(100) NOT NULL default 'admin@estore.com',  PRIMARY KEY  (`e_mail`) ) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

-- Table structure for table `REQUEST` --
DROP TABLE IF EXISTS `request`;
CREATE TABLE `request` (  `ref_no`tinyint(3) unsigned NOT NULL auto_increment,  `e_mail` varchar(100) NOT NULL,  `criteria` varchar(250) collate latin1_general_ci NOT NULL default '',  `sent` varchar(2) NOT NULL default 'n',  `modified_time` date NOT NULL default '0000-00-00',  `modified_by` varchar(100) NOT NULL default 'admin@estore.com',  PRIMARY KEY  (`ref_no`) ) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

-- Table structure for table `MATCH` --
DROP TABLE IF EXISTS `match`;
CREATE TABLE `match` (  `ref_no`tinyint(3) unsigned NOT NULL auto_increment,  `e_mail` varchar(100) NOT NULL,  `reg_no` varchar(250) collate latin1_general_ci NOT NULL default '',  `sent` varchar(2) NOT NULL default 'n',  `modified_time` date NOT NULL default '0000-00-00',  `modified_by` varchar(100) NOT NULL default 'admin@estore.com',  PRIMARY KEY  (`ref_no`) ) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

Figure 6.8: Data Layer code snippet

Figure 6.9: Data Layer code snippet - Cont...
### Figure 6.10: Automobile Catalog

<table>
<thead>
<tr>
<th>REG NO</th>
<th>MAKE</th>
<th>TYPE</th>
<th>LOCATION</th>
<th>MODEL</th>
<th>PRICE</th>
<th>NO_OF</th>
<th>CC</th>
<th>MILEAGE</th>
<th>PICTURE_LOC</th>
<th>MODIFIED</th>
<th>MODIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAE378K</td>
<td>TOYOTA COROLLA</td>
<td>AUTOMATIC</td>
<td>KAMPALA</td>
<td>1999</td>
<td>900000</td>
<td>18</td>
<td>73000</td>
<td></td>
<td>home/fk/kmarchi/project/ca001</td>
<td>9/01/2006</td>
<td>root</td>
</tr>
<tr>
<td>UAF216S</td>
<td>SUZUKI</td>
<td>MANUAL</td>
<td>KAMPALA</td>
<td>1992</td>
<td>700000</td>
<td>16</td>
<td>98000</td>
<td></td>
<td>home/fk/kmarchi/project/ca002</td>
<td>9/01/2006</td>
<td>root</td>
</tr>
<tr>
<td>UAG004V</td>
<td>TOYOTA COROLLA</td>
<td>AUTOMATIC</td>
<td>KAMPALA</td>
<td>1999</td>
<td>120000</td>
<td>16</td>
<td>96000</td>
<td></td>
<td>home/fk/kmarchi/project/ca003</td>
<td>9/01/2006</td>
<td>root</td>
</tr>
</tbody>
</table>
Chapter 7

Conclusion, Recommendations and Future work

7.1 Conclusion

The Internet technology is one of the most rapidly growing field in today’s ICT world. From the time of its inception, to date, the number of programming languages supported by WWW has been on the increase. Additionally, WWW has been improved, to not only support static text, but also to support dynamic pages that allows interactivity with the end-user. The defacto HTML language has been perceived simple to learn and use, and as such there are so many web masters, authoring all sort of contents on the web. Although the WWW environment supports true 3-tier development, many of the existing web applications are still 2-tier. These kinds of applications are difficult to maintain. In this research, we have walked through a systematic approach to building component-based, 3-tier, Web based application by developing a simple web portal based on independent pieces of software agents, that interact together to achieve the project objectives.
7.2 Recommendations

Web site developers adopt the agent-based model as a basis for developing dynamic Web sites.

7.3 Future Work

This research was a demonstration of concept, and as such did not present a complete benefit of component-based web development. The following areas still calls for further research:-

- Integration with defacto agents communication languages - Message passing techniques between the client and the various agents can be improved to utilize known agent communication language. This project used a self-defined format, and thus presents some limitations.

- agents as self-extractors - It would be a good idea to bring agents to the front: at the user interface. Currently, most agents are back-end services. It could be very interesting if the various agents just go on advertising themselves on random websites, telling users what services they provide and once a user clicks on them, they extracts themselves into dynamic 3-tier architecture and serves the given user.

- implementing the transition phase - the RUP’s transition phase was not implemented in this research. Further work in this phase will take the product through usability testing and features enhancement to the product.
Bibliography


APPENDIX A-I : PROCESS.PHP

```php
<? include "conf.php"; ?>

//We are dealing with the Search Form - Generate the search select by the user and send to server
//We know make and transmission are mandatory fields - no need to check
if($_POST['formname']=='1') {
    $make = " make='" . $_POST['make'] . "' " ;
    $transmission = " type='" . $_POST['type'] . "' " ;
    $milleage = " milleage='" . $_POST['milleage'] . "' " ;
    $no_of_door = "no_of_door='" . $_POST['no_of_door'] . "' " ;
    $cc = "cc='" . $_POST['cc'] . "' " ;
    $price = "price between " . $_POST['price_max'] . " AND " . $_POST['price_min'] . "' " ;
    $model = "model='" . $_POST['model'] . "' " ;
    //GENERATE SQL STATEMENT
    if($_POST['milleage'] > 10){
        $SQL = $SQL . " AND " . $milleage ;
    }
    if ($_POST['no_of_door'] >= 2){
        $SQL = $SQL . " AND " . $no_of_door ;
    }
    if ($_POST['cc'] >= 10){
        $SQL = $SQL . " AND " . $cc ;
    }
    if ($_POST['price_max'] >= 1 and $_POST['price_min'] >=1){
        $SQL = $SQL . " AND " . $price ;
    }
    if (strlen($_POST['model']) == 4){ //model must be 1900,1998 etc.
        $SQL = $SQL . " AND " . $cc ;
    }
    echo " This is now the SQL STATEMENT TO PROCESS \n n $SQL" ;
    //Send the message to server
    send_to_server($CUSTOMER["e_mail"] . ": $SQL",8080);
}

//WE ARE DEALING WITH THE ACCOUNT FORM
else if ($_POST['formname']=='2'){ //If no session exist, then its a new user
    $action=1;
    if($CUSTOMER["e_mail"]){$action=2;}else{$action=1;}
    $msg .= "$action:" . $_POST['e_mail'] . ":" . $_POST['password'] . ";" ;
    $msg .= $_POST['first_name'] . ":" . $_POST['last_name'] . ":" . $_POST['phone'] ;
    //Send to Server
    send_to_server($msg,9090);
}
```

Figure 7.1: process.php
function Send_to_server($msg,$service_port) {
    echo "<h2>TCP/IP Connecting to Server...</h2> \n";
    getservbyname ('www', 'tcp');
    gethostbyname ('localhost');
    /* Create a TCP/IP socket. */
    $socket = socket_create (AF_INET, SOCK_STREAM, 0);
    if ($socket < 0) {
        echo "socket_create() failed: reason: " . socket_strerror ($socket) . " \n";
    } else {
        echo "Connected to Server. \n";
    }
    $result = socket_connect ($socket, $address, $service_port);
    if ($result < 0) {
        echo "socket_connect() failed. \nReason: (\n" . socket_strerror ($result) . " \n";
    } else {
        echo "OK. \n";
    }
    $in = $msg ;
    //Send to Server
    socket_write ($socket, $in, strlen ($in));
    socket_close ($socket);
}

function log_user($e_mail, $pass) {
    //connnect to database
    $db = mysql_pconnect ('localhost', 'root', 'goaway');
    if (!$db) { echo "Failed to connect to database...", exit ; }
    $u = 0;
    $u = mysql_query("select e_mail from account where e_mail = '$e_mail' AND password='$pass' ");
    if (mysql_num_rows($u))
        return 1;
    else
        return 0 ;
    //disconnect from database
    mysql_close($db);
}

Figure 7.2: process.php cont...
APPENDIX A-II : SVRSEARCH.PL

#!/usr/bin/perl -w
#-------------------
use strict;
use Socket;
my $port = shift || 8080;
my $proto = getprotobyname('tcp');
my $client_addr;
my $msg;

# create a socket, make it re usable and attach it to this machine
socket(SERVER, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
setsockopt(SERVER, SOL_SOCKET, SO_REUSEADDR, 1) or die "setsock: $!";
my $paddr = sockaddr_in($port, INADDR_ANY);
# bind to a TCP port, then listen
bind(SERVER, $paddr) or die "bind: $!";
listen(SERVER, SOMAXCONN) or die "listen: $!";
print "SERVER started on port $port\n";

#Now we need to create two processes here - The parent will be checking
#for matching automobile - while the child waits for client connections.
#
#parent process
if (fork){
    #we also enter infinite loop...but we just wake up occasionally - to search for
    #matching products
    while(1){
        sleep (60*60*30);
        &search ;
    }
}
else{
    &set_criteria ;
}

#Sub Routine to Read input from the client
sub set_criteria {
    while ($client_addr = accept(CLIENT, SERVER)){
        # Get details of connecting client and log it
        my ($client_port, $client_ip) = sockaddr_in($client_addr);
        my $client_ipnum = inet_ntoa($client_ip);
        my $client_host = gethostbyaddr($client_ip, AF_INET);
        print "got a connection from - read_client: $client_host", "$client_ipnum\n";
        #Receive message from the client
        while ( $msg = <CLIENT> ){
            print "MESSAGE RECEIVED: [$msg] \n";
            $_=$msg;
        }
        close CLIENT;
    }
}

#child process - Set criteria of whatever the user has sent.
#we trust that the client has done all checks before sending anything to us.

else(
    &set_criteria ;
)

#Sub Routine to Read input from the client


Figure 7.3: svrSearch.pl
sub save_criteria{
  use DBI   ;
  $|=1     ;
  $dsn = "DBI:mysql:estore;localhost";
  $dbh = DBI->connect($dsn, 'root', 'goaway');

  if ( !defined $dbh ) {
    exit;
  }

  # use the four lines below if the query results
  # in many rows or if you get 'out of memory' errors
  # We are sure the client passed the correct fields. No need to check here
  $SQL_QUERY="<<__CURSOR_1__;
    insert into request (e_mail,criteria)
    values ('$_[1]','$_[2]')
  __CURSOR_1__
  $cursor = $dbh->prepare( "$SQL_QUERY" );
  $cursor->execute;
  $cursor->finish;
  $dbh->disconnect;
}

sub search{
  use DBI   ;
  $|=1     ;
  $dsn = "DBI:mysql:estore;localhost";
  $dbh = DBI->connect($dsn, 'root', 'goaway');

  if ( !defined $dbh ) {
    exit;
  }

  # use the four lines below if the query results
  # in many rows or if you get 'out of memory' errors
  # We are sure the client passed the correct fields. No need to check here
  $SQL_QUERY="<<__CURSOR_1__;
    select ref_no,e_mail,criteria from request;
  __CURSOR_1__
  $cursor = $dbh->prepare( "$SQL_QUERY" );
  $cursor->execute;
  $cursor->finish;
  $dbh->disconnect;
}

Figure 7.4: svSearch.pl cont...
sub process_row
use DBI;
$s|=1;
$dsn = "DBI:mysql:estore:localhost";
$dbh = DBI->connect($dsn, 'root', 'goaway');
if ( !defined $dbh ) {
  exit;
}

# use the four lines below if the query results
# in many rows or if you get 'out of memory' errors
#$SQL_QUERY="SET SQL_BIG_TABLES=1";
# $cursor = $dbh->prepare("$SQL_QUERY");
# $cursor->execute;
# $cursor->finish;

# We are sure the client passed the correct fields. No need to check here
# The second parameter contains the correct sql statement to search for a product
$SQL_QUERY=<<__CURSOR_1__;
  $_[1];
__CURSOR_1__
$cursor = $dbh->prepare("$SQL_QUERY");
$cursor->execute;
# retrieve all rows matching this criteria and save into the a list object
while ( $cursor->fetchrow ){  
  ($ref_no,$e_mail,$reg_no)=split($_)   ;
  &save_row($ref_no,$e_mail,$reg_no)    ;
}  
$cursor->finish;
$dbh->disconnect;
# for each row retrieved, we return list to caller...so, it better be the last statement
}

sub save_row
use DBI;
$s|=1;
$dsn = "DBI:mysql:estore:localhost";
$dbh = DBI->connect($dsn, 'root', 'goaway');
if ( !defined $dbh ) {
  exit;
}

# use the four lines below if the query results
# in many rows or if you get 'out of memory' errors
#$SQL_QUERY="SET SQL_BIG_TABLES=1";
# $cursor = $dbh->prepare("$SQL_QUERY");
# $cursor->execute;
# $cursor->finish;

# We are sure the client passed the correct fields. No need to check here
# We are sure the client passed the correct fields. No need to check here
$SQL_QUERY=<<__CURSOR_1__;
  insert into match (ref_no,e_mail,reg_no,sent)
  values ('$_[1]','$_[2]','$_[3]','N')
__CURSOR_1__
$cursor = $dbh->prepare("$SQL_QUERY");
$cursor->execute;
$cursor->finish;
$dbh->disconnect;
}  

Figure 7.5: svSearch.pl cont...
#!/usr/bin/perl -w
#-----------------------------
use strict;
use Socket;
my $port = shift || 9090;
my $proto = getprotobyname('tcp');
my $client_addr;
my $msg;

# create a socket, make it reusable and attach it to this machine
socket(SERVER, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
setsockopt(SERVER, SOL_SOCKET, SO_REUSEADDR, 1) or die "setsock: $!";
my $paddr = sockaddr_in($port, INADDR_ANY);

# bind to a TCP port, then listen
bind(SERVER, $paddr) or die "bind: $!";
listen(SERVER, SOMAXCONN) or die "listen: $!";
print "SERVER started on port $port
";

# Wait for client connections
sub read_cli;

sub read_cli
{
    while ($client_addr = accept(CLIENT, SERVER))
    {
        my ($client_port, $client_ip) = sockaddr_in($client_addr);
        my $client_ipnum = inet_ntoa($client_ip);
        my $client_host = gethostbyaddr($client_ip, AF_INET);
        print "got a connection from - read_cli: $client_host", 
            "[$client_ipnum]\n";
        
        #Receive message from the client
        while ( $msg = <CLIENT> )
        {
            print "MESSAGE RECEIVED: [ $msg ]\n";
            $_=$msg;
            close CLIENT;
            #--subread_cli
        }
    }
}

#Sub Routine to Read input from the client

my ($flag,$e_mail,$password,$first_name,$last_name,$phone)=split(':',$_);

if ($message[0] ==1)
    &new_acc(@message);
elsif ($message[0] ==2)
    &update_acc(@message);
elsif ($message[0] ==3)
    &delete_acc(@message);
elsif ($message[0] ==4)
    &read_acc() ;
}    #--close accept
}    #--close sub read_cli

Figure 7.6: svrAccount.pl
sub new_acc{
  use DBI   ;
  $|=1     ;
  $dsn = "DBI:mysql:estore;localhost";
  $dbh = DBI->connect($dsn, 'root', 'goaway');
  if ( !defined $dbh ) { exit; }
  # use the four lines below if the query results
  # in many rows or if you get 'out of memory' errors
  #$SQL_QUERY="SET SQL_BIG_TABLES=1";
  # $cursor = $dbh->prepare("$SQL_QUERY");
  # $cursor->execute;
  # $cursor->finish;
  #We are sure the client passed the correct fields. No need to check here
  $SQL_QUERY=<<__CURSOR_1__;
    insert into account (e_mail,password, first_name,last_name, phone)
    values ('$_[1]','$_[2]','$_[3]','$_[4]')
  __CURSOR_1__
  $cursor = $dbh->prepare( "$SQL_QUERY" );
  $cursor->execute;
  $cursor->finish;
  $dbh->disconnect;
}

sub update_acc{
  use DBI   ;
  $|=1     ;
  $dsn = "DBI:mysql:estore;localhost";
  $dbh = DBI->connect($dsn, 'root', 'goaway');
  if ( !defined $dbh ) { exit; }
  # use the four lines below if the query results
  # in many rows or if you get 'out of memory' errors
  #$SQL_QUERY="SET SQL_BIG_TABLES=1";
  # $cursor = $dbh->prepare("$SQL_QUERY");
  # $cursor->execute;
  # $cursor->finish;
  #We are sure the client passed the correct fields. No need to check here
  $SQL_QUERY=<<__CURSOR_1__;
    update account
    set password = '$_[2]',
    first_name = '$_[3]',
    last_name = '$_[4]',
    phone = '$_[5]' where e_mail = '$_[1]';
  __CURSOR_1__
  $cursor = $dbh->prepare( "$SQL_QUERY" );
  $cursor->execute;
  $cursor->finish;
  $dbh->disconnect;
}
sub delete_acc
{
  use DBI ;
  $|=1     ;
  $dsn = "DBI:mysql:estore:localhost";
  $dbh = DBI->connect($dsn, "root", "goaway");

  if ( !defined $dbh ) {
    exit;
  }

  # use the four lines below if the query results
  # in many rows or if you get 'out of memory' errors
  #$SQL_QUERY="SET SQL_BIG_TABLES=1";
  # $cursor = $dbh->prepare("$SQL_QUERY");
  # $cursor->execute;
  # $cursor->finish;

  # We are sure the client passed the correct fields. No need to check here
  $SQL_QUERY=<<__CURSOR_1__;
  delete from account
  where account.e_mail = '$_[1]';
  __CURSOR_1__

  $cursor = $dbh->prepare("$SQL_QUERY");
  $cursor->execute;
  $cursor->finish;
  $dbh->disconnect;
}

sub read_acc
{
  use DBI ;
  $|=1     ;
  $dsn = "DBI:mysql:estore:localhost";
  $dbh = DBI->connect($dsn, "root", "goaway");

  if ( !defined $dbh ) {
    exit;
  }

  # use the four lines below if the query results
  # in many rows or if you get 'out of memory' errors
  #$SQL_QUERY="SET SQL_BIG_TABLES=1";
  # $cursor = $dbh->prepare("$SQL_QUERY");
  # $cursor->execute;
  # $cursor->finish;

  # We are sure the client passed the correct fields. No need to check here
  $SQL_QUERY=<<__CURSOR_1__;
  select e_mail,password,first_name, last_name,phone from account
  where account.e_mail = '$_[1]';
  __CURSOR_1__

  $cursor = $dbh->prepare("$SQL_QUERY");
  $cursor->execute;
  my @row = $cursor->fetchrow ;
  $cursor->finish;
  $dbh->disconnect;

  # We return list to caller...so, it better be the last statement
  @row ;
}

Figure 7.8: srvAccount.pl cont...
#!/usr/bin/perl -w

# when the dispatch agent starts, it enters infinite loop and just wakes up
# periodically to send existing matches to users' email.

while(1){
    sleep (60*60*30);
    &find_match;
}

sub send_mail{
    use Socket;
    use strict;
    my($mailTo) = @$_[0];
    my($mailServer) = @$_[1];
    my($subject) = 'Your Vehicle Request';
    my($mailFrom) = 'sales@restore.com';
    my($realName) = 'Customer Rep.';
    $main::SIG{'INT'} = 'closeSocket';
    my($proto) = getprotobyname("tcp") || 6;
    my($sport) = getservbyname("SMTP", "tcp") || 25;
    my($serverAddr) = (gethostbyname($mailServer))[4];
    if (!defined($length)) {
        die('gethostbyname failed.');
    }
    socket(SMTP, AF_INET(), SOCK_STREAM(), $proto)
        || die("socket: $!");
    $packFormat = 'S n a4 x8'; # Windows 95, SunOs 4.1+
    connect(SMTP, pack($packFormat, AF_INET(), $port, $serverAddr))
        || die("connect: $!");
    select(SMTP); $| = 1; select(STDOUT); # use unbuffered i/o.
    
    my($inpBuf) = '';
    recv(SMTP, $inpBuf, 200, 0);
    recv(SMTP, $inpBuf, 200, 0);
    sendSMTP(1, "HELO
");
    sendSMTP(1, "MAIL From: <$mailFrom>
");
    sendSMTP(1, "RCPT To: <$mailTo>
");
    sendSMTP(1, "DATA
");
    send(SMTP, "From: $realName
", 0);
    send(SMTP, "MIME-Version: 1.0
", 0);
    send(SMTP, "Content-Type: multipart/mixed
", 0);
    send(SMTP, "Content-Transfer-Encoding: 8bit
", 0);
    send(SMTP, "Content-Disposition: attachment; filename="$_[2]
", 0);
    send(SMTP, "Subject: $subject
", 0);
    send(SMTP, ";
");
    close(SMTP);
}

sub closeSocket {
    # close smtp socket on error
    close(SMTP);
    die("SMTP socket closed due to SIGINT\n");
}

Figure 7.9: srvDispatch.pl
sub sendSMTP {
    my($debug) = shift;
    my($buffer) = @_;  
    print STDERR ("> $buffer") if $debug;
    send(SMTP, $buffer, 0);
    recv(SMTP, $buffer, 200, 0);
    print STDERR (< $buffer") if $debug;
    return( (split(/ /, $buffer))[0] );
}

sub find_match{
    use DBI ;
    $|=1     ;
    $dsn = "DBI:mysql:estore;localhost";
    $dbh = DBI->connect($dsn, 'root', 'goaway');

    if ( !defined $dbh ) {
        exit;
    }
    
    # use the four lines below if the query results
    # in many rows or if you get 'out of memory' errors
    #$$SQL_QUERY="SET SQL_BIG_TABLES=1";
    # $cursor = $dbh->prepare("$$SQL_QUERY");
    # $cursor->execute;
    # $cursor->finish;

    #We are sure the client passed the correct fields. No need to check here
    $SQL_QUERY="select ref_no,e_mail,reg_no from request where sent='N';"
    $cursor = $dbh->prepare( "$SQL_QUERY" );
    $cursor->execute;
    #retrieve all the rows into a list object
    while ( $cursor->fetchrow ){
        #split the row into components and send for processing
        ($ref_no,$e_mail,$reg_no)=split($_)   ;
        &process_row($e_mail,$reg_no)         ;
    }
    $cursor->finish;
    $dbh->disconnect;
}

Figure 7.10: srvDispatch.pl cont...
sub process_row{
  use DBI ;
  $|=1 ;
  $dsn = "DBI:mysql:estore;localhost" ;
  $dbh = DBI->connect($dsn, 'root', 'goaway');

  if ( !defined $dbh ) { exit; }

  # use the four lines below if the query results
  # in many rows or if you get 'out of memory' errors
  #$SQL_QUERY="SET SQL_BIG_TABLES=1" ;
  # $cursor = $dbh->prepare( "$SQL_QUERY" );
  # $cursor->execute;
  # $cursor->finish;

  #We are sure the client passed the correct fields. No need to check here
  #The second parameter contains the correct sql statement to search for a product
  $SQL_QUERY=<<__CURSOR_1__ ;
    select * from automobile where reg_no='$_[1]';
  __CURSOR_1__
  $cursor = $dbh->prepare( "$SQL_QUERY" );
  $cursor->execute;

  #There is always only one row matching a given reg_no. so retrieve it
  $cursor->fetchrow
  $cursor->finish;
  $dbh->disconnect;

  #split the row into components, formulate server address and message body
  # then send message to customer
  ($make,$type,$condition,$location,$model,$price,$full_desc,$picture_loc)=split($_) ;
  ($name,$smtp_server)=split('@',$_[0]);
  my ($message_body) .= "We found specification for the car you requested
    Type = $type 
    Condition=$condition
    location=$location
    model=$model
    price=$price
    Description=$full_desc
    
    #Send the mail now!!
    &send_mail($_[0],$smtp_server,$picture_loc,$message);
}

Figure 7.11: srvDispatch.pl cont...