INFORMATION TO USERS

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

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

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

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

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

UMI®
University of Ottawa

Master's Program in Systems Science

Thesis

System Integration: Application of Web-Based DBMS and Intelligent Agent

Student Name: Wei Chen
Student Number: 1841467

Thesis Director: Professor Daniel Lane

Faculty of Administration
University of Ottawa

Aug. 15, 2002

© Wei Chen, Ottawa, Canada, 2002
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

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

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

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

0-612-76569-5
Abstract

Considering the redundancy and the irrelevancy of current search engines in the World Wide Web (WWW), this thesis explores an alternative web-based method of Internet search and retrieval of information via an Intelligent Agent. The Intelligent Agent retrieves the information back to users and it inserts this information automatically into the users' database.

There are three components in this system: (1) Multiple User Dungeon (MUD) as a front end, (2) dynamic database management system, and (3) Intelligent Agent as a back end. The front end is a web site open to all the users of this system. It allows different users to access the system simultaneously from different PCs inside the same Intranet. The DBMS database system is designed using Microsoft Access. It is the reservoir of all the data. It provides information to the front end when the users query the database. Intelligent agent (back-end) is implemented using Visual C++ 6.0. It takes advantage of the technique of Open Database Connectivity (ODBC) to connect to the DBMS. The Intelligent Agent searches the Internet through the search engine of Google and then inserts the records into the database based on the user preference. The Intelligent Agent can also update the database by deleting the obsolete records automatically.

In order to test and implement this application system, it is applied two business cases: (1) Fisheries Resource Conservation Council (FRCC) and (2) Sport Utility Vehicle (SUV) database system. DBMS and back-end are transparent to users. Users simply need to manipulate the front end interface that links the user to all database review and modification functions, as well as to the search and automatic retrieval options. Results of the implementation of the applications are presented and future research applications for Internet information search and retrieval processes are discussed.
--- Table of Contents ---

1. INTRODUCTION .................................................................................. 5
  1.1 MOTIVATION ........................................................................... 5
  1.2 INTELLIGENT INFORMATION SEARCH AND RETRIEVAL AGENT .... 6
  1.3 WEB-BASED FRONT END ....................................................... 7
  1.4 WEB DATABASE .................................................................. 9
  1.5 WEB-BASED BACK END: THE INTELLIGENT AGENT .............. 10
  1.6 PLAN OF THE PROPOSAL ...................................................... 11

2. REVIEW OF THE LITERATURE .......................................................... 12
  2.1 MUD .................................................................................. 12
  2.2 DBMS .............................................................................. 13
  2.3 SEARCH ENGINE .................................................................. 15

3. METHODOLOGY ................................................................................ 18
  3.1 WEB DESIGN USING FRONTPAGE 2000 ............................... 18
  3.2 DBMS USING ACCESS ......................................................... 18
  3.3 INTELLIGENT AGENT .......................................................... 20
  3.4 GENERAL ISSUES FOR IMPLEMENTATION ......................... 23
    3.4.1 Internet search .............................................................. 23
    3.4.2 Insertion into database ............................................... 28
    3.4.3 Update of the database ............................................... 30

4. APPLICATION AND CASE STUDY .................................................... 33
  4.1 FRCC ECO DATA CASE STUDY ........................................... 34
    4.1.1 FRCC Background (FRCC 2001) .................................... 34
    4.1.2 EcoData Web Application ............................................. 34
    4.1.3 Intelligent Agent in FRCC EcoData ............................... 37
  4.2 E-COMMERCE CASE STUDY ............................................... 40
    4.2.1 Intelligent Agent in E-commerce .................................. 40
    4.2.2 Application in E-Commerce: Buying a SUV .................... 40
    4.2.3 Some SUV Issues ....................................................... 41
    4.2.4 Web implementation of Intelligent Agent in SUV ............ 42

5. RESULTS ......................................................................................... 47
  5.1 APPLICATION ................................................................. 47
  5.2 REPORTS ABOUT THE SEARCH RESULT ............................ 47
  5.3 DATABASE SIZE ............................................................... 47
  5.4 ECOSYSTEM DATABASE AND USE --- MANUAL ENTRY VS. AUTOMATIC... 48

6. FUTURE RESEARCH CONSIDERATIONS ......................................... 49
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Spider vs. Agent</td>
<td>49</td>
</tr>
<tr>
<td>6.2 Other Applications</td>
<td>49</td>
</tr>
<tr>
<td>6.3 ISAPI</td>
<td>50</td>
</tr>
<tr>
<td>6.4 Security</td>
<td>50</td>
</tr>
<tr>
<td>6.5 Interpretation of the Search Engine Formats</td>
<td>51</td>
</tr>
<tr>
<td>6.6 Local Database Search or Internet Search</td>
<td>52</td>
</tr>
<tr>
<td>7. Bibliography</td>
<td>53</td>
</tr>
<tr>
<td>Appendix A</td>
<td>A-1</td>
</tr>
<tr>
<td>Appendix B</td>
<td>B-1</td>
</tr>
<tr>
<td>Appendix C</td>
<td>C-1</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: Web Database Design for a General Application ........................................ 8
Figure 2: The relationship diagram of a company (Nel 2001) .................................. 19
Figure 3: System integration application diagram showing components for Front End, DBMS and Agent-Internet Back End. ................................................................. 21
Figure 4: Web-Based Back End: The Intelligent Agent ............................................ 22
Figure 5: Internet Search Process ........................................................................... 24
Figure 6: GUI of the Intelligent Agent ................................................................. 25
Figure 7: Google search result .............................................................................. 26
Figure 8: Final search result .................................................................................. 27
Figure 9: Database insertion process .................................................................... 28
Figure 10: The feedback web page of an insertion into the database .................... 30
Figure 11: Principle of updating the database ......................................................... 31
Figure 12: The pop-up window of update ............................................................... 32
Figure 13: FRCC EcoData Project Home Page ....................................................... 35
Figure 14: The Query Web Page of FRCC EcoData Project .................................... 36
Figure 15: The Result Page of the Query ............................................................... 37
Figure 16: Results of Query by Date ..................................................................... 38
Figure 17: A Specific Record ................................................................................. 39
Figure 18: The Home Page of SUV Application ..................................................... 43
Figure 19: The Query Web Page of the Intelligent Agent ....................................... 44
Figure 20: The Database of SUV .......................................................................... 45
Figure 21: A Query record of SUV ....................................................................... 46
Figure 22: Interpretation of the search result (Google 2002) .................................. 51
1. INTRODUCTION

1.1 Motivation

The rapid popularization of the Internet has changed the mode of access to information for millions of economic agents, from households to companies of all sizes. What is often termed the "information revolution" can be characterized by a radical communication change. The Internet enables the provision of services and transmission of information from a distance. It has been the most immediate application of Internet communication. However, looking for specific information can be more and more often like searching for a needle in the haystack of Cyberspace. Internet Database Technologies has to be developed by removing the guesswork from complicated Internet and database issues.

Human beings have organized information for future usage and retrieval since about 4,000 years ago. A very common example is the table of contents of a book. People can just simply browse them in order to find the interesting information. They don’t have to waste time flipping through the book page by page. A table of contents can also be viewed as a data structure for fast information access. Further, there is another old and popular data structure for information search too. It is an index. Most people are familiar with the use of index. Accompanying the emergence of the computer, database software was developed and used. IBM, Microsoft, Oracle and many different companies are in this field of business. Software such as Informix, MySQL, Oracle9i, and Access are some popular database systems. These software provide lots of efficiency to the human daily work.

A library may be the first concept that comes to mind whenever talking about information search and retrieval. In the very beginning, the library system allowed searches based on the author name and title. Later on, they add some more searching functionality such as subject headings, keywords, and some more complex query facilities using SQL language. Currently, the search is more and more graphical directed and electronic (Yates & Neto, 2001).

Even today, when we using the Internet search engines, it is still using the same principle as a library. But the search engine does have a lot of revolutionary changes. It can access the information faster and cheaper. The digital communication provides an information ocean to the
search engine. Thus, it can reach an information source that is maybe far away from the user. This user can search and get this piece of distant information in just a few seconds.

Moreover, the advent of e-commerce is allowing the dissemination of widely different services. In particular, it enables more advanced network connections, either in a technical sense, or resorting to every human aspect of networking. For instance, not only can consumers perform on-line shopping as exemplified by such “dotcom” enterprises as Amazon.com, but they can also leverage their queries into a chain of communications, as one server relays a request, transforms it, uses its own resources and propagates the request to other servers. This spawns the concept of agent searches, an outgrowth of simple search services. On the Internet, an agent or an intelligent agent is a customized software or program that gathers information on user demand (Whatis.com 2000).

There are several concerns about this concept. First, we have to think about the users. Normal Internet search engines such as Google or Yahoo retrieve search results, which may contain duplicate addresses, and many useless pages that do not correspond to the user’s request. Alternatively, an intelligent agent search should review pages, remove duplicates and cut down on the number of peripheral links to the user’s request. The agent should not only help the user to query and find useful information, but it should also update and insert this information automatically into the user’s database. A web-based software application and template framework need to be developed with the users specific application in mind to fulfill this entire procedure of search and build. A user interface is also required to assist the search and build data easily. We coin the name for this application as the “Intelligent Agent”.

1.2 Intelligent Information Search and Retrieval Agent

The current research focuses on the issues of intelligent web database design through the use of existing Internet search engines. Users may visit the application site and database through a dedicated web site, and can update or modify their own database by using the new focused searching application program – Intelligent Agent. An illustrative graphic of this model is described in Figure 1.
There are three components to this model:

Front end – user interface module.

Database Management System (DBMS) – Physical memory space for data storage.

Back end – search engine, analysis and retrieval. We describe these components below.

1.3 Web-Based Front End

The front end consists of multiple user hosts and the web server. The web server can support all the concurrent users. Every user is independent of one another. These characteristics are defined as a MUD-like (Multiple User Dungeon) (Lovato, 1999) design. This design introduces many MUD functionalities.

Under the front end environment (the user host), each user can make entries independently and then link to other users from the web server as long as these entries do not violate any constraint of the database. For example, a price of a book must be greater than 0. The database will reject an entry if the user enters a negative value for the price. The design of the web server must contain all the integrity constraints (Nel, 2001): domain constraints, key constraints, entity integrity, and referential integrity. These constraints are used for type checking in the database. For example, a course code in a university cannot be “Friday”, a professor name cannot be “David3S5* Smith”, or a key attribute in a table cannot be “NULL”.
Figure 1: Web Database Design for a General Application
Furthermore, if the user inserts an improper entry, the web server also should be able to prompt a message in order to provide more information to this user for correction. Otherwise, the web server should reject the entry. The web server is acting as a security guard for the physical database. It prevents any malicious intruders from hostile entry that will corrupt the entire database. The authorized users can insert data with regard to keyword, record, geography area, or resource for the particular application.

Aside from the data entry, the most frequent usage of the web server is the data query. There are a couple ways for users to make a query. They can enter a word to search the user’s database or use the built-in search functionality (e.g. combo-boxes, keywords, record, geographic area, resource label, or even data entry date). Security restrictions must be applied for the web design. It restricts unauthorized access. Different checks can also be established for each type of access: query, modify, insert, or delete.

1.4 Web Database

The database, the second component of the Intelligent Agent, is stored in a physical memory space, and is designed by applying the most popular database model – relational model. The database will also contain definitions for the type and the structure for all data in the database. Data are stored as objects; relationships are stored as object references to other objects. Each object may have one or more than one attributes (i.e. columns in a table). The database supports multiple views for different users and supports data sharing for concurrent access of several users. Thus, in a more accurate and technical terminology, the database and its use should be called the Database Management System (DBMS) (Elmasri and Navathe, 2000).

In order to maintain the consistency, the DBMS requires a careful and elegant design. A good design will reduce the data redundancy and increase the data independency to a great extent. It should provide capabilities for defining and enforcing the domain constraint, key constraints, entity integrity, and referential integrity due to the central control. Under these constraints, the database will reject any data with a wrong type. For example, a bank account number cannot be named “Friday” and it should be rejected. In this DBMS design, it would focus on the overall
requirement of the broader organization rather than some individual or special case requirement. It supports a concurrent view of the data but must avoid changing the same data at the same time by different users, e.g. no two persons can both buy a airplane ticket for the same seat.

Security is also a serious problem because most people will not be allowed to access all the data. A student should not be allowed to see the professor's grades data for the entire class. DMBS will enforce and manage the accessing policies. Moreover, the query speed will be an issue of concern to most users if the database becomes too large. Therefore, the reference between tables must be very clear; no repeated or unnecessary information would be added into the database.

1.5 Web-Based Back End: The Intelligent Agent

After building up the database, the information needs to be put into tables. The information is what the users require. Some practical and critical problems are: what kinds of information meet the users' needs and how are these inserted into the database memory space? To answer these questions, we consider the following situations.

The good news is that there exist lots of search engines in the Internet world. Google, Yahoo, or Alta Vista, these can all do the searching job easily and quickly. They all can generate thousands and thousands of references matching user search keywords. They will also further sort these messages, or web pages in terms of the level of match or some other factor. However, the good news is exactly the bad news. It is impossible for a user to handle the vast amount of messages that are produced. This situation is worse because most of these resulted web pages are irrelevant to user's need. Therefore, we may find a great deal of worthless information from these common search engines.

We have to find another more effective way to search for useful information. It is for this reason that this research designs a new search software application, called an Intelligent Agent.

Intelligent Agent, as its name implies, only gathers the relevant information as the user requires by taking advantage of the current search engines like the ones mentioned above. In our customized Intelligent Agent program, we will build a connection to the well-known Google
search engine to carry out a basic Internet search based on the parameters which are provided by the user. Next, the results are retrieved and temporarily stored in the cache memory pointed to by the agent. After that, according to the set of keywords in the database, the content of the application, and combining the Boolean operators constructed from excluding non-relevant material, a certain amount of "garbage" web pages in the cache could be thrown out automatically. Furthermore, by applying relational algebra or Standard Query Language (SQL), we can identify specific pages that match most of the users focused needs. Based on these pages, we further look for the information that specifies what we need to put in the field of the database, e.g., we insert the corresponding database tables in terms of their attributes and data domain. These procedures will all be carried out by our smart "Intelligent Agent".

1.6 Plan of the proposal

Based on this three-component model described above, coordination among the components, and communication between the adjacent layers is required. In the following section, we describe the systems integration model beginning with its foundations in the literature review, methodology and application and results.

The literature review reports on the development of related technologies, including traditional published sources as well as web-based sources, for the MUD, DBMS, and intelligent agent search components of the model. The methodology section discusses the basic design of the system, the software development and general implementation issues.

The Web-based database application section describes two cases where the intelligent agent is applied to real applied examples. Finally, the section on preliminary results presents the experience of the applications to date and future consideration for research.
2. REVIEW OF THE LITERATURE

Innovative web-based database and web design technology specialize in helping users make the most of the Internet by providing solutions that work. It retrieves the proper information from the complex and large Internet pool, and then conveys them to the related users in different levels. Internet database technologies provide flexible organization solutions for complicated situations. In the following we will describe the current technology development based on the three components of our system model: (1) the MUD front end, (2) the DBMS, and (3) the Intelligent Agent Search.

2.1 MUD

The MUD-like front end, Central Dynamic Database Management System (DBMS), and Intelligent Agent components all interact with each other. The lower component (agent) supports the higher one (DBMS), but each of them is transparent to the other. Each component has its individual functions and is independent of the other two.

The MUD-like front end is a web server that integrates the concept and design of MUD. MUD is a multiple user-simulated environment. It inventively structures all users’ social experience on the Internet by involving a loosely organized context or theme. According to Joson Nolan (Nolan, 2001):

“MUD offers a unique text-based Computer Mediated Communication (CMC) application that allows for not only synchronous and non-synchronous communications, but the creation of and interaction with virtual objects in a shared communication space. As an educational CMC (EdCMC) tool, MUDs provide students with the opportunity for an integrated curriculum experience, combining computer programming/literacy with narrative and informal text-based language skills acquisition, and peer to peer learning.”

MUD can apply to an on-line ongoing adventure game, distance education, or even some social activities such as chatting. MUD web site requires the participant to register as a member. Then this participant adopts a character or avatar when logging into the MUD. Each MUD has its own name, special character and ambience, and set of rules. Usually, MUDs are run by advanced
participants or some programmers called wizards. Currently, along with the development of test-based MUDs, virtual reality also have been added and participants can see the characters. In fact, MUD has lots of variants: MOO (Mud-Object-Oriented), MUCKs (Multi-User Character Kit), and MUSHes (Multi-User Shared Hallucination). The major difference among these is the programming language used and the service offered.

As Nolan (2001) mentioned, MUD can apply to Educational Computer Mediated Communication (EdCMC). MUD has a proper generic nature, multiple interaction possibility, and a strong ability for data collection and analysis. At the initial state, a MUD database is blank and user-extensible. Within MUD, students and teachers can interact with each other synchronously through direct communication or asynchronously by message posting. Diversity University (DU), a MUD favourite, has successfully developed an educational MOO (DU Journal, 1994) which is a variant MUD. Diversity University Main Campus is a real-time Internet accessible virtual reality educational environment. Thousands of students, teachers, and administrators worldwide use DU classes, literature, and consulting services. By applying MUD, DU provides non-proprietary multimedia interaction via its web gateway. Accessed from Netscape, Internet Explorer, or any other graphics and Java capable browser, DU web gateway allows the addition of graphics, sound, movies, or textual objects.

2.2 DBMS

The central dynamic Database Management System (DBMS) is the core of the entire model. Elmasri and Navathe, in their new book “Fundamentals of Database Systems”, state that a DBMS is a collection of programs that enables users to create and maintain a database (Elmasri, R. and Navathe, S. B., 2000). The DBMS is hence a general-purpose software that facilities the process of defining, constructing and manipulating databases for various applications. A database can be defined as a collection of information organized in such a way that it can be accessed, updated, or managed. DBMS can further be defined as the tool that enables us to manage and interact with the database. The DBMS manages user requests (and requests from other programs) so that users and other programs are free from having to understand where the data are physically located on storage media and, especially in a multi-user system, other users may also be accessing the data. A DBMS can perform many functions, including:
Data storing;

Data structure creation and maintenance;

Concurrent access of multi users;

Security and privacy enforcement;

Data insertion, modification, and data query;

An efficient indexing for fast extraction of selected data;

Consistence among tables and attributes;

Backup and recovery protection.

Several different types of DBMSs have been developed to support all these functionalities. These systems can broadly be classified in the four classes:

Hierarchical DBMS that stores data in a tree-like structure,

Network DBMS that stores data in the form of records and links,

Relational DBMS that data is organized simply in a table structure, and

Object-oriented DBMS which were designed to handle data such as numbers and words.

The most typical class is the Relational DBMS (RDBMS). Oracle, Informix, and Sybase are some popular RDBMSs available in the market.

The application of database systems, e.g. Oracle8, is all over the world. Oracle8 is an object relational database management system, which allows objects to be stored in tables, in a manner similar to numbers and words being stored in an RDBMS system. An Oracle database physically resides in various files. For example, there is a car dealer who wants to sell cars via a web site (Hotka 2001). The dealer needs to have a sales tracking database and Oracle8 may become a good application in this situation. When this car dealer has some automobiles ready for sale,
these cars are added to a so-called "CARS_FOR_SALE" table. It then tracks the car ID, the
description of the car and the image from the physical memory space. Some built-in SQL
procedures can be used to extract the image to a specific location. An additional Oracle form
could also be added to maintain the CARS_FOR_SALE table. After the car is sold, of course, a
report or form to display or process the BUYER information will have to be made too.

By applying Oracle8 efficiently, this web site should also dynamically display auto
information on those vehicles that have been added to the CARS_FOR_SALE table object.
Another good example of applying relational DBMS is the FRCC EcoData web site:
http://www.ecodata.management.uottawa.ca/default.htm, which was developed for this research
and which we will describe in detail later. The database administrator (DBA) integrates the
DBMS - Microsoft Access in the web server.

2.3 Search Engine

The back end of the system is an agent which can fulfill the search function. Whereas agent
search is a relatively new concept, it was preceded by more common attempts such as: search
engine, robots, web crawlers, and worms. Basically, an agent is a robot. Actually, sometimes an
agent is called a bot which is short for robot (Whatis 2000). But it is user-friendlier and has more
functionality. Usually, most users are already familiar with robots and search engines. But they
may not know what they are exactly.

A robot is a program which traverses the hypertext structure on the web automatically by
retrieving a document, and recursively retrieving all documents that are referenced. As a matter
of fact, here it does not mean any particular definition of traversal algorithm. It may have some
other related applications. A search engine is a program that searches through some data sets. In
the context of the Web, the word "search engine" is most often used for search forms that search
through databases of HTML documents gathered by a robot. Most users may not distinguish the
subtle difference between web browser and robot. In fact, normal web browsers are neither
robots nor search engines, because they are operated by a human, and don't automatically
retrieve referenced documents (other than inline images).
Search engine technology has had to scale dramatically to keep up with the growth of the web. In order to understand the development of robot, we have to trace back to 1994. One of the first web search engines, the World Wide Web Worm (WWW) (McBryan, 1994) had an index of 110,000 web pages and web accessible documents. It is foreseeable that a comprehensive index of the Web will contain over a billion documents and the number of requests search engines handle will grow dramatically in only a few years. In March and April 1994, the World Wide Web Worm received an average of about 1500 requests per day. In November 1997, AltaVista claimed it handled roughly 20 million queries per day. With the increasing number of users on the web, and automated systems which query search engines, it was estimated that top search engines will handle hundreds of millions of queries per day by this year.

Our back end, Intelligent Agent, is an application software program building on other search engines such as Google or Yahoo. In order to solve search problems automatically, it exchanges information by implementing agent communication protocols. An intelligent agent must maintain some autonomous and rational features. These features enable the agent to retrieve and analyse information without intensive human effort (Hu 1997). Intelligent Agent refines the resulting web pages from other normal search engines by using Boolean operations and proper keywords. Boolean operations include "logical AND", "logical OR", and "logical NOT". For example, "A AND B" means both A and B must be true. "A OR B" means either A or B is true can make the statement true. "NOT A" means it cannot equal to A. The agent continues working on these selected web pages and extracts the useful and related information from them. Then it inserts the information into the database intelligently. The major object of our intelligent agent is to improve the quality of web search engines. This is set against previous efforts for developing Internet searching mechanisms. In 1994, some people believed that a complete search index would make it possible to find anything easily. According to Best of the Web 1994 -- Navigators, "The best navigation service should make it easy to find almost anything on the Web (once all the data are entered)." However, the Web of 1997 is quite different. Anyone who has used a search engine recently, can readily testify that the completeness of the index is not the only factor in the quality of search results. "Junk results" often wash out any results that a user is interested in. In fact, as of November 1997, only one of the top four commercial search engines
could find itself (i.e., return its own search page in response to its name in the top ten results) (Brin and Page 1999).

One of the main causes of this problem is that the number of documents in the indices has been increasing by many orders of magnitude, but the user's ability to look at documents has not. People are still only willing to look at the first few tens of results. Because of this, as the collection size grows, we need tools that have very high precision (number of relevant documents returned, say in the top tens of results). Indeed, we want our notion of "relevant" to only include the very best documents since there may be tens of thousands of slightly relevant documents. This very high precision is important even at the expense of recall (the total number of relevant documents the system is able to return). There is quite a bit of recent optimism that the use of more hyper textual information can help improve search and other applications (Enguix 1998). In particular, link structure and link text provide a lot of information for making relevance judgments and quality filtering. The Intelligence agent makes use of both of these to satisfy our customer's particular market information needs.
3. METHODOLOGY

The key design factor in this project is to integrate three layered components of the Intelligent Agent model while developing each independently. The lower layer search result provides information to the upper layer interface. The upper layer user interface will use the results from the lower layer search to meet user database requirements.

3.1 Web Design Using FrontPage 2000

With the help of Microsoft FrontPage 2000, we first design the web server via a web site for users. This web site is composed of several pages that allow user access to the database. The web site has an index page, which tells the customers how to use this web site. It also contains pages that allow users to query, insert, and update the database. We also integrate the function of MUD into this web site. Users can browse any web page at any time. They may also query the database concurrently.

On the default home page, there are several functions for user to select: query, insert, modify, links and contact. The Query function allows the user to retrieve information existing in the current database based on record characteristics including keyword, resource or date. The Update function authorizes the user to add more data to the database. The Modify function lets the user change or even delete the information from the database. The Links function lists some interesting links that are directly related to the database. The Contact function tells the users with whom they can get in touch personal for database inquiries.

3.2 DBMS using Access

By applying the relational database model, we design the DBMS using Microsoft Access. This component uses functions of the Internet Database Connectivity (IDC) that is a dynamic link library (DLL). IDC connects to the Open DataBase Connectivity (ODBC). The following figure, Figure 2 illustrates these design characteristics.

During the thesis research, further building (and modifying) of the Structured Query Language (SQL) queries in the database management tool are carried out using Microsoft Access.
According to the properties of the data, these can be grouped into several tables. Each table contains a key that is distinctive and used to identify each individual record. To illustrate, consider the simple example taken from Nel (2001). Suppose we have a company that is organized into several departments. Each department has a few employees who work for it and has a manager who manages it. Further, the manager supervises the employees within that department. Each department also controls a few projects. Employees are working on projects. Every employee may have a few dependants. Moreover, employee, project, department, and dependant are related to each other. They also have some attributes what belong to each of them respectively. In other words, these attributes have some kind of relationships with them. Inside these attributes, we can always find one or more attributes to act as a "key".

Based on the description above, an illustration of this situation is illustrated by the relationship diagram of Figure 2 below. Underscored attributes in the figure are the "keys".

Figure 2: The relationship diagram of a company (Nel 2001)

where the building blocks of the figure are defined as follows:

- **Entity**
- **Weak Entity**
- **Relationship**
- **Address**
3.3 Intelligent Agent

The objective of this research is to build an intelligent search engine that can find web pages according to multiple keywords and Boolean operations provided by a user for a specific application database. The Intelligent Agent will select and also insert the related information directly into the user’s application database. The Intelligent Agent could search and retrieve automatically, periodically, or continuously and thus update the database accordingly. Some special features of the system developed in this research are as follows:

Run at the request of the user.

Dynamically match the changing Internet web pages.

Filter the information automatically, according to the existing database structure and keywords.

Select information and transfer records directly into the database.

This entire procedure described in sections 3.1 to 3.3 above is illustrated by the following diagram (Figure 3).
Figure 3: System integration application diagram showing components for Front End, DBMS and Agent-Internet Back End.
The Intelligent Agent itself is a software program written in the object-oriented language Visual C++. In order to provide an interface to DBMS, its basic principles are illustrated in the following diagram (Figure 4):

![Diagram](image)

**Figure 4: Web-Based Back End: The Intelligent Agent**

This interface will fulfill the responsibilities of connecting to a search engine server such as the Google server, retrieving the requested information, selecting the specific user-required records, and inserting these records automatically into the DBMS. Further, if some records are already in the database, it can filter them out and only insert new records to the database.
3.4  General issues for implementation

Now that the three components of the system are described, the key issue is how to integrate them into an effective system process. The following sections explain this integration process according to the steps of: (1) Internet search, (2) database insertion, and (3) database update.

3.4.1 Internet search

We first describe the basic principles of an Internet search process. The User submits the search request through the front end interface web browser on any PC. This request goes to the server where the Intelligent Agent resides. By applying the technology of Internet Server Application Interface (ISAPI), a dynamic linked library (DLL) file is written, and its name is "Hello.dll". Inside this dll file, there is a function called "Default" (Please refer to line 100 in the code provided in Appendix A.). This function can process the request and send it to the server of some search engine such as Google. The search engine then takes the user's request and carries out the Internet search. Google, for example, searches its own database through its own application technology (e.g.: Common Gateway Interface (CGI)). After it retrieves the results, the search engine will return records back to the server of the Intelligent Agent. These results are processed by the "Default" function of Hello.dll. When searching the Internet, the user first keys in the required search keyword and other database parameters. Next, the user clicks the button "Submit" and activates the Hello.dll function. This action calls the Default function inside Hello.dll. All the parameters are also passed to this function. The Default function calls another function "do_search" (Please refer to line 109 in the code provided in Appendix A.). By taking advantage of the parameters such as title, keyword, area, resource, and number, "do_search" links to the Google search engine. The result page of the Google search then returns back to the Default function. This function then displays the results page to the user. All these processes are transparent to the user. What the user is required to do is simply to enter the search parameters and to wait for the result page display. Default function simplifies the relevant search results by throwing out the irrelevant information such as subtitles to an identified main Website, and keeps only the user-preferred results. These final search results are then transferred back to the user as
a dynamic-produced Webpage. The principle behind the searching process is described in Figure 5 below.

Figure 5: Internet Search Process

The Intelligent Agent Search interface facility exists within and in support of a defined database of interest to the user. Because the entire application is based on the World Wide Web, the graphic user interface (GUI) has to be designed using web page software such as FrontPage 2000. This interface has the general appearance and functionality as illustrated in Figure 6. The characteristics of the GUI typically evolves with the database application, the users input and feedback (for an example see Figure 6). In this illustration, the user has input a keyword as a search title (e.g. "water temperature" in this illustrated case).
Figure 6: GUI of the Intelligent Agent

The user can enter this interface using a web browser (e.g., either Internet Explorer or Netscape Explorer). As noted in Figure 6, from this interface, the user keys in the search title request, selects the proper keyword, picks up an appropriate geographical area, chooses the appropriate resource, and also picks a max number of returning search results to be considered. The user now can click on the “Submit” button to process the request.

This interface actually links to the well-known search engine Google (www.google.com). Based on the user’s keyed input, Google searches the Internet and retrieves the results. For example, by using the words shown in Figure 6 (“water temperature”, “Oceans”, “Ecosystem” and “Species” in this specific user’s designed database example), Google obtains the following result page (Figure 7):
Figure 7: Google search result

From the Figure 6 request, there were actually 1430 results from this Google search. The Intelligent Agent generates a temporary file to store the HTML source code of these 1430 results. For each result, the Google output is composed of the following parts (as part of the standard Google output report as in Figure 7):

1. Page Title,
2. Description,
3. URL of Title, and
4. Indented Result.

Page Title is the first line of the result. It is a hyperlink which contains the URL and links to the actual web page. Description is an excerpt from the returned result page with the query terms we type in. These excerpts contain context in which the query terms appear on the page. If Google finds multiple results from the same web site, the most relevant result is listed first with
the other relevant pages from that same site indented below it. Thus, the most interesting parts for the database of the Google search result are only the Page Title, the Web page links and the Description. The others must be deleted in order to keep the final search result page clear and concise without overlap and without identifying multiple pages from the same web page reference.

The example of 1430 results may be too many for any user to consider. To make the Intelligent Agent more user-friendly, the web-based interface contains a combo box which allows the user to choose the amount of results he/she wants (10 in this case), refer to “Max number” in Figure 6. All these functionalities are done by the “default” function that is completely transparent to the use. The Agent will do all these interim tasks automatically. It links to Google, retrieves results, deletes the irrelevant information, and generates new pages and records in the user’s database according to user's preference. The following is the web page output to the user on the final generated search results (Figure 8).

Figure 8: Final search result
However, the structure of the search result of the Google search engine (Figure 7) may not remain unchanged. If the structure of the Google search result alters, the corresponding codes have to be modified in order to maintain consistency is retrieving information. Fortunately, no matter how the structure changes, the contents of every search result of any search engine must contain page title, brief page description, and the URL. Based on this fact, our Intelligent Agent can still reuse most code. This increases the reusability of the code segment. Furthermore, each component in the search result is separated by some specific html tag. Therefore, our task is to find the new position of these tags and then modify the code accordingly (see also discussion of this issue in Section 6.5).

3.4.2 Insertion into database

After the Intelligent Agent finishes processing the Internet search and fetching back the search results, the next step is to insert these results into the database. The following diagram illustrates the fundamental principle of this insertion process (Figure 9).

![Diagram of database insertion process]

Figure 9: Database insertion process
The user first chooses from among the returned search results those records that the user wants to insert into the database from the web browser in a PC. Then, he/she sends this request to the server in which the database physically locates using World Wide Web service. On the server side, this request will run the function of “insert” (Please refer to line 571 in the code appendix.) inside the file Hello.dll. As we mentioned above, this file is written using the technology of Inter Server Application Interface (ISAPI). The “Insert” function now inserts the entire user chosen results into the physical database according to the connection of ODBC. After all the selected files have been inserted, the system sends a message back to the client-end user and informs the user that the records have been successfully inserted. When function “Insert” is called, it also calls another function “do_insert” (please refer to line 583 in the code provided in Appendix A). This function receives all the parameters: title, keyword, area, resource, number of records, and selected records. These parameters then can be inserted into the database according to the corresponding fields.

For illustration purposes, as shown in Figure 8 above, the user chooses 3 interesting results by clicking on the check box. The user then clicks on the button “Submit”. At this point, the request right is sent to the database server. It activates the function “Insert” in the file Hello.dll. The “Insert” function then inserts these selected three records (Figure 8) into the database one by one. At the end of this insertion process, the user in the client-end is provided with a feedback page that lists all the records just added into the database (see also Figure 10 below). This entire procedure is done on the server side. Therefore, it is also transparent to the user. The user in the client-end needs no knowledge about how the database works. He/she only has to point and click on the check boxes of the search results and then click on the “submit” button to have the agent update the database with the newly chosen records.
Insertion Results Feedback

3 URLs selected
1 URL inserted

Following records already exist in database:

http://seawifs.gsfc.nasa.gov/OCean_PLANET/Opportunities/.....
Ocean Planet:opportunities changes... pressure, changes in water temperature, and decline in... England and Nova Scotia. It...
http://modshore.er.usgs.gov/project-maps/oracle/edailine/data.html
Gulf of Maine Observational Database... NOAA: Monthly Ecosystem Indices from... Sea Surface Temperature Images processed... More...
(4 species) Shellfish... New Brunswick, Nova Scotia... charts, water level...

Following records were inserted into database successfully:

http://octopus.sm.org/landscape/ecosystem/
Undersea Landscapes | The Bering Sea... tremendous variation in temperature and salinity... from southwest Nova Scotia around the...

Gulf of Maine Observational Database... NOAA: Monthly Ecosystem Indices from... Sea Surface Temperature Images processed... More...
(4 species) Shellfish... New Brunswick, Nova Scotia... charts, water level...

Figure 10: The feedback web page of an insertion into the database

However, users may still have to change the content of some record in order to meet the specific requirements of the database after the insertion. Under these circumstances, users can open the database and modify that individual record using the internal database functions.

3.4.3 Update of the database

After a period of time, some URLs in the database may be out-of-date or broken, i.e., are no longer valid pages. Thus, the database needs also to be updated. The database should delete all the useless records. In order to fulfill this feature, again, the technology of Internet Server Application Interface (ISAPI) is implemented. In the file Hello.dll we mentioned above, there is a procedure call “Update”. It is this function which does all the jobs for updating the database. This update functionality is an on-request process. The principle of updating the database is illustrated in Figure 11:
In the client-end, the user, who may be the database administrator, sitting in the front of a PC, wants to update the database after some time. This person opens a web browser such as Microsoft Internet Explorer, activates the web page and enters the username and the password, then click on the button “submit”. This request is sent to the server in which the database located. This request is sent using the World Wide Web service. All such menial updating tasks now can be done by the procedure “Update”. This procedure checks every record in the database. If there is any link that is broken or has an error, it deletes this record automatically. It repeats this until the last record has been reviewed. After updating the entire database, it returns a successful message to the client-end.

The following real world applications will further describe the use of the Integrated Database Search Agent system. First, the user clicks on the button “update” at the down left corner from the main web page. It brings a pop-up window (Figure 12) which have two textboxes there, one for username and the other for password. Then, the database administrator or any other authorized user can just key in valid username and the password, and click on the button of submit. The database will be clean up after a while. For example, there are 100 records in the database originally, but 15 out of them are broken or cannot be accessed any longer. The procedure “Update” in Hello.dll can check all 100 records and pick up the 15 useless ones. These
15 ones then can be thrown out of the database. The remaining 85 records will still be kept in the database until another "Update" procedure is carried out. Moreover, as a research issue, this update process can also be improved. It should be done automatically and periodically without the user requesting it.

![Update Database](image)

**Figure 12: The pop-up window of update**

The following section presents specific applications of the Intelligent Agent model, their development and their use.
4. APPLICATION AND CASE STUDY

This chapter presents 2 distinct applications of the Web-based system integrations and the Intelligent Agent applications. These applications are:

1. **FRCC EcoData** – database and facility for amassing data on marine ecosystems information and analysis in Atlantic Canada

2. **E-commerce and the SUV market** – database for information on the marketing, sales, availability and use of sport utility vehicles (SUV) in Canada

The three components in our Intelligent Agent model discussed in Chapter 3 are actually independent components developed separately and interchangeably for each application. The components for the two applications presented here were developed individually. Compared to the traditional database model, the functions are applied in exactly the same way for each specific application. The central dynamic database management system defines the database. It specifies the data type such as integer, string, date, and so on. It also specifies the structure (e.g. hash function and/or indexing) and the constraints (e.g. primary key, foreign key, and/or null). The back end can fulfill the function of constructing the database, including data collection and data building. It searches the information directly from the Internet and then filters out redundant non relevant information. This is done automatically by using Intelligent Agent. The third component, the front end, manipulates the database. It is a web site consisting of a series of linked web pages that can carry out various user interface tasks. This web site can be used for query and to generate reports routinely. We examine below the particular application of the Intelligent Agent Integrated System for each of the two applications described above.
4.1 FRCC EcoData Case Study

4.1.1 FRCC Background (FRCC 2001)

The Fisheries Resource Conservation Council (FRCC) was founded in 1993. Its task is to coordinate the fishing industry and the scientific and academic expertise in the analysis and management of the commercial fisheries on Atlantic Canadian ground fish stocks. The mandate of the Council is to provide regular recommendations to the Minister of Fisheries and Oceans on the issues such as total allowable catches (TACs) and other conservation measures for the Atlantic ground fish fishery. The Council also gives provides suggestions on the scientific research and assessment of the fish resources and on general issues of ecosystem considerations effecting the status of the ground fish stocks.

A small Secretariat in Ottawa provides support to the Council. This Secretariat is responsible for most administrative work, web page maintenance, report preparation, and public liaison. There are 15 members of this Council. Appointed by the Minister of Fisheries and Oceans, Council members include individuals from the fishing industry and scientific and academic fields of study related to the fisheries.

4.1.2 EcoData Web Application

One objective of the FRCC is to understand better the effects on groundfish due to general ecosystem considerations (changes in temperature, area ice cover, predator-prey relationships, etc.), and to help the Minister of the Department of Fisheries and Oceans (DFO) with recommendations on ongoing scientific research into these effects. Given that there is currently no unique source of ecosystem information, the FRCC is interested in constructing a database of ecosystem information sources for Atlantic Canada. The FRCC is seeking to develop a database that could coordinate and organize available information about marine ecosystems. This database of information is directly related to the FRCC mandate of applying an “ecosystem approach” to a sustainable management of the marine environment. The FRCC front end and the DBMS, known as EcoData, have been developed, tested and used by select members of the FRCC during the course of this research. The web-based Intelligent Agent has been prepared and tested and is awaiting final approval for use on the official FRCC website. The website is currently found at
the URL: http://www.ecodata.management.uottawa.ca/default.htm. Users are permitted access on provision of a username and password.

The home page of the FRCC EcoData website provides basic information about the database system. Figure 13 below illustrates the home page.

Figure 13: FRCC EcoData Project Home Page

The EcoData system is used to collect information about natural marine ecosystem in Atlantic Canada primarily from sources posted on the Internet. This information is inserted into corresponding records and tables in the database. There are several tables in the database. Each one covers different kinds of information. The relationships existing between each information type are linked. Users can retrieve information based on the desired relationship. Just like the MUD system, multiple users, e.g., various members of the FRCC, DFO officials supporting the FRCC, and the FRCC Secretariat staff can access the database at the same time. Each user can
query and retrieve different information in the database concurrently. The following is the query page of the website:

![Query the Database](image)

**Figure 14: The Query Web Page of FRCC EcoData Project**

After the user enters the keyword in the textbox, which is beside “Input word in title” (for example, the Figure 14 example uses “water” as the input in this case), the user then clicks on the button “Submit”. The result of this query yields all records in the database that have the word string “water temperature” in their record title. The result page shows the query result in Figure 15. Furthermore, the user can also query the database by choosing another method such as by (2) record number and title, (3) keyword, (4) geographical area, (5) resource name, and (6) date that record was added to the database.
The database administrator (DBA) updates or inserts any attributes in any tables belonging to this database directly from the web site. Aside from doing the updating and insertion manually, it is proposed that the DBA will also regularly run the Intelligent Agent software that would automatically search, retrieve and update the entire database with records corresponding to a particular search topic. This regular procedure will ensure that exogenous changes to linked websites are updated as the information in the information age changes frequently. Therefore, in order to keep all records in the database as not being obsolete, the DBA can thus take advantage of the Intelligent Agent software to automatically maintain and update the database. Moreover, through the advanced functionality of our Intelligent Agent, users can always get the latest information from our website database. This reduces search efforts that can be otherwise time-consuming.

4.1.3 Intelligent Agent in FRCC EcoData

As described in the above subsection 4.1.2, the user of this system should run the Intelligent Agent regularly in order to update the database automatically. The following paragraphs explain this updating process for the case of the EcoData application.
The user accesses the web page of the Intelligent Agent (see also Figure 6 – GUI of the Intelligent Agent) by clicking on the button “Search” from the EcoData home page (Figure 13). Suppose the user now enters “air pressure” as the search title, “biogeography” as the keyword, “oceans” as the geography area, “species” as the resource, and “10” as the maximum number of records to select as the fields for Figure 6. After clicking on “Submit”, a page of search results containing 10 records is fetched back to the user’s browser screen (as in Figure 8 – Final Search results). The user can then select the most interesting and relevant records for automatic insertion into the database.

In order to retrieve and view the results, the user can simply click on “Query” from the EcoData function frames, e.g., on the left of the home page and do the query (see also Figure 14). By using “Search records entered since past date”, the following web page can be retrieved which show all the matched records inserted by the Intelligent Agent (Figure 16).

Figure 16: Results of Query by Date
From this query page, the user can then view any particular record by just clicking on the record ID. For example, the following diagram illustrates the record if the user clicks on record ID 118 (Figure 17).

**Figure 17: A Specific Record**
4.2 E-commerce Case Study

Carrying out day-to-day business transactions on the Internet has become more and more an everyday occurrence that will undoubtedly continue to grow in popularity. This case study presents a database and facility for the collection of data for a particular target market of popular motor vehicles. This case illustrates the use of an Internet-based with a specific product in mind that is designed to assist users investigating and searching Internet sources before making a decision on final purchase of a "big ticket item" such as an automobile. At the same time, this case illustrates the general versatility of the front end, database, and Intelligent Agent search and retrieval system for the case of e-commerce. The following sections describe how the Intelligent Agent is applied to this target market analysis.

4.2.1 Intelligent Agent in E-commerce

During the information age, electronic transactions or "e-commerce" is one of the basic commercial tools. More and more people feel comfortable with doing business on the Internet. People can make a deal without walking out of the door. They can buy or sell products by just clicking the mouse. Therefore, Internet, or Electronic Commerce, becomes a very useful marketing utility. Moreover, people are now becoming used to analyzing business opportunities based on the vast sources of information that is available from an Internet search. Thus this exercise of informed shopping and buying can benefit from a powerful searching tool – an application of the Intelligent Agent.

Electronic Commerce is a virtual market on Internet where secure electronic transactions take place. The providers and customers of the services or resources meet in this virtual market and do their business activities there. Therefore, a well-designed integrated regulation of user interface, DBMS and Intelligent Agent can make the application of Electronic Commerce more attractive to users. It can also provide more efficiency and convenience to current and future E-Commerce application (Hu 1996)

4.2.2 Application in E-Commerce: Buying a SUV

As a specific example for e-commerce, we apply the Intelligent Agent to Sport Utility Vehicle (SUV). Since Henry Ford invented the combustion engine, Model T, the automobile has
evolved into the high-tech versions of today’s Sport Utilities Vehicles. SUV owners possess a sort of a passion for their vehicles. It becomes a fashion statement of transportation similar to clothing fashions where everything old becomes new again.

The first SUV came from a ‘depot hack’ that was a vehicle that functions as a taxi today. In the 1920’s, the depot hack was also called a ‘carryall’ or ‘suburban’ which is the origin of the name of the longest running SUV model, the Suburban. Another long-lasting SUV model is the Jeep Wagon, which was introduced by Wagoneer in 1963. At that time, it was advertised as a utility vehicle for the family. Early SUVs were meant to be practical and a means to ‘carry all’. In the 1970’s, inflation, emissions control, high gas price and the death of big engines had a negative impact on the popularity of the SUV. All these made the small fuel-efficient Japanese cars became popular. But in the 1980’s, better fuel prices and low interest rates allowed people to consider more recreational type vehicles.

Today, more and more people enjoy the SUV and every automaker in the world offers no less than 2-3 different classes of these vehicles. With the proliferation of SUV and classes of SUVs competing in similar markets, consumers may easily be confused about their options and opportunities. They would clearly benefit from an efficient search tool to help satisfy their information gaps and to express their individual interests. The Intelligent Agent can help fulfill the requirements of efficient information gathering about the large amount of data on SUVs now available on the Internet.

4.2.3 Some SUV Issues

Along with the popularity of the Internet for ease of communications and as passive provider of popular “pop” information, more and more people are using the Internet to do active information searches. In order to make the Intelligent Agent meet the requirements of most users, we have to know what potential consumers of the SUV are concerned about. Keeping this in mind, we do an extensive Internet search and look at several SUV web sites. Among the preoccupations found during this analysis, we noted that consumers (and marketers) of SUVs are attracted to: (1) SUV operating costs and (2) SUV vehicle safety were the two main issues.
SUV operating costs. Buying a sport utility vehicle is more and more attractive to consumers. Thus, automakers can charge relatively high prices for SUVs in order to increase profits. Further, running an SUV also involves lots of extra expenses. Generally speaking, a SUV costs more for repair, gasoline, maintenance, and insurance than other passenger cars. The liability rates on SUVs are increased by some big national insurance companies, while these companies are lowering the rates for other car owners. For instance, Farmers Insurance gives a discount of 5 to 19 percent to passenger car owners, but it is charging 5 to 16 percent more liability rates on SUV owners. In addition to these, the burden for taxpayers is also increasing due to the growth of sales in SUV. This is because a loophole in the gas-guzzler tax provides automakers an encouragement to produce fuel-efficient vehicles. The more efficient are the vehicles, the less taxes they pay. But SUVs pay no tax. Thus, the automakers have potentially a free ride by manufacturing and marketing inefficient vehicles. And unfortunately, this tax saving is not passed on to the buyers hands; it enters into the automakers’ pockets (Suv.org 2001).

SUV vehicle safety. Whenever driving a vehicle, safety is very important. Given the design and marketing of the SUV for rugged, all-terrain use, this is especially a concern for SUV owners. SUVs are different from passenger cars, mini-vans, pickup trucks or sport cars. There are several factors that are of concern in the safety of SUVs: weight of vehicle, height, width, personal response, seat belts, ABS, and four-wheel drive / all wheel drive (Suv.com 2001).

4.2.4 Web implementation of Intelligent Agent in SUV

As the first step mentioned above, we need to build up a front end. It is a web site that allows user to query the database.

The home page for the SUV web site application is presented in Figure 18 below.
SuVdata is a Database system developed for collecting online information of sport utility vehicles in North America. This Database is for users who are interested in SUVs, to acquire information, to consider purchase options, including vehicle prices of different models of SUVs on the market, insurance costs, and new and used vehicle availability.

Figure 18: The Home Page of SUV Application

The user can query the database by clicking on the button of “Query”. Similar to the EcoData case, it brings a web page to the user. The user selects keywords and a few relevant parameters as illustrated in Figure 19. Clicking on the command button “Submit” activates the query request. As before, the result will retrieve the matching records from the database. These records are listed on the web page and shown to the user.
Figure 19: The Query Web Page of the Intelligent Agent

The database is designed using relational structure to link several tables together. This database has three tables which are related to each other. The first table is the "record" table. It has 8 fields: RecordID, title, URL Description geographical area, resource, citation, and date. All these are the properties of a search result. When the user searches for SUV information, he/she may be interested in the internet URL of that SUV, the geographic area of that vehicle, and some more description such as the demand, the sales, and so on of that particular SUV. Thus, the table is designed according to the user requirements and major characteristics of areas of interest. RecordID is the key of this table. That means each record in this table has a unique RecordID. The second table is "have_keyword". Query searches for example can be triggered according to some keyword provided. So the system needs a keyword table that is linked to the "record" table by RecordID. The third table is "keyword". It has two fields: keyword and detail. Field keyword is the key of this table. As the name of that field speaks, detail gives the detailed explanation of keyword. It tells people what a particular keyword means in this database. The field of keyword
is the relation between table "keyword" and table "have_keyword". Moreover, as a user of the Intelligent Agent, he/she does not have to know anything about this database. It is transparent. The following diagram (Figure 20) illustrates how these tables link together.

![Diagram of database structure](image)

**Figure 20: The Database of SUV**

On the home page of this web site (refer to Figure 18), there is a "Search" button. It brings the user to a search page that is similar to Figure 6. On this page, user enters a few parameters and click on Submit. This action will activate the Intelligent Agent and ask Google to do the searching job for the user. According to the processing procedures which were discussed above for the FRCC EcoData case earlier in this thesis (Section 4.1.3). The Intelligent Agent can retrieve the relevant information back to the user. Aside from showing the search results to the user, the Intelligent Agent also inserts user-selected records into the database automatically. The following diagram (Figure 21) shows an example record which the user viewed by querying the database after the insertion by the Intelligent Agent:
Figure 21: A Query record of SUV
5. RESULTS

5.1 Application

The Intelligent Agent system actually can be applied everywhere when a database is required to build and records added. For example, applications in university information program databases for student applicants, industry information databases on suppliers and prices of key product needs, commerce information databases on company performances and so on. The Intelligent Agent’s usage is just like Yahoo or Google, but it does a much more focused Internet search in conjunction with a well-defined database and recurrent information need due to it’s stronger functionalities and the fundamental database design. All users can use it to search the Internet, analyze and filter the search results and then automatically include the records of the appropriate information to the user’s database.

When using the Intelligent Agent, users can flexibly adjust keywords and the parameters for the special kind of information they are seeking. It provides a user-friendly interface for clients, and then it run automatically and efficiently via the Internet. The information stored in the database can be updated easily and frequently such that the user can always have the access to the latest information. This is particularly useful when the web pages cited change or obsolete.

5.2 Reports about the search result

After each search, the final result that the user really needs and preselects will be inserted into the database and made available for the further queries of the database interface. Furthermore, Intelligent Agent will generate a temporary log file (see also Figure 7) that stores all the intermediate results for every action taken by the agent. Thus, the user can trace back the information and see the entire procedure if necessary.

5.3 Database Size

Due to the efficient and logical design of the relational model, all the tables in the database are connected by using the key attributes. They are also controlled by the internal DBMS. Thus, it doesn’t require a very large disk memory size to store the entire database.
5.4 Ecosystem database and use — manual entry vs. automatic

The EcoData database application was originally developed to be updated directly by the database administrator (DBA) by using the front end – the web site, and searching out website independently. During the use of this system, user complained of the tedious task of needing to insert each new record. In contrast to this inefficient recording process, the Intelligent Agent was found to be much more receptive by the select users by enabling them to do the retrieval and insertion tasks automatically. Moreover, it also saved time and avoided human mistakes of duplication of records. In any case, it was found that user's still used the “modify” function to update retrieved records and to record their own notes on particular websites of key interest including attaching other keywords, etc. This step is still necessary, especially when the information is very up-to-date and in cases for records that are not necessarily available on the Internet, e.g., some academic publications.
6. FUTURE RESEARCH CONSIDERATIONS

Although this software program works fine for the above two cases, there are still some issues with the use of the Intelligent Agent and the database system. Before the design of the Intelligent Agent described in this thesis, this research began with the notion of constructing an active "Intelligent Spider". In the future, if the working environment requires it, the agent may have to upgrade to a "spider". Moreover, such a system also applies to the other types of application referred to above. In this program, the Intelligent Agent implements the technology of Internet Server Application Interface (ISAPI). This interface also requires further review and consideration for search and retrieval systems in general.

6.1 Spider vs. Agent

A "spider" is an application program which can visit different web sites and retrieve information from these web sites for a search engine. It can usually visit many sites in parallel at the same time. An agent is a program that can gather information or perform some other services on a basis of user requirement by taking advantage (as is done in this thesis) of some other direct search engine. The agent is not required to visit any other web sites except for the web site of the search engine, e.g., the Google site referred to in the applications of this thesis. Thus, an agent is totally different from a spider. They are based on different design principles. Basically, the spider is itself a prototype of a search engine; and the agent is merely a program that connects to a search engine and uses its functionality based on user requests.

6.2 Other applications

Aside from the two cases we discussed above, this Intelligent Agent can also be applied into other fields such as industry, government, or even academic information needs. For example, in an University, in order to prepare the teaching material for a course delivery, the professor may want to do some specific Internet search on a course-related topic and store the results for later reference and use. The professor does not have to do an extensive Internet search. He / she can simply enter some keywords and the other parameters into our Intelligent Agent associated with the course database. The agent can then search the Internet and present selected options for inclusion in the database.
The technology development process for describing this new application is as follows. In the client machine, which has IIS installed, a new database is established. Next, develop the html page codes for the front end and complete the Internet data connector scripts for the query operations. The configuration of ODBC for the ACCESS database also has to be done simultaneously. According to the Internet data search for which the new application is required, the coding of the back-end (output formats) must be modified as well. When all these steps are done, the new application can be uploaded to the Internet and put into practice at once.

6.3 ISAPI

Since more and more small-to-medium sized enterprises have established themselves in the last decade, these companies require their own “intranets” to make connect to the World Wide Web (Internet). Accompanied by the growth of intellectual and economic forces, businesses in all fields are driven to exploit the technology of Intranet-Internet links in their corporate environment. The Internet Server Application Programming Interface (ISAPI) is thus developed in order to meet the demand of this technology growth. ISAPI is an interface to hypertext transport protocol (HTTP) servers. By using this interface, developers can extend web servers and provide an increased degree of functionality. ISAPI extensions and filters enable servers to become more powerful and to allow these servers to have the functions that the Common Gateway Interface (CGI) or some other competing interfaces could not fulfill. The high-performance and scalable ISAPI solution allows developers to create web applications dynamically.

6.4 Security

This Intelligent Agent system and applications developed in this research are basically working on the public World Wide Web. Therefore, security becomes a fundamental issue under this situation. In order to prevent the attack of Internet hackers on database holdings, security has to be taken into account. As a simple solution here, the application web sites are design for authorized users only, including the database administrator and individuals to whom personalized IDs and system passwords have been issued. Before entering the web site, a pop-up window will show up to ask the user input the username and password. This is by no means a
“fool-proof” security mechanism, but it is initially effective. More sophisticated security would also include virus protection devices, and firewalls especially in Intranet-Internet applications.

6.5 Interpretation of the Search Engine Formats

This research takes advantage of the Google search engine. From the search result page, it is obvious that every result is composed of several components, such as page title, description, URL of the result, and so on and so forth. The information that is going to be inserted into the database is stored here and we have to retrieve the required information for the database from this result. The following diagram illustrates the interpretation of the search result.

Figure 22: Interpretation of the search result (Google 2002)

Based on the contents described in the figure above, items J, K, and N are the interesting information we want. In the program, for example, the codes we use to handle J and N are as follows (see also line 393 to 402 of code provided in Appendix A):
This code can insert the URL and the title of the result into different string arrays and then insert them into the database. However, the format of this search result may change after a certain period of time. But the principle of the insertion is still the same. What we have to do is find the html tag for the relevant information and modify the code slightly. For example, if there are five more characters in the html code between items J and N, the fifth line in the above code may change from “t=s+1” to “t=s+6”.

This thesis uses the Google search engine and Google search results formats to apply automatic insertions into the “home” database. Different search engines (e.g., Yahoo, AltaVista, Opentext, MSN, etc.) use unique results formats. Thus, using a different search engine would require modification of the interpretation of the format for seizing and inserting data records.

Secondly, it may be the case that some search engines occasionally modify their public search record formats in order to avoid the application of intelligent agents, such as proposed here. In such cases, a modification of the Intelligent Agent would be required to adapt to the format changes.

6.6 Local Database Search or Internet Search

In this Intelligent Agent system, the user usually queries the local database while searching for desired interesting results. This user will further search the Internet if the results in the database do not satisfy the user requirements. Currently, this process is carried out manually by the user. In order to be more convenient, this search process could be made automatic. For example, when the user has finished the local database queries, the system can automatically activate the Internet search for the same keyword request.
7. BIBLIOGRAPHY


53


http://whatis.techtarget.com/definition/0,289893,sid9,gci212609,00.html


APPENDIX A

SOURCE CODE OF MAIN FUNCTIONS FOR THE INTELLIGENT AGENT

Description:

Intelligent agent, the back-end of this system, is written by C++. The integrated development environment is Microsoft Visual C++ 6.0. After compiling and debugging, the code runs on the Internet Information Server with a Windows 2000 or Windows XP platform.

This source code contains documentation and comments throughout the code to describe code functionality.
VOID CSUVEXTENSION::DEFAULT(CTHTTPSERVERCONTEXT*, CHAR*, CHAR*, CHAR*, CHAR*) .......................................................... A-4

CSTRING CSUVEXTENSION::DO_SEARCH(CSTRING&, CSTRING&, CSTRING&, CSTRING&, INT) .................................................................................................. A-5

VOID CSUVEXTENSION::INSERT(CTHTTPSERVERCONTEXT*, CHAR*, CHAR*, CHAR*, CHAR*, CHAR*, CHAR*) .................................................. A-13

CSTRING CSUVEXTENSION::DO_INSERT(CSTRING&, CSTRING&, CSTRING&, CSTRING&, INT&, CSTRING&, CSTRING&) .................................. A-13

VOID CSUVEXTENSION::MY_UPDATE(CTHTTPSERVERCONTEXT*, CHAR*, CHAR*) ................................................................................................. A-17

INT CSUVEXTENSION::RECORD_NUM(CSTRING&) ................................................. A-18
// SUV.CPP - Implementation file for your Internet Server
// suv Extension

#include "stdafx.h"
#include "suv.h"
#include "CRobot.h"
#include "CRobotInternet.h"

#include <stdlib.h>
#include <time.h>

CCriticalSection cs, cs2, cs3;
CString path = "C:\vc_temp\suvs\";
CStringArray tmp_file_list;
bool update = true;

// The one and only CWinApp object
// NOTE: You may remove this object if you alter your project to no
// longer use MFC in a DLL.

CWinApp theApp;

BEGIN_PARSE_MAP(CSuvExtension, CHttpServer)
    // TODO: insert your ON_PARSE_COMMAND() and
    // ON_PARSE_COMMAND_PARAMS() here to hook up your commands.
    // For example:

    DEFAULT_PARSE_COMMAND(Default, CSuvExtension)
    // ON_PARSE_COMMAND(Default, CHelloExtension, ITS_EMPTY)
    ON_PARSE_COMMAND(Default, CSuvExtension, ITS_PSTR ITS_PSTR ITS_PSTR)
    ITS_PSTR ITS_PSTR
    ON_PARSE_COMMAND_PARAMS("r_title r_key r_area r_res number")
    // DEFAULT_PARSE_COMMAND(Default, CHelloExtension, ITS_PSTR ITS_PSTR ITS_PSTR
    ITS_PSTR ITS_PSTR
    // DEFAULT_PARSE_COMMAND_PARAMS("r_title r_key r_area r_res number")
    // DEFAULT_PARSE_COMMAND(Default, CHelloExtension, ITS_PSTR)

    // DEFAULT_PARSE_COMMAND_PARAMS("r_title r_key r_area r_res number")
    ON_PARSE_COMMAND(insert, CSuvExtension, ITS_PSTR ITS_PSTR ITS_PSTR ITS_PSTR
    ITS_PSTR ITS_PSTR ITS_PSTR)
    ON_PARSE_COMMAND_PARAMS("title keyword area resource number filename
    selected_items")
    ON_PARSE_COMMAND_PARAMS(my_update, CSuvExtension, ITS_PSTR ITS_PSTR)
    ON_PARSE_COMMAND_PARAMS("username passwd")

END_PARSE_MAP(CSuvExtension)
CSuvExtension theExtension;

BOOL CSuvExtension::GetExtensionVersion(HSE_VERSION_INFO* pVer)
{
    // Call default implementation for initialization
    CHttpServer::GetExtensionVersion(pVer);

    // Load description string
    TCHAR sz[HSE_MAX_EXT_DLL_NAME_LEN+1];
    ISAPIVERIFY(rightGetString(AfxGetResourceHandle(),
        IDS_SERVER, sz, HSE_MAX_EXT_DLL_NAME_LEN));
    _tcsncpy(pVer->lpszExtensionDesc, sz);
    return TRUE;
}

BOOL CSuvExtension::TerminateExtension(DWORD dwFlags)
{
    // extension is being terminated
    // TODO: Clean up any per-instance resources
    return TRUE;
}

void CSuvExtension::Default(CHttpServerContext* pCtx, char* title, char* r_key, char* r_area, char* r_res, char* number)
{
    CString title(r_title);
    CString keyword(r_key);
    CString area(r_area);
    CString resource(r_res);
    CString return_page;
    int num = atoi(number);
    return_page = do_search(title, keyword, area, resource, num);
    //produce the result page to user
    StartContent(pCtx);
    WriteTitle(pCtx);
}
*pCtxt << return_page;
EndContent(pCtxt);
}

//a help method to do keyword search
//it is the real one to communicate with google search engine
CString CSuvExtension::do_search(CString& m_title, CString& m_key, CString& m_area,
CString& m_resource, int Max_records)
{
    CRobotInternet internet;
    internet.m_bReadFromCache = true;
    internet.m_bWriteToCache = true;
    CString sData, sData_mem, sURL, sHTML, sErrMsg, test, test1, test2, sm_Html, sm_Html1;
    mytest1 = "";
    CStringArray URLs, Titles, Contents;
    CString content_tmp;
    int count = 0;
    int sum_num = 0;
    int num = 0;
    char buff[30];
    bool flag = false;
    int nResult, jj;

    sData = "q=";
    sData = sData + "%22";
    test4 = m_title;
    test4.Replace("", "+");
    sData = sData + test4;
    sData = sData + "%22";
    sData = sData + "+";

    sData = sData + "%22";
    test4 = m_area;
    test4.Replace("", "+");
    sData = sData + test4;
    sData = sData + "%22";
    sData = sData + "+";

    sData = sData + "%22";
    test4 = m_key;
    test4.Replace("", "+");
    sData = sData + test4;
    sData = sData + "%22";
    sData = sData + "+";

    sData = sData + "%22";
    test4 = m_resource;
    test4.Replace("", "+");
    sData = sData + test4;
    sData = sData + "%22";
    sData = sData + "\&num=100";
sData = sData +
"&btM=Google+Search&as_eq=&as_oq=&as_eq=&ir=&as_qdr=all&as_occt=any&as_dt=i&as_sitesearch=";

//
sData = sData +"&start=
char = safe=off&start=";

//
sData = sData + itoa(count, buff, 10);

sData = sData + ";

//
sData = sData + ";
sURL = "www.google.com/search?"+sData;

//
CFile my_file("C:\m_result.htm". CFile::modeCreate | CFile::modeWrite);

if(internet.httpGet(sURL, sHTML, nResult, sErrMsg))
{
    if(sHTML.Find("did not match any documents")!=-1)
    {
        sm_Html = "<html><\n<title>Results Page</\n<title><\n<body
background="/backgrnd.jpg"\n<center><\h1>Search finished without any\nresult</h1></\center></\n<body></html>";
        return sm_Html;
    }
    sHTML.Delete(0, sHTML.Find("<p>") );
    if(sHTML.Find("/nav_next.gif")!=-1)
    {
        flag = true;
    }
    sHTML.Delete(sHTML.Find("<div class="n"), sHTML.GetLength()-sHTML.Find("<div
class="n"),
    sHTML.Delete(sHTML.Find("<br clear=all=""), sHTML.GetLength()-sHTML.Find("<br
clear=all="") ;
    }
else
    {
        sHTML.Delete(sHTML.Find("<br clear=all=""), sHTML.GetLength()-
        sHTML.Find("<br clear=all="") );
    }
    if(sHTML.Find("<br clear=all="")!=-1)
    {
        sHTML.Delete(sHTML.Find("<br clear=all="") , sHTML.GetLength()-
        sHTML.Find("<br clear=all="") ;
    }
    if(sHTML.IsEmpty()==0)
    {
        int e, f;
        CString tmp;
        sHTML.Replace("n", "");
        sHTML.TrimLeft();
        sHTML.TrimRight();
        e = sHTML.Find("<blockquote>");
        while(e != -1)
        {
            f = sHTML.Find("</blockquote>");
            tmp = sHTML.Mid(e, f+13-e);
            sHTML.Delete(e, tmp.GetLength());
            sHTML.TrimLeft();
            sHTML.TrimRight();
e = sHTML.Find("<blockquote>");

CString xixi = "here";
return sm_Html;
}

sm_Html = "";
my_file.Write(sHTML, sHTML.GetLength());

sum_num += record_num(sHTML);
if(sum_num > num){
    flag = false;
}

sm_Html = sm_Html + sHTML;
if(sm_Html)
    my_file.Write(sData, sData.GetLength());

while(flag)
{
    flag = false;
    count += 100;
    sData.Empty();
    sData = sData_mem;
    sData = sData + itoa(count, buff, 10);
    sData = sData + "&sa=N&filter=0";
    sURL = "www.google.com/search?" + sData;
    sHTML.Empty();
    if(internet.httpGet(sURL, sHTML, nResult, sErrMsg))
        if(sHTML.Find("/nav_next.gif") != -1)
            flag = true;

    sHTML.Delete(0, sHTML.Find("<p>"));
    sHTML.Delete(sHTML.Find("<br clear=all>")), sHTML.GetLength();
    sHTML.Find("<br clear=all>");
    /*
    if(flag)
    {
        
    }
    
    sHTML.Delete(sHTML.Find("<div class=n"), sHTML.GetLength());
    sHTML.Delete(sHTML.Find("<center>")), sHTML.GetLength();
    
    else
    {
        sHTML.Delete(sHTML.Find("<br clear=all>")), sHTML.GetLength();
        sHTML.Find("<br clear=all>");
        sHTML.Delete(sHTML.Find("<p><div class=n">".sHTML.GetLength()-sHTML.Find("<p><div class=n">"));
        /*
        my_file.Write(sHTML, sHTML.GetLength());
        if(sHTML.IsEmpty()) == 0)
        
    
        A-7
int e, f;
CString tmp;
sHTML.Replace("n", "");
sHTML.TrimLeft();
sHTML.TrimRight();
e = sHTML.Find("<blockquote>");
while(e != -1)
{
    f = sHTML.Find("</blockquote>");
    tmp = sHTML.Mid(e, f+13-e); // change 12 to 13
    sHTML.Delete(e, tmp.GetLength());
    sHTML.TrimLeft();
    sHTML.TrimRight();
    e = sHTML.Find("<blockquote>");
}
CString xixi = "here";
return sm_Html;
}

sum_num += record_num(sHTML);
if(sum_num >= num){
    flag=false;
}

sm_Html = sm_Html + sHTML;
}
else
{
    return sErrMsg;
}

/// CString tmp = itoa(count_buff, 10);
/// my_file.Write(sURL, sURL.GetLength());
/// my_file.Close();
/// test3 = test3 + test;

}
else
{
    sm_Html = "<html><title>Results Page</title><body
background="/bgnd.jpg">"<center><h1>Network error! The job of searching can not be
finished.<h1><center></body></html>";
    return sm_Html;
}

/* if(sm_Html.IsEmpty()==0)
{

    int e, f;
    CString tmp;
    sm_Html.Replace("n", "");
    sm_Html.TrimLeft();
    sm_Html.TrimRight();
e = sm_Html.Find("<blockquote>");
    while(e != -1)
```c
[Code Snippet]
```

A-9
tmp.TrimRight();
}
t = tmp.Find("<a href="") + 8;
s = tmp.Find(">", t);
CString url_tmp = tmp.Mid(t, s-t);
URLs.Add(url_tmp);
t = s + 1;
s = tmp.Find("</a>"", t);
CString title_tmp = tmp.Mid(t, s-t);
title_tmp.Replace("<b"><", "" "");
title_tmp.Replace("</b>"", "" "");
Titles.Add(title_tmp);
CString check4 = check1 + itoa(index+1, buff, 10) + check2 + 
itoa(index+1, buff, 10) + check3;
tmp.Insert(3, (LPCSTR)check4);
c = 0;
d = tmp.Find("<br>"", c);
cntent_tmp = " ";
jj = 0;
while(d != -1)
{
    tmp1 = tmp.Mid(c, d+4);
    tmp.Delete(c, tmp1.GetLength());
    tmp.TrimRight();
    tmp.TrimLeft();
    tmp1.TrimLeft();
    tmp1.TrimRight();
    if((tmp1.Find("File Format:")==-1) && (tmp1.Find("Your browser may not")==-1) && (tmp1.Find("Similar pages")==-1) && (tmp1.Find("Category:")==-1))
    {
        mytest1 = mytest1 + tmp1;
        if(jj != 0)
            content_tmp = content_tmp + tmp1;
    }
    d = tmp.Find("<br>"", c);
    jj++;
}
content_tmp.Replace("<br>"", "" "");
content_tmp.Replace("<b><"", "" "");
content_tmp.Replace("</b>"", "" "");
content_tmp.Replace("<font size=-1>"", "" ");
Contents.Add(content_tmp);
b = sm_Html1.Find("<p>"", a+3);
index++;  
num += 1;
if(num == Max_records)
{
    goto there;
}
sm_Html1.TrimLeft();
sm_Html1.TrimRight();
CString check4 = check1 + itoa(index+1, buff, 10) + check2 + itoa(index+1, buff, 10) + check3;
sm_Html1.Insert(3, (LPCSTR)check4);
c = 0;
d = sm_Htm11.Find("<br>", c);
jj = 0;
index++;
while(d != -1)
{
    t = sm_Htm11.Find("<a href="", t);
    s = sm_Htm11.Find(""", t);
    CString url_tmp = sm_Htm11.Mid(t, s-t);
    URLS.Add(url_tmp);
    t = s + 1;
    s = tmp.Find("</a">", t);
    CString title_tmp = tmp.Mid(t, s-t);
    title_tmp.Replace("<b>", """);
    title_tmp.Replace("</b>", """);
    Titles.Add(title_tmp);
    tmp1 = sm_Htm11.Mid(c, d+4);
    sm_Htm11.Delete(c, tmp1.GetLength());
    sm_Htm11.TrimLeft();
    sm_Htm11.TrimRight();
    tmp1.TrimLeft();
    tmp1.TrimRight();
    content_tmp = "";
    if((tmp1.Find("File Format:")) == -1) &
    (tmp1.Find("Your browser may not")) == -1) &
    (tmp1.Find("Similar pages") == -1) &
    (tmp1.Find("Category:")) == -1)
    {
        mytest1 = mytest1 + tmp1;
        if(jj != 0)
            content_tmp = content_tmp + tmp1;
    }
    else
        d = sm_Htm11.Find("<br>", c);
    jj++;
}
content_tmp.Replace("<br>", "");
content_tmp.Replace("<b>", "");
content_tmp.Replace("</b>", "");
content_tmp.Replace("<font size=-1>", "");
Contents.Add(content_tmp);

there:
CString tmp_file_name = "";
rand(time(NULL));
CString file_name = itoa((rand()%1000), buff, 10);
file_name += ".txt";
CS.Lock();
CFile my_file2(path+file_name, CFile::modeCreate | CFile::modeWrite);
cs.Unlock();

CFile my_file2(C:\\result1.htm", CFile::modeCreate | CFile::modeWrite);
CString html_tmp = "<html><title>Results Page</title><\n<\/SCRIPT>
LANGUAGE="JavaScript"\nfunction change(){\n\n\n\n i<=document.f2.number.value; i++{\n tmp = document.f1.elements[i-1];\n if(tmp.checked){\n document.f2.selected_items.value +=\n document.f2.selected_items.value +=\n document.f2.number.value = j;\n alert(document.f2.number.value);\n
A-11
function help()
var a = window.open("
width=300 height=500"");
var b = a.document;
b.write("<html>");
b.write("<head>");
b.write("<title>Usage of Intelligent Agent Search</title>");
b.write("<body>");
b.write("<center>Usage of Intelligent Agent Search</center>");
b.write("<hr>");
step = Step 1: After receiving the search results, users select records to be inserted into database by clicking on the check box corresponding to the entry;
step = Step 2: Selected records are automatically submitted to the database by pressing the Submit button at the bottom of the page.

"Note: Users can not submit the form to update the database without first selecting at least one entry from the list. The Reset button at the bottom of the page cancels existing selections.
"

"<form method=\"POST\" name=\"f2\" action=\"--WEBBOT-SELF--\">

"<input type=\"hidden\" name=\"m_title\" value=\"">;

"<input type=\"hidden\" name=\"m_key\" value=\"">;

"<input type=\"hidden\" name=\"area\" value=\"">;

"<input type=\"hidden\" name=\"resource\" value=\"">;

"<input type=\"hidden\" name=\"number\" value=\"">;

"<input type=\"hidden\" name=\"filename\" value=\"">;

"<input type=\"hidden\" name=\"selected_items\" value=\">

onSubmit = function()

mytest1 = html_tmp;

CString tmp = "a";

int j;

for(j=0; j<index; j++)

CString tmpIndex = itoa(j+1, buff, 10);

my_file2.Write(tmpIndex, tmpIndex.GetLength());

my_file2.Write(tmp, tmp.GetLength());

my_file2.Write(URLs[j], URLs[j].GetLength());

my_file2.Write(Title[j], Titles[j].GetLength());

my_file2.Write(Contents[j], Contents[j].GetLength());

my_file2.Write(tmp, tmp.GetLength());
my_file2.Write(tmp, tmp.GetLength());
}
tmp = "EOF";
my_file2.Write(tmp, tmp.GetLength());
my_file2.Close();
tmp_file_list.Add(file_name);
return mytest1;

// the method used to insert record selected by user into database

void CSuvExtension::insert(CHttpServerContext* pCtx, char* title, char* keyword, char* area, char* resource, char* number, char* filename, char* selected_items)
{
    CString r_title(title);
    CString r_keyword(keyword);
    CString r_area(area);
    CString r_resource(resource);
    int r_number = atoi(number);
    CString r_filename(filename);
    CString r_selected_items(selected_items);
    CString return_page;
    return_page = do_insert(r_title, r_keyword, r_area, r_resource, r_number, r_filename, r_selected_items);
    StartContent(pCtx);
    WriteTitle(pCtx);
    *pCtx << _T("<br><br>");
    *pCtx << return_page;
    EndContent(pCtx);
}

// a help method used to insert records into database automatically

CString CSuvExtension::do_insert(CString& n_title, CString& n_keyword, CString& n_area, CString& n_resource, int& n_number, CString& n_filename, CString& n_selected_items)
{
    CString tmp_index="", tmp_return="<html><head><title>/n<insertion Results Feedback
page</title>

<body background="/backgrnd.jpg">

<h1>Insertion Results</h1>

Feedback</br><br></center>

". tmp_index = "n", statement, max_index;
    CString exist_items="";
    CString feedback:
    CString non_exist_items="";
    CStringArray index, url, title, content;
    char buff[30];
    bool* record_status = new bool[n_number];
    bool flag = false;
    int c, a, b, z=0;
    int exist_num = 0;
    c = n_selected_items.Find(" ", 0);
    while(c != -1){
        index.Add(n_selected_items.Mid(0, c));
        n_selected_items.Delete(0, c+1);
c = n_selected_items.Find(" ", 0);

z++;

} CStdioFile f;

.cs2.Lock();

CString my_file = path + n_filename;

.cs2.Unlock();

f.Open(_T(my_file), CFile::modeRead);

c = 0;

CString tag("EOF");

while(tmp_index.Compare(tag) != 0){

if(tmp_index.Compare(index.GetAt(c)) == 0){

    f.ReadString(tmp_index);

    url.Add(tmp_index);

    f.ReadString(tmp_index);

    title.Add(tmp_index);

    f.ReadString(tmp_index);

    content.Add(tmp_index);

    c++;

}

    f.ReadString(tmp_index);

}

f.Close();

f.Remove(_T(my_file));

CRobotDatabase db;

//establish the connect with database through odbc

if(!db.Open("svu", "Record", "sa", "){

    tmp_return += "<p>Sorry!, cannot open database "Record"!</"br>n";

}

}

db.m_nFields = 8;

    db.m_sFieldName[0] = "RecordID";

    db.m_bKeyField[0] = true;

    db.m_bNumeric[0] = true;

    db.m_sFieldName[1] = "Title";

    db.m_bKeyField[1] = false;

    db.m_bNumeric[1] = false;

    db.m_sFieldName[2] = "URL";

    db.m_bKeyField[2] = false;

    db.m_bNumeric[2] = false;

    db.m_sFieldName[3] = "Description";

    db.m_bKeyField[3] = false;

    db.m_bNumeric[3] = false;

    db.m_sFieldName[4] = "g_area";

    db.m_bKeyField[4] = false;

    db.m_bNumeric[4] = false;

    db.m_sFieldName[5] = "resource";

    db.m_bKeyField[5] = false;

    db.m_bNumeric[5] = false;
db.m_sFieldName[6] = "citation";
db.m_bKeyField[6] = false;
db.m_bNumeric[6] = false;

db.m_sFieldName[7] = "date";
db.m_bKeyField[7] = false;
db.m_bNumeric[7] = false;

for(int k=0; k<n_number;k++){
    statement = "URL=" + T(urlAt(k)) + "\n";
    if(db.SelectMatchingRecords(statement)){
        record_status[k] = true;
    }
    else{
        record_status[k] = false;
        flag = true;
    }
}

for(int j = 0; j < n_number; j++){
    if(record_status[j]==true){
        exist_items += "<p>";
        exist_items += url[j];
        exist_items += "<br>\n";
        exist_items += title[j];
        exist_items += "<br>\n";
        exist_items += content[j];
        exist_num++;
    }
    else{
        non_exist_items += "<p>";
        non_exist_items += url[j];
        non_exist_items += "<br>\n";
        non_exist_items += title[j];
        non_exist_items += "<br>\n";
        non_exist_items += content[j];
    }
}

statement = "recordid = (select max(recordid) from record)";
bool bHaveData = db.SelectMatchingRecords(statement);
while(bHaveData){
    bHaveData = db.NextMatchingRecord();
}
max_index = db.m_sFieldValue[0];
feedback += db.m_sFieldValue[1];
feedback += max_index;
a = atoi(T(max_index));
b = a;

//produce the sql command
for(int i=0; i<z;i++){
    if(record_status[i]==false){
        a++;
        statement = "insert into Record values(";
        statement += atoi(a, buff, 10);
        statement += ");
        
    
}
statement += "\";  
if(title[i].Find("\",0)){  
    title[i].Replace("\", "\");  
}  
statement += title[i];  
statement += "\";  
statement += ";";  
statement += ";";  
if(url[i].Find("\",0)){  
    url[i].Replace("\", "\");  
}  
statement += url[i];  
statement += "\";  
statement += ";";  
statement += ";";  
if(content[i].Find("\",0)){  
    content[i].Replace("\", "\");  
}  
statement += content[i];  
statement += "\";  
statement += ";";  
statement += ";";  
statement += n_area;  
statement += "\";  
statement += ";";  
statement += ";";  
statement += n_resource;  
statement += "\";  
statement += ";";  
statement += ";";  
statement += ";";  
statement += ";";  
statement += "date()";  
statement += ");";  
if(!db.Execute(statement)){  
    tmp_return += "db error msg: ";  
    tmp_return += db.m_sErrMsg;  
    tmp_return +="<br>n";  
}  
statement = "insert into have_keyword values(";  
statement += itoa(a, buff, 10);  
statement += ",\";  
statement += n_keyword;  
statement += ");";  
if(!db.Execute(statement)){  
    tmp_return += "db1 error msg: ";  
    tmp_return += db.m_sErrMsg;  
    tmp_return +="<br>n";  
}  
    }  
    }  
    }  
    }  
db.Close();  
tmp_return += "<p>";  
tmp_return += itoa(z, buff, 10);  
tmp_return += " URLs selected.";  
tmp_return += "<p>";
void CSuvExtension::myUpdate( CHttpServerContext* pCtx, char* username, char* password)
{
    CRobotInternet internet;
    CString sURL, sHeader, sErrMsg, index, return_page;
    int nResult;
    int del_num = 0;
    int sum = 0;
    char buff[30];
    CString statement;
    CString my_username = "lane";
    CString my_passwd = "q23456";
    CString tmp_username(username);
    CString tmp_passwd(password);
    CString msg_f = "<html><head><title>nReulst page\n</title><\n<head><body background="/backgrd.jpg">\n<h2>Sorry, there is something wrong with your username and password, please try again!\n</h2><\n<head><\n";  // background of the error page.
    CString msg_s1 = "<html><head><title>nReulst page\n</title><\n<head><body background="/backgrd.jpg">\n<h2>Updation of database is finished successfully!\n</h2><\n<head><\n";  // background of the success page.
    CString msg_s2 = " records in the database. ";  // message for records.
    CString msg_s3 = " records have been deleted\n</body></\n<head><\n";  // message for records deleted.
    CString msg_d = "<html><head><title>nReulst page\n</title><\n<head><body background="/backgrd.jpg">\n<h2>Sorry, We can not finish the Update operation at this moment, please try again later! The error message is:\n</h2><\n<head><\n";  // message for error.
    if((my_username != tmp_username)||(my_passwd != tmp_passwd)) {
        return_page = msg_f;
    } else {
        CRobotDatabase db;
        if(!db.Open("suv", "Record", "sa", "")) {
            return_page = msg_d + db.m_sErrMsg;
            return_page += "</body></\n<head><\n";
            goto there;
        } else {
            db.m_nFields = 8;
            db.m_sFieldName[0] = "RecordID";
            db.m_bKeyField[0] = true;
            db.m_bNumeric[0] = true;
        }
    }
    // the method used to update all records in database
    // delete those records whose url doesn't exist anymore.
db.m_sFieldName[1] = "Title";
db.m_bKeyName[1] = false;

db.m_sFieldName[2] = "URL";
db.m_bKeyName[2] = false;

db.m_sFieldName[3] = "Description";
db.m_bKeyName[3] = false;

db.m_sFieldName[4] = "g_area";
db.m_bKeyName[4] = false;

db.m_sFieldName[5] = "resource";
db.m_bKeyName[5] = false;

db.m_sFieldName[6] = "citation";
db.m_bKeyName[6] = false;

db.m_bNumeric[6] = false;

while(bHaveData){
    sum++;
    index = db.m_sFieldVal[0];
    sURL = db.m_sFieldVal[2];
    if((internet.httpHeader(sURL, sHeader, nResult, sErrMsg))
        statement = "delete * from Record where RecordID=";
        statement += index;
        db.Execute(statement);
        del_num++;
    }
    bHaveData = db.NextMatchingRecord();
}

return_page = msg_s1 + _itoa(sum-del_num, buff, 10);
return_page += msg_s2;
return_page += _itoa(del_num, buff, 10);
return_page += msg_s3;
}

there:
    StartContent(pCtx);
    WriteTitle(pCtx);
* pCtx << return_page;
EndContent(pCtx);

```
int pos = 0;
int number = 0;
while(page.Find("<p>".pos)!=-1)
{
    number++;
pos += 3;
}
return number;

// Do not edit the following lines, which are needed by ClassWizard.
#ifndef
BEGIN_MESSAGE_MAP(CSuvExtension, CHttpServer)
    // {AFX_MSG_MAP(CSuvExtension)
    // }AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

/******************************************************************************
 // If your extension will not use MFC, you'll need this code to make
 // sure the extension objects can find the resource handle for the
 // module. If you convert your extension to not be dependent on MFC,
 // remove the comments around the following AfxGetResourceHandle()
 // and DllMain() functions, as well as the g_hInstance global.

/****
static HINSTANCE g_hInstance;

HINSTANCE AFXISAPI AfxGetResourceHandle()
{
    return g_hInstance;
}

BOOL WINAPI DllMain(HINSTANCE hInst, ULONG ulReason,
    LPVOID lpReserved)
{
    if (ulReason == DLL_PROCESS_ATTACH)
    {
        g_hInstance = hInst;
    }
    return TRUE;
}
****/
```
APPENDIX B

JAVA SCRIPT FOR FRONT-END

Description:

These functions, embedded in the HTML page, are written in JavaScript language. They provide user several facilities to interact with web server.

This source code contains documentation and comments throughout the code to describe code functionality.
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>checktitle()</td>
<td>B-3</td>
</tr>
<tr>
<td>update()</td>
<td>B-3</td>
</tr>
<tr>
<td>re_set()</td>
<td>B-4</td>
</tr>
<tr>
<td>change()</td>
<td>B-4</td>
</tr>
<tr>
<td>help()</td>
<td>B-4</td>
</tr>
</tbody>
</table>
function help()
{
    var a = window.open("", ", " width=300 height=450");
    var b = a.document;
    b.write("<html>");
    b.write("<head>");
    b.write("<title>Usage of Agent Search</title>");
    b.write("<head>");
    b.write("<body background="/background.jpg">");
    b.write("<center><h2>Usage of Agent Search</h2><center>");
    b.write("<hr><hr><hr>");
    b.write("<li>Step 1: Users are required to input search title in the search title box.");
    b.write("<li>Step 2: Users may select related keyword, area and resource from the corresponding
    checkboxes.");
    b.write("<li>Step 3: Users may select the maximum number of results for each search from a set
    of options provided.");
    b.write("<br><hr><br><br>");
    b.write("Note:<br>");
    b.write("Since Agent Search is carried out in real time, it may take several minutes to complete the
    search task. Thank you for waiting result back. The reset button returns the field entries to their default
    values.");
    b.write("<body>");
    b.write("</html>");
}

//the method used to prevent user to submit blank keyword to search agent

function checktitle()
{
    while(document.fl.elements[0].value == ""||document.fl.elements[0].value=="null")
    {
        document.fl.elements[0].value = prompt("Search Title can not be blank! Please input search
        title:" , "");
    }

    //the method used to prompt out a windows to allow user to update database

function update()
{
    var a = window.open("", ", " width=400 height=220");
    var b = a.document;
    b.write("<html>");
    b.write("<head>");
    b.write("<title>Update database</title>");
    b.write("<head>");
    b.write("<body background="/background.jpg">");
    b.write("<center><h2>Update Database</h2><center>");
    b.write("<hr><br>");
    b.write("<form method="post" action="/svu_scripts/suv.dll?my_update" >");
    b.write("<p>Username: <input type=text name="username" value="" size="20"></p>");
    b.write("<p>Password: <input type=password name="password" value="" size="20"></p>");
function re_set()
{
    document.f1.reset();
}

function change()
{
    var j = 0; for(var i=1; i<=document.f2.number.value;i++)
    {
        tmp = document.f1.elements[i-1];
        if(tmp.checked)
            document.f2.selected_items.value +=i;
        document.f2.selected_items.value +=" ";
        j++;
    }
    document.f2.number.value = j;
    alert(document.f2.number.value);
    alert(document.f2.selected_items.value);
}

// the method used to prompt out a help windows to display the contents of the usage of Agent Search

function help()
{
    var a = window.open("", "", "width=300 height=400");
    var b = a.document;
    b.write("<html>");
    b.write("<head>");
    b.write("<title>Usage of Spider Search<title>");
    b.write("<head>");
    b.write("<body background=background.jpg>");
    b.write("<center><h2>Usage of Spider Search<h2></center>");
    b.write("<br><hr><br>");
    b.write("<li>Step 1: After receiving the search results, users select records to be inserted into database by clicking on the check box corresponding to the entry.");
    b.write("<li>Step 2: Selected records are automatically submitted to the database by pressing the Submit button at the bottom of the page.");
b.write("<br><br><br><br>");
b.write("Note:");
b.write("Users can not submit the form to update the database without first selecting at least one entry from the list. The Reset button at the bottom of the page cancels existing selections.");
b.write("</body>");
b.write("</html>");
}
APPENDIX C
IDC SCRIPT FOR FRONT-END

Description:

IDC, Internet Database Connector, is a technology by Microsoft used to establish connection between the web server and the database through ODBC, Open Database Connectivity.

In the Front end of this project, there are several IDC file sets (.idc/.htx). The .idc file contains sql commands. The commands are invoked by users with parameters to manipulate database through ODBC. The .htx file works as a html template, it is used to display the result of operation of database to user.

The comments included for each IDC file set will introduce its functionality.
//bring add-drop page to user and retrieval related information from database and display
//them in corresponding combo box
s_add_drop.idc
s_add_drop.htx
//bring the location template to user and let user to input record manually
s_add_new_area.idc
s_add_new_area.htx
//insert keyword into database manually and display result of insertion
s_add_new_keyword.idc
s_add_new_keyword.htx
//insert record into database manually and display result of insertion
s_Add_new_record.idc
s_Add_new_record.htx
//bring the vehicle type template to user and let user to input record manually
s_add_new_resource.idc
s_add_new_resource.htx
//bring the record template to user and let user to insert record into database manually
s_add_record.idc
s_add_record.htx
// let user to establish the connection between keyword and record
s_add_relation.idc
s_add_relation.htx
/////bring the delete page to user, retrieval related information from database and display
//them in corresponding combo box
s_delete.idc
s_delete.htx
//retrieval locations out from database and let user to select one to delete
s_delete_area.idc
s_delete_area.htx
//retrieval keywords out from database and let user to select one to delete
s_delete_keyword.idc
s_delete_keyword.htx
//retrieval records out from database and let user to select one to delete
s_delete_record.idc
s_delete_record.htx
//retrieval vehicle type out from database and let user to select one to delete
s_delete_resource.idc
s_delete_resource.htx
// let user to drop the connection between keyword and record
s_drop_relation.idc
s_drop_relation.htx
//bring the modify page to user, retrieval related information from database and display
//them in corresponding combo box
s_modify.idc
s_modify.htx
//make change of keyword information and display the result of change to user
s_modify_keyword.idc
s_modify_keyword.htx
// make change of record information and display the result of change to user
s_modify_record.idc
s_modify_record.htx
// retrieval corresponding keyword out from database to let user make change
s_need_modify_keyword.idc
s_need_modify_keyword.htx
// retrieval corresponding record out from database to let user make change
s_need_modify_record.idc
s_need_modify_record.htx
// bring the query page to user, retrieval related information from database and display
// them in corresponding combo box
s_query.idc
s_query.htx
// do query by area in database and display result to user
s_querybyarea.idc
s_querybyarea.htx
// do query by input date in database and display result to user
s_querybydate.idc
s_querybydate.htx
// do query by keyword in database and display result to user
s_querybykid.idc
s_querybykid.htx
// do query by vehicle type in database and display result to user
s_querybyres.idc
s_querybyres.htx
// do query by record title in database and display result to user
s_querybyrid.idc
s_querybyrid.htx
// do query by input word in record title in database and display result to user
s_querybytitle.idc
s_querybytitle.htx
// bring search page to user and retrieval related information from database and display
// them in corresponding combo box
s_search_page.idc
s_search_page.htx