Université d'Ottawa · University of Ottawa

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

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---------

---------List of Figures---------

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 "David3$5* Smith", or a key attribute in a table cannot be "NULL".

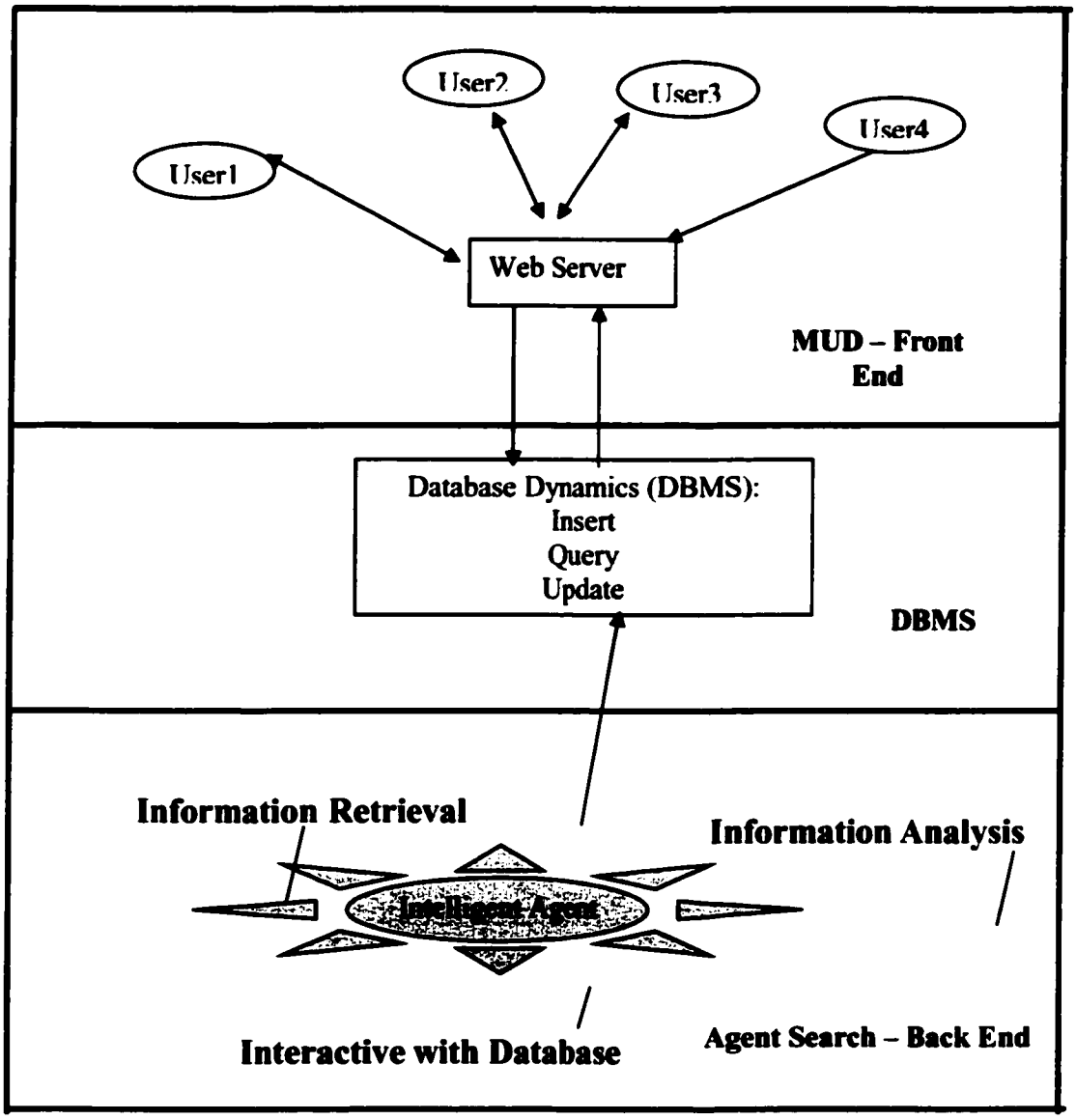


Figure 1: Web Database Design for a General Application

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 (WWWW) (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 charteristics.

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".

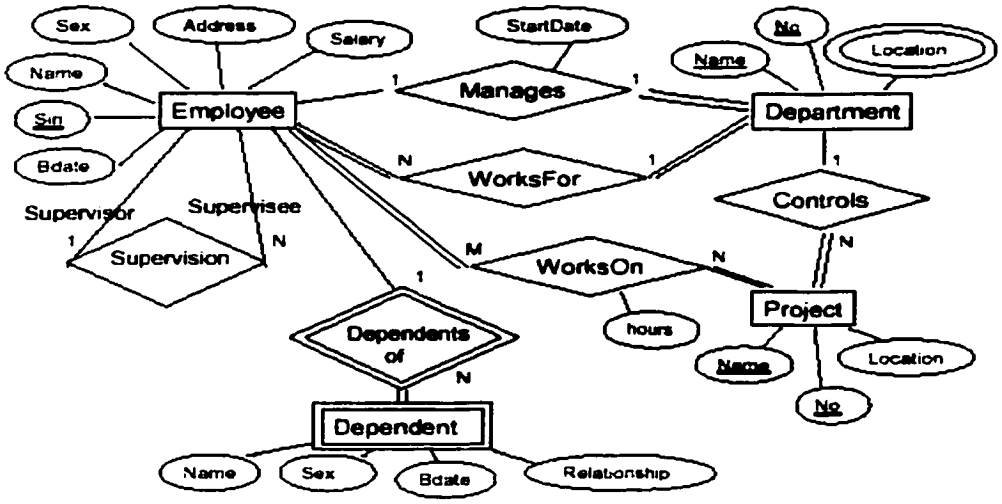
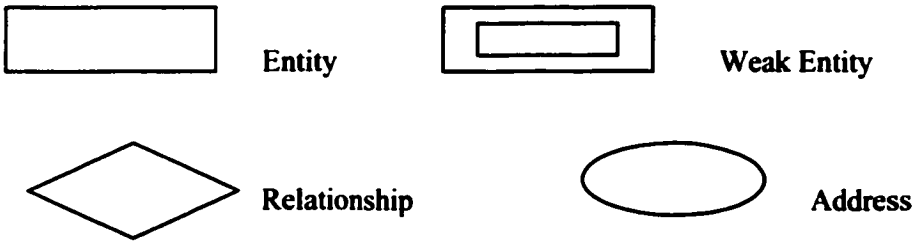
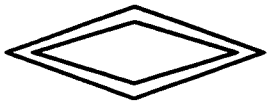


Figure 2: The relationship diagram of a company (Nel 2001)

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





Identifying Relationship (for weak entity)

Key Attributes

Multi-Valued Attributes

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).

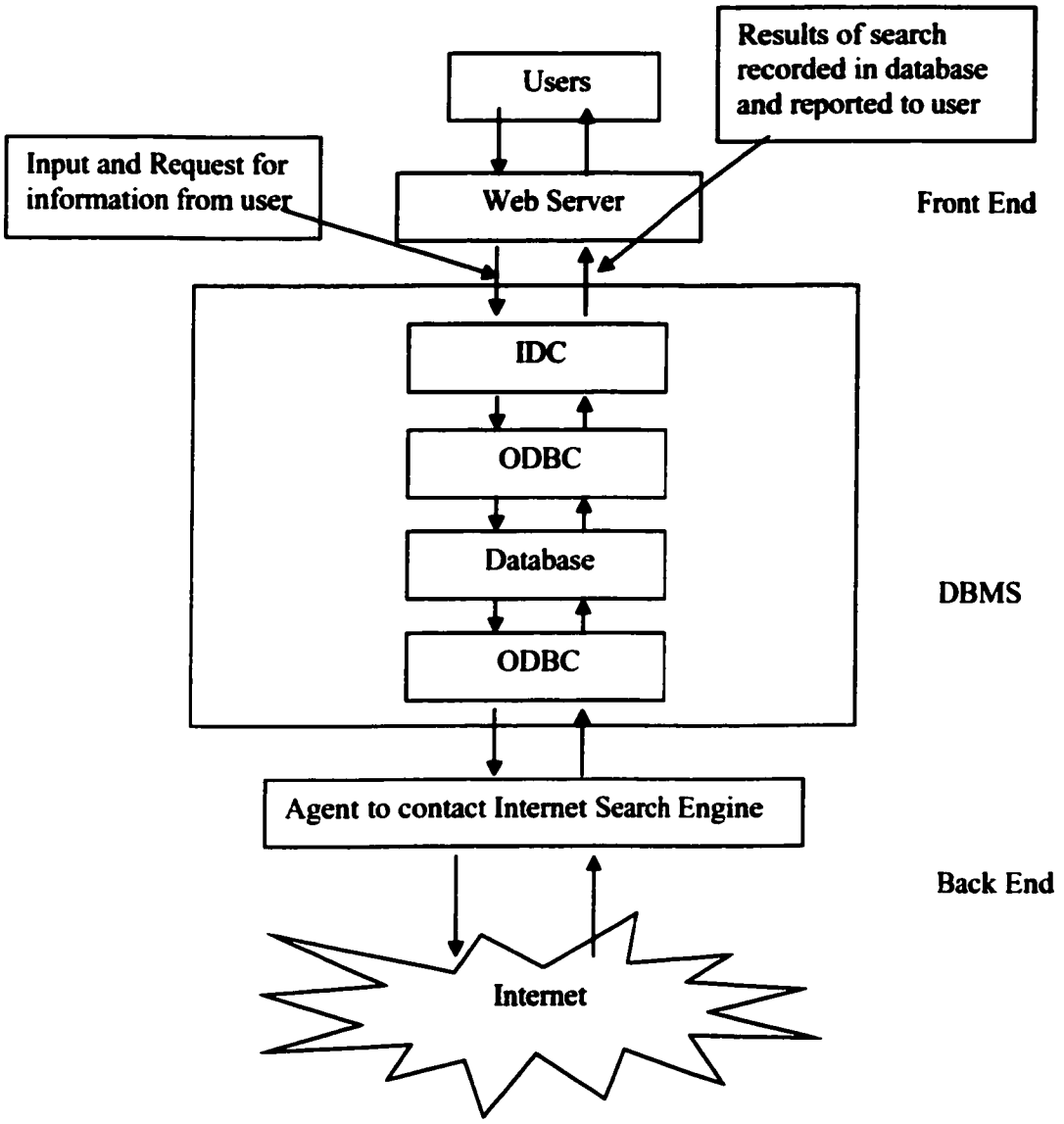


Figure 3: System integration application diagram showing components for Front End, DBMS and Agent-Internet Back End.

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):

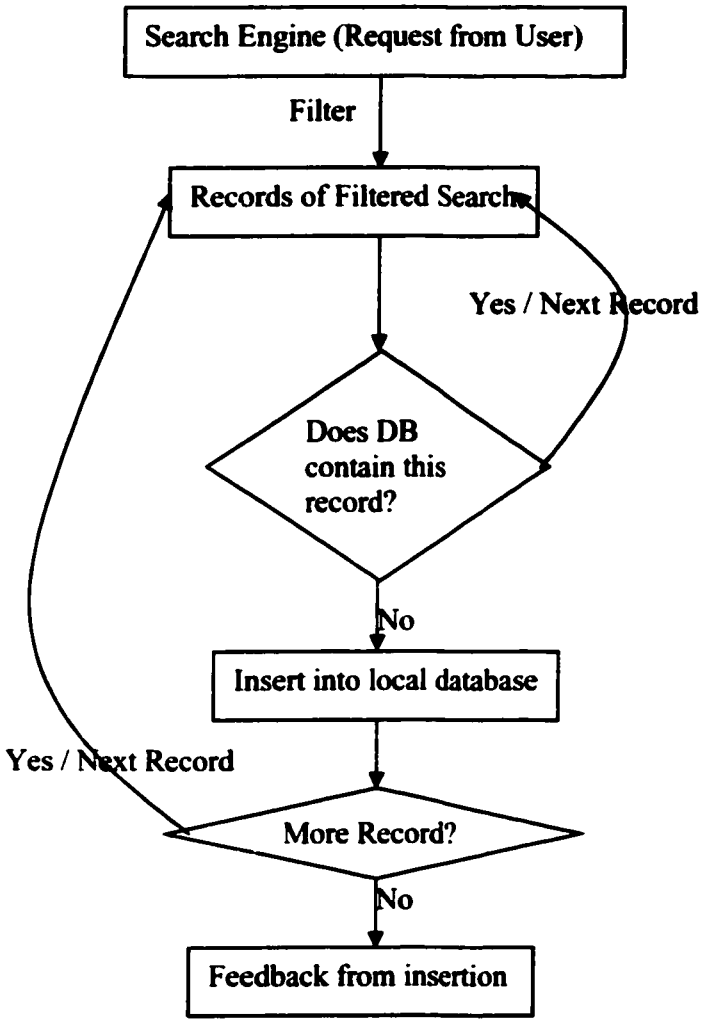


Figure 4: Web-Based Back End: The Intelligent Agent

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.

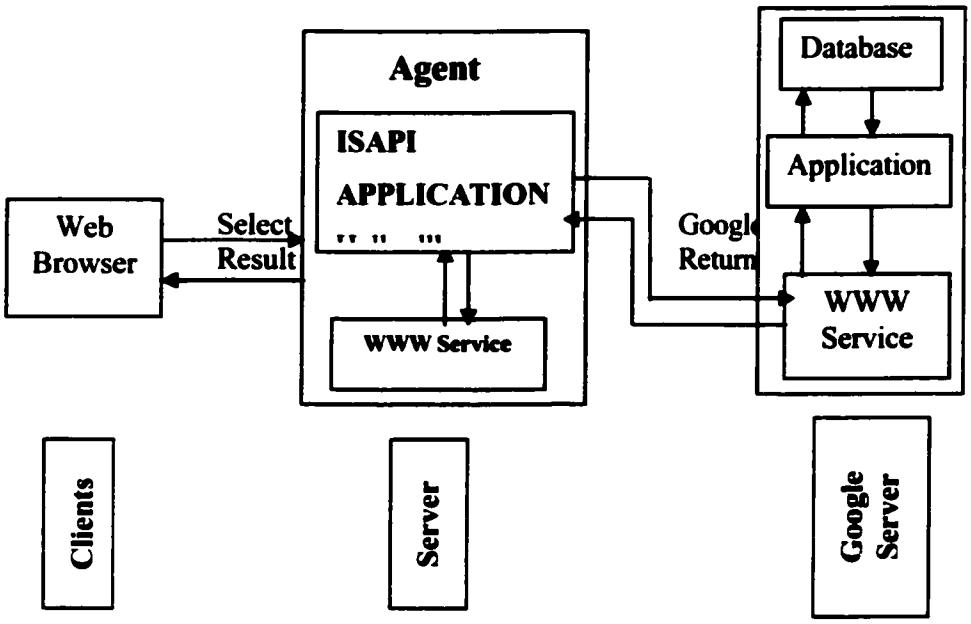


Figure 5: Internet Search Process

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).

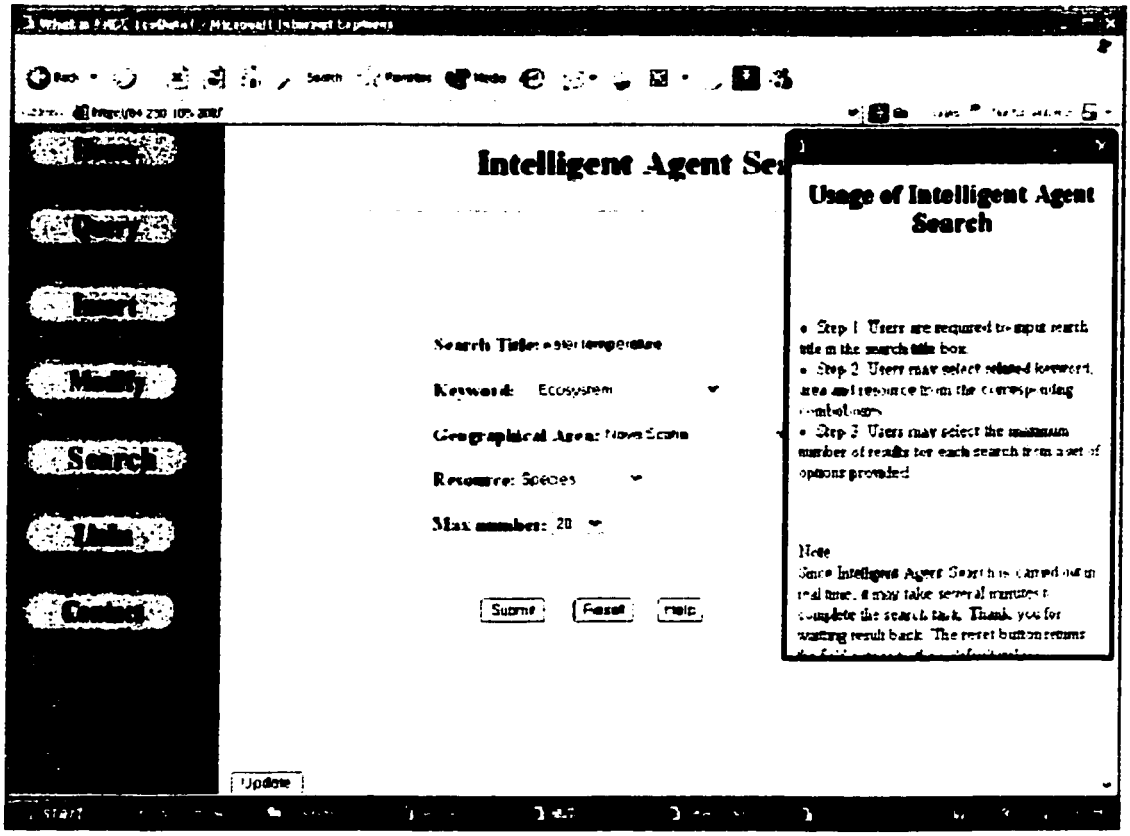


Figure 6: GUI of the Intelligent Agent

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):

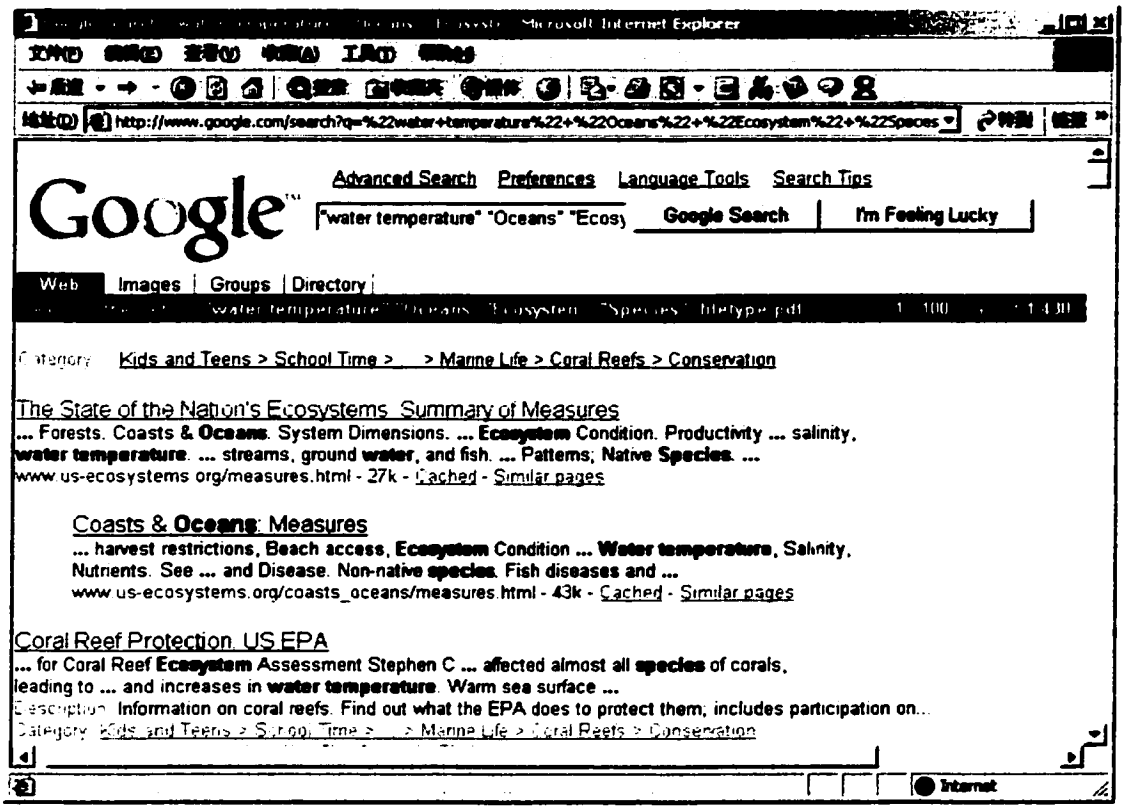


Figure 7: Google search result

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).

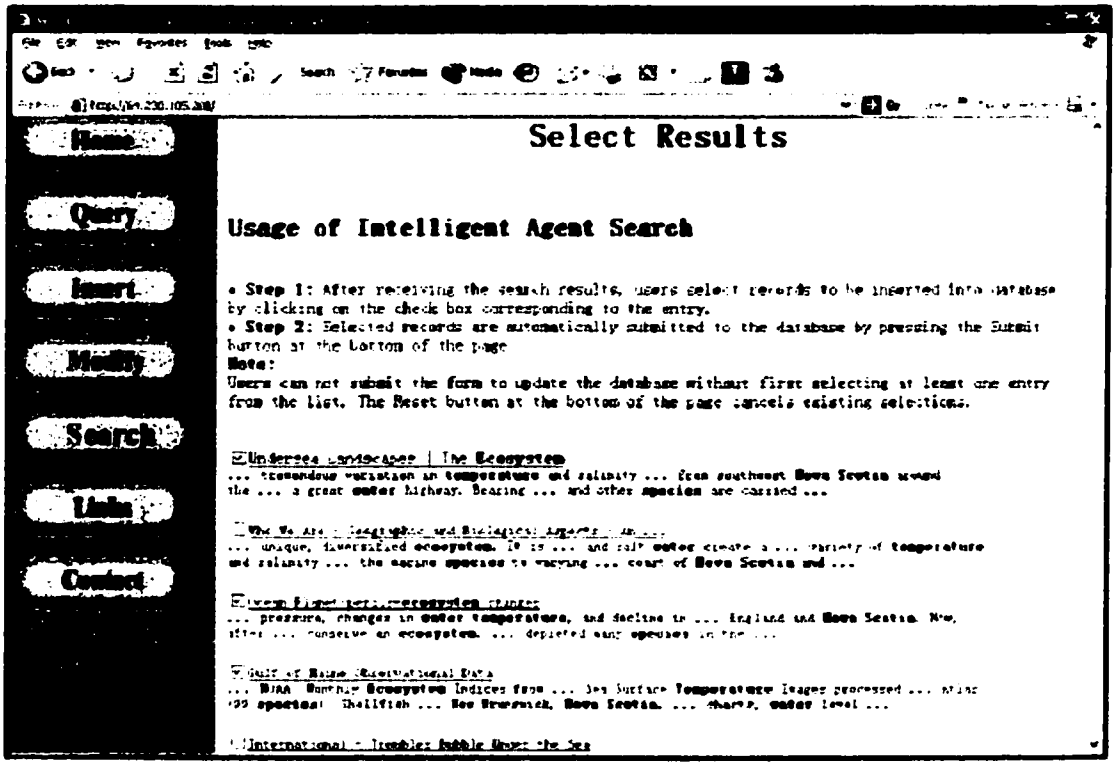


Figure 8: Final search result

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).

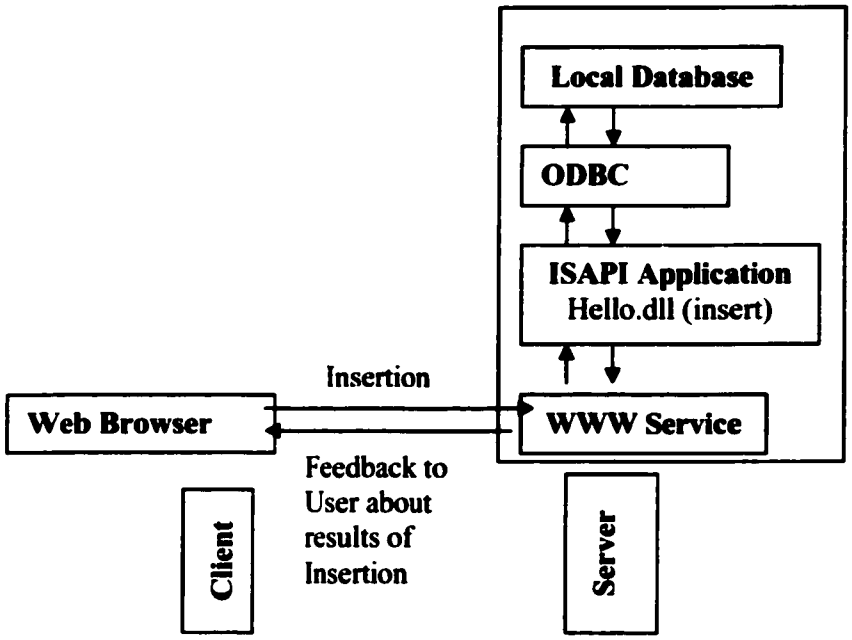


Figure 9: Database insertion process

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.

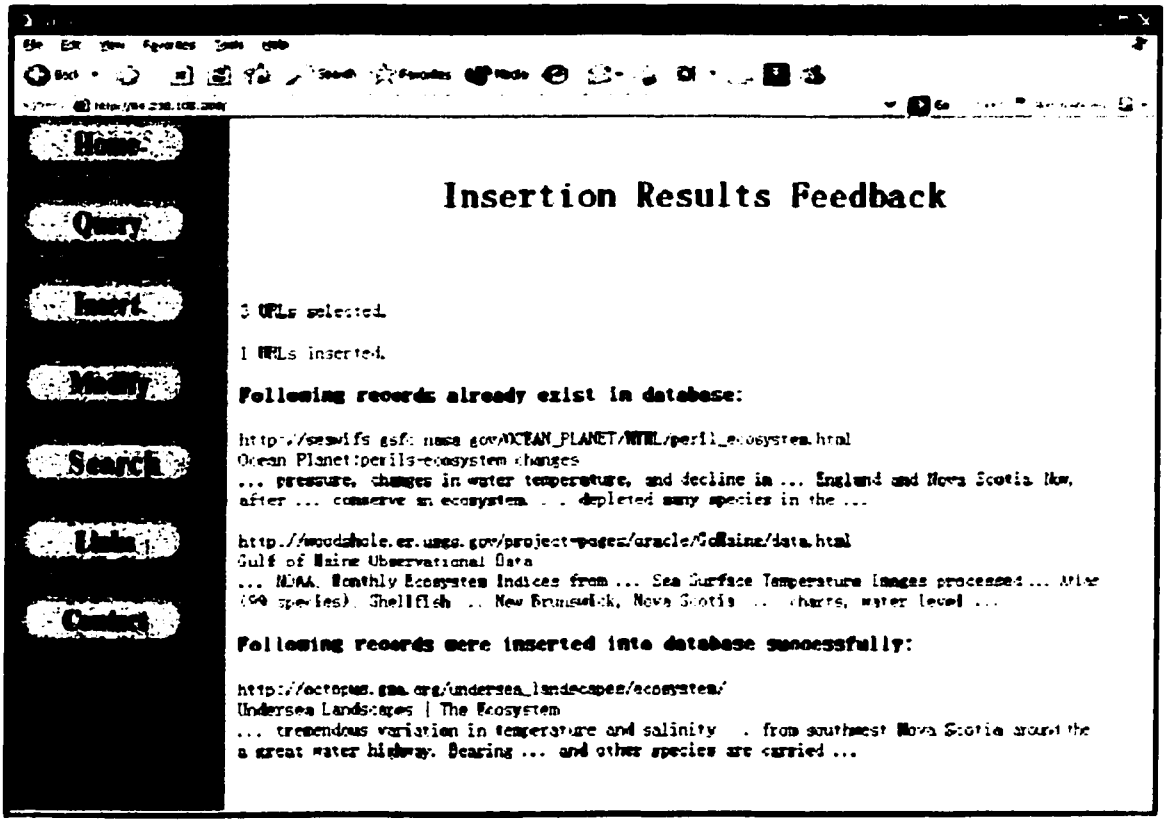


Figure 10: The feedback web page of an insertion into the database

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:

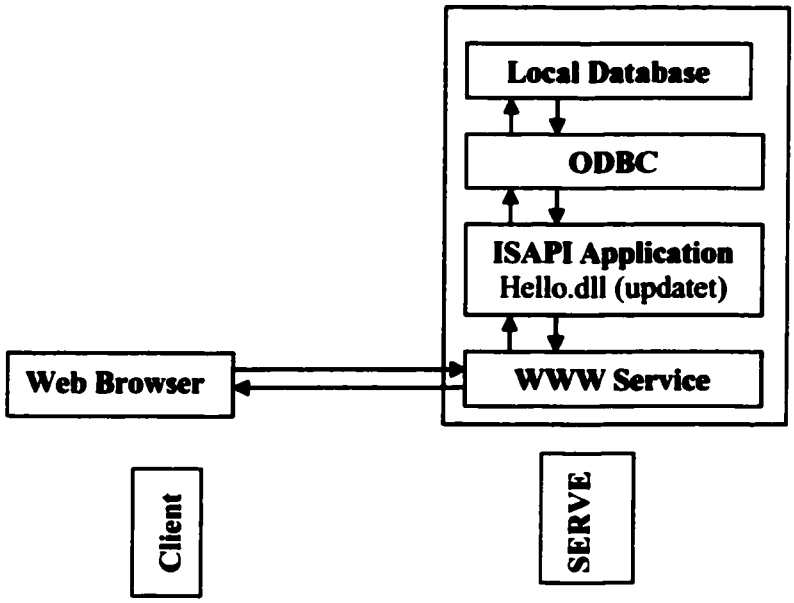


Figure 11: Principle of updating the database

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.

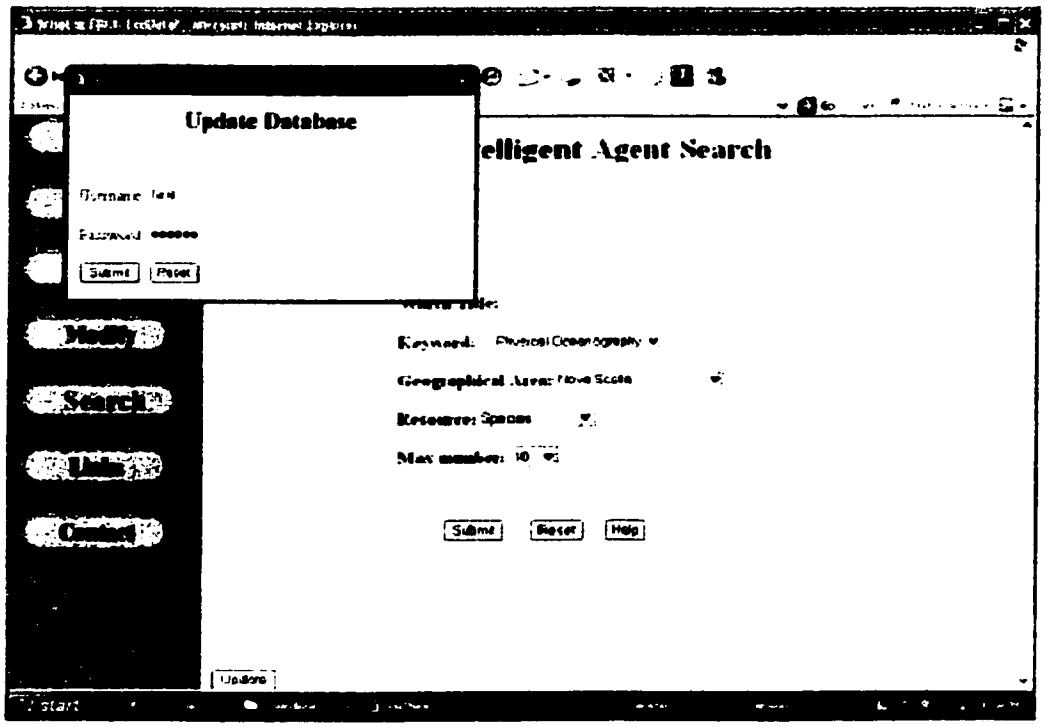


Figure 12: The pop-up window of update

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, preadtor-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.

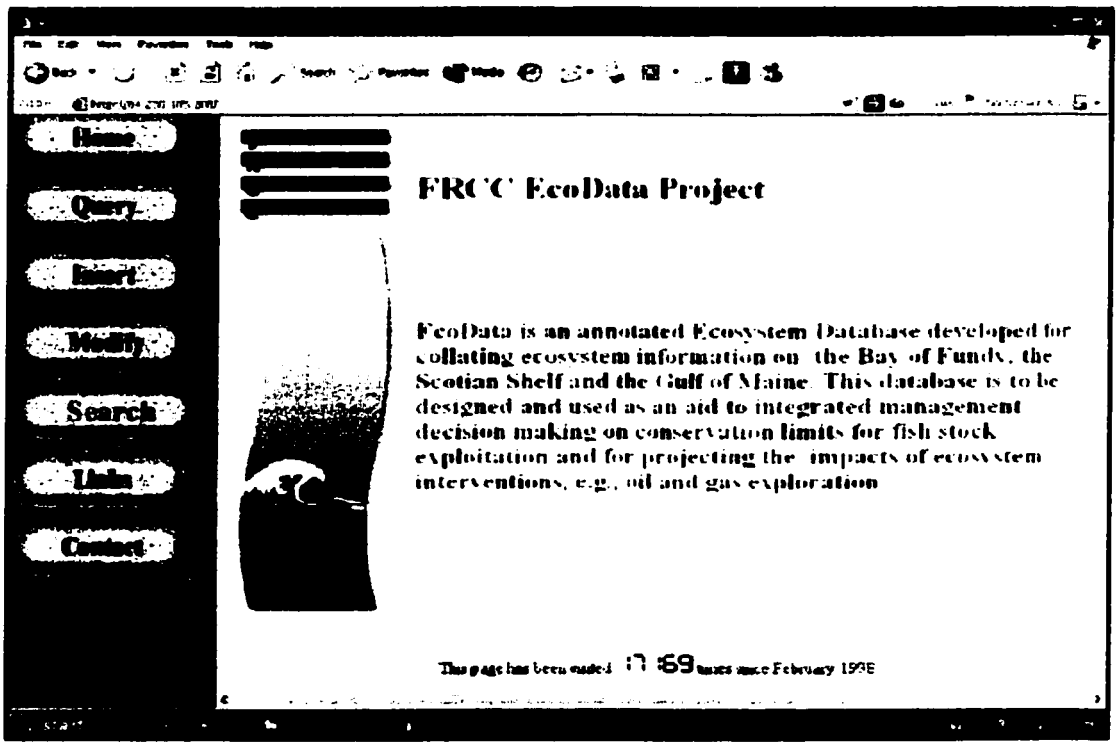


Figure 13: FRCC EcoData Project Home Page

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:

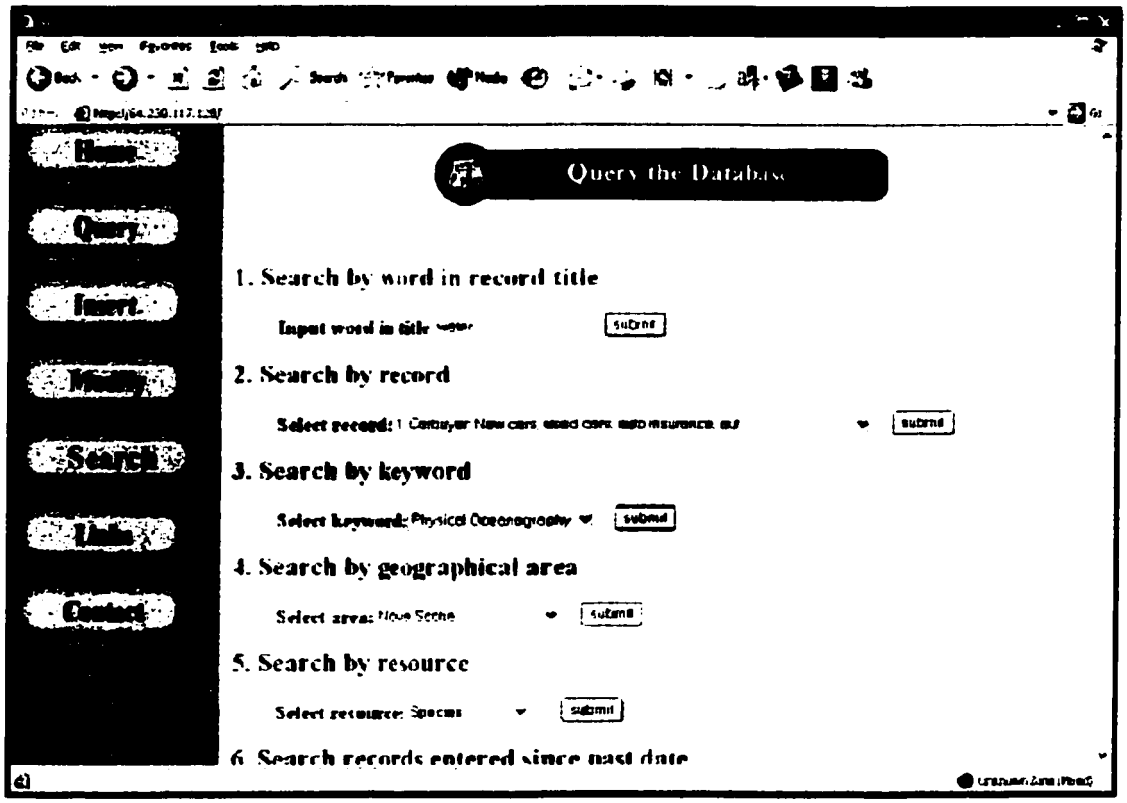


Figure 14: The Query Web Page of FRCC EcoData Project

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.

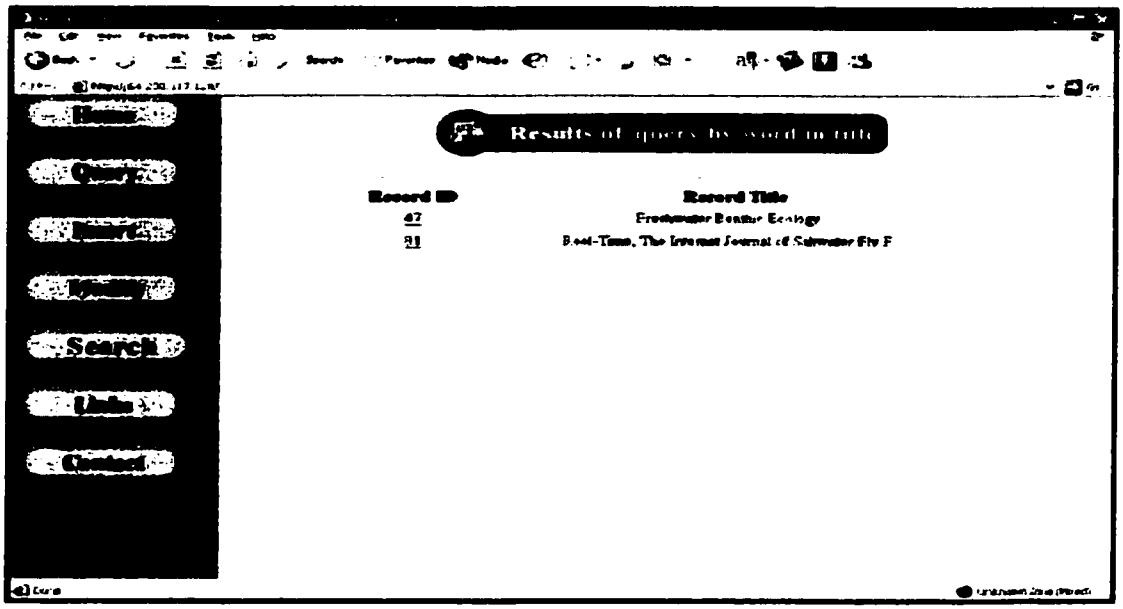


Figure 15: The Result Page of the Query

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 brower 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).

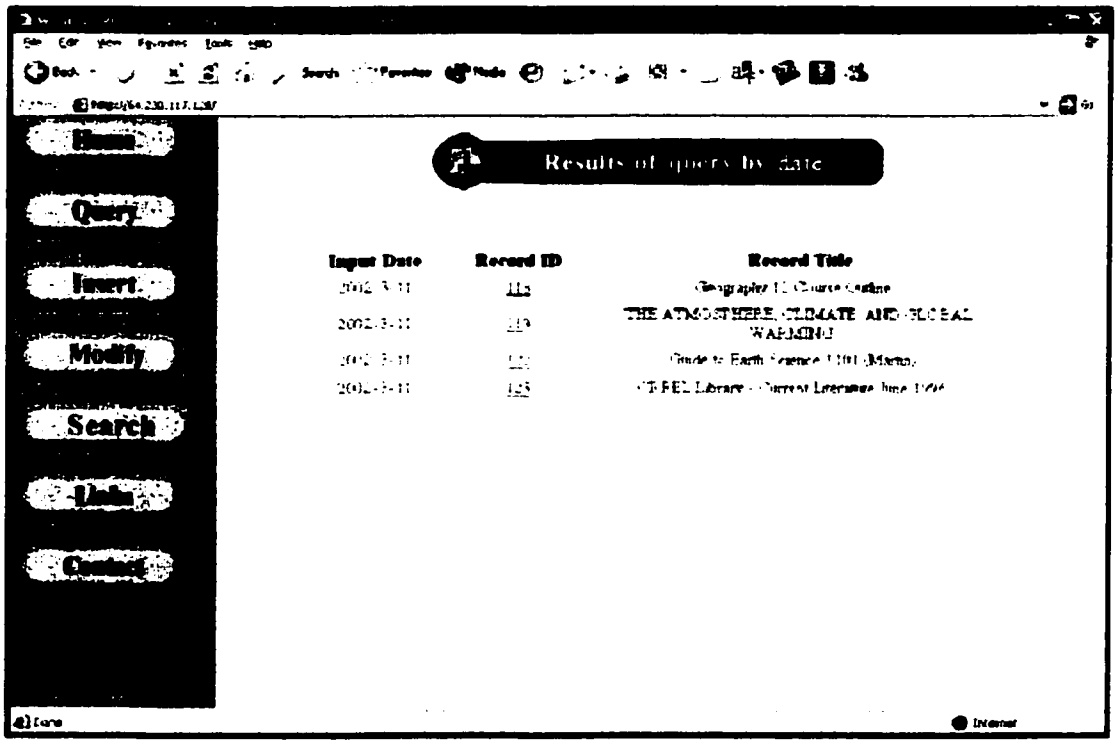


Figure 16: Results of Query by Date

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).

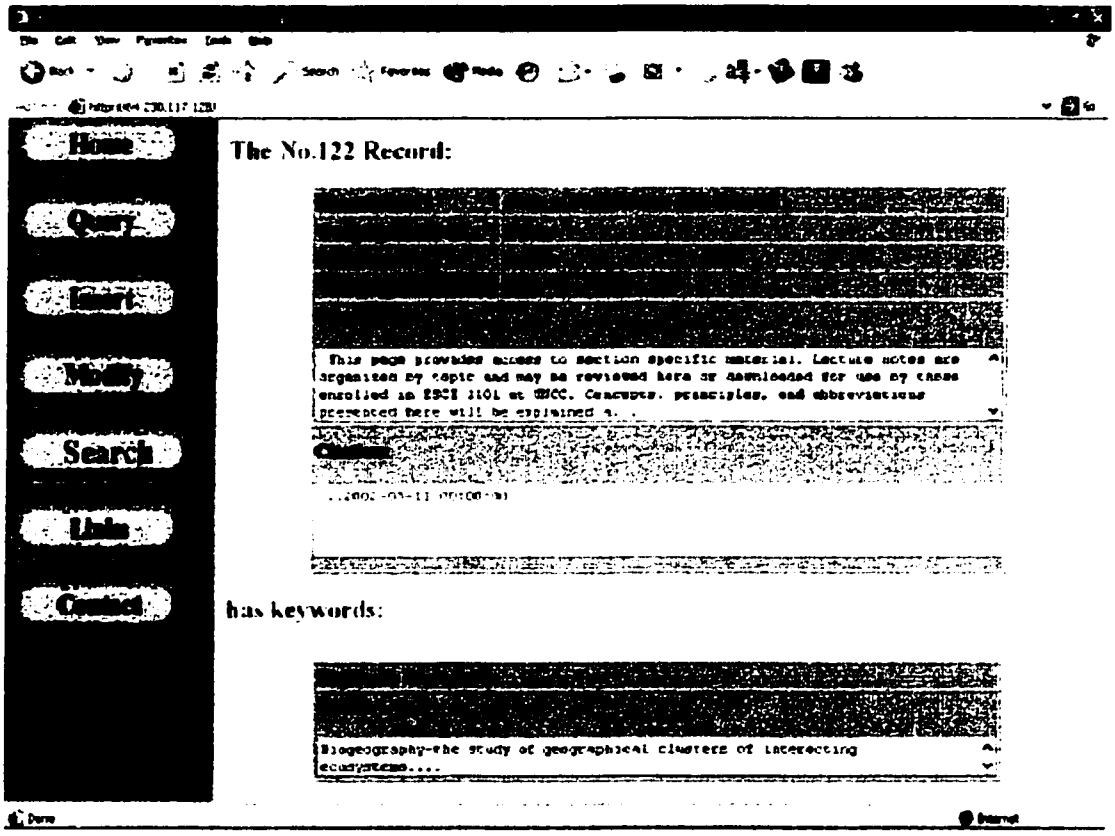


Figure 17: A Specific Record

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 row 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 soures 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 vehcile, 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.

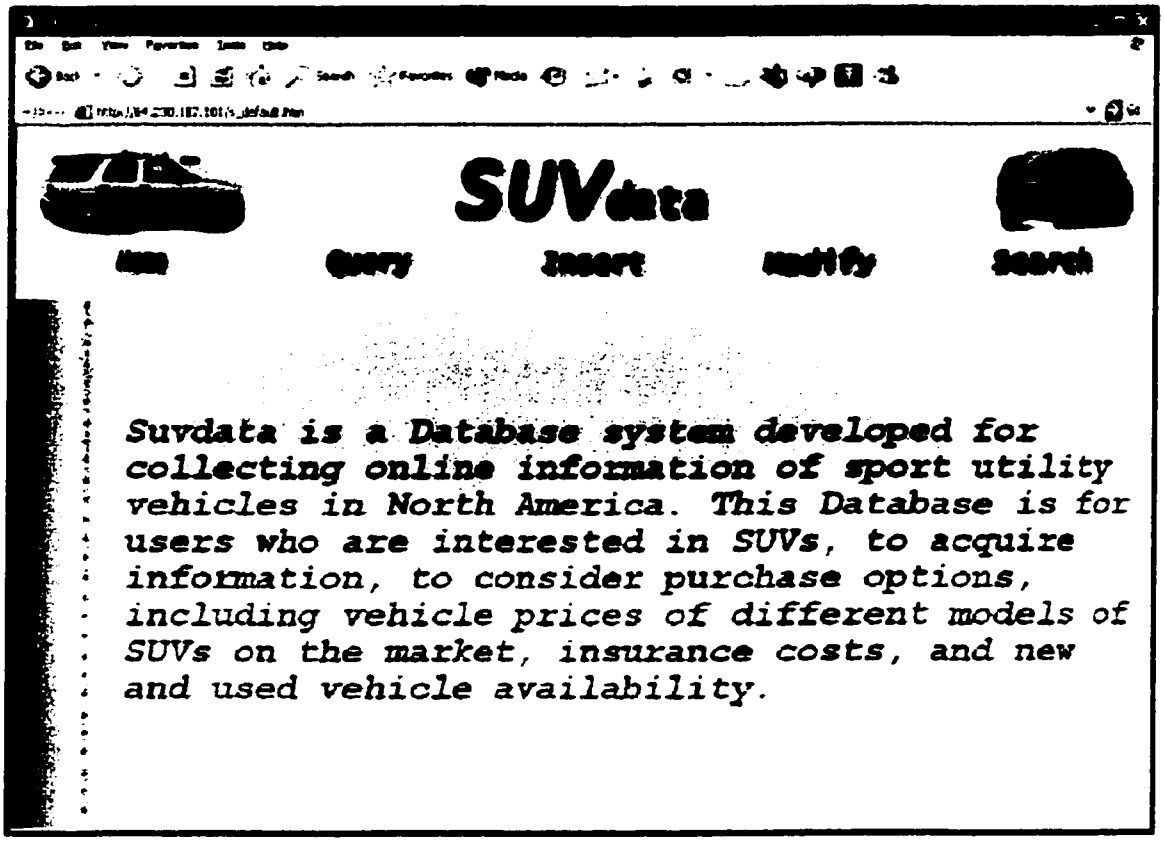


Figure 18: The Home Page of SUV Application

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.

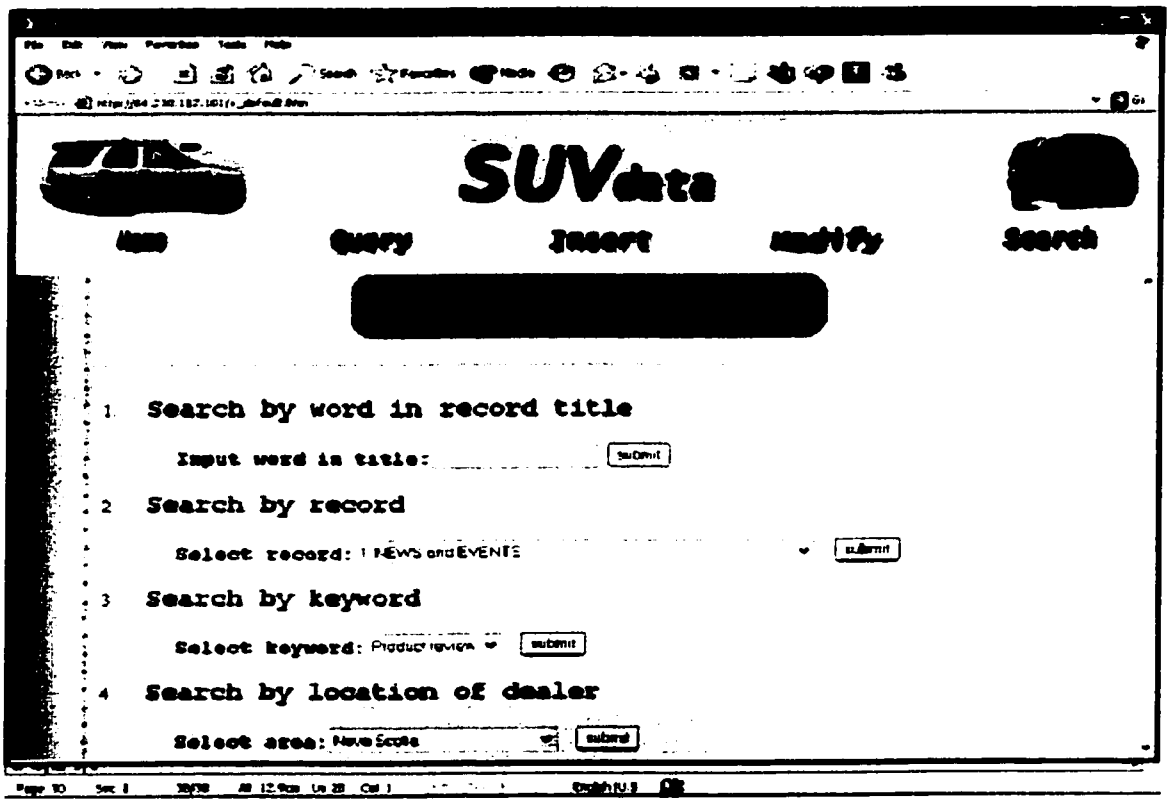


Figure 19: The Query Web Page of the Intelligent Agent

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.

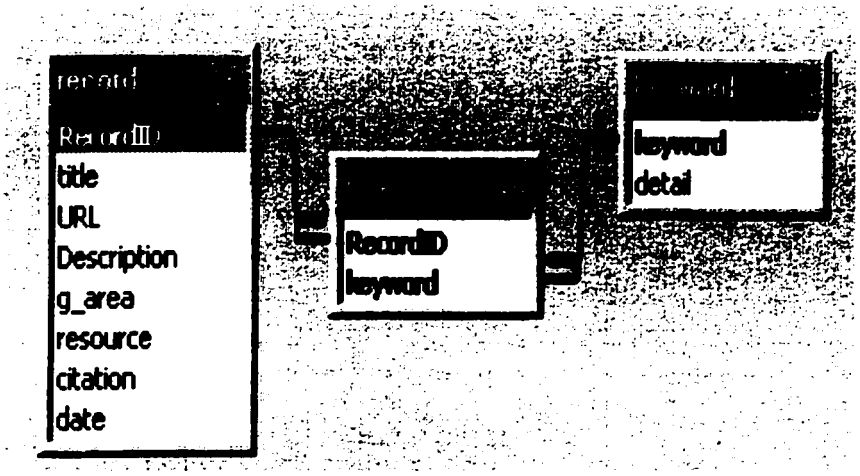


Figure 20: The Database of SUV

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:

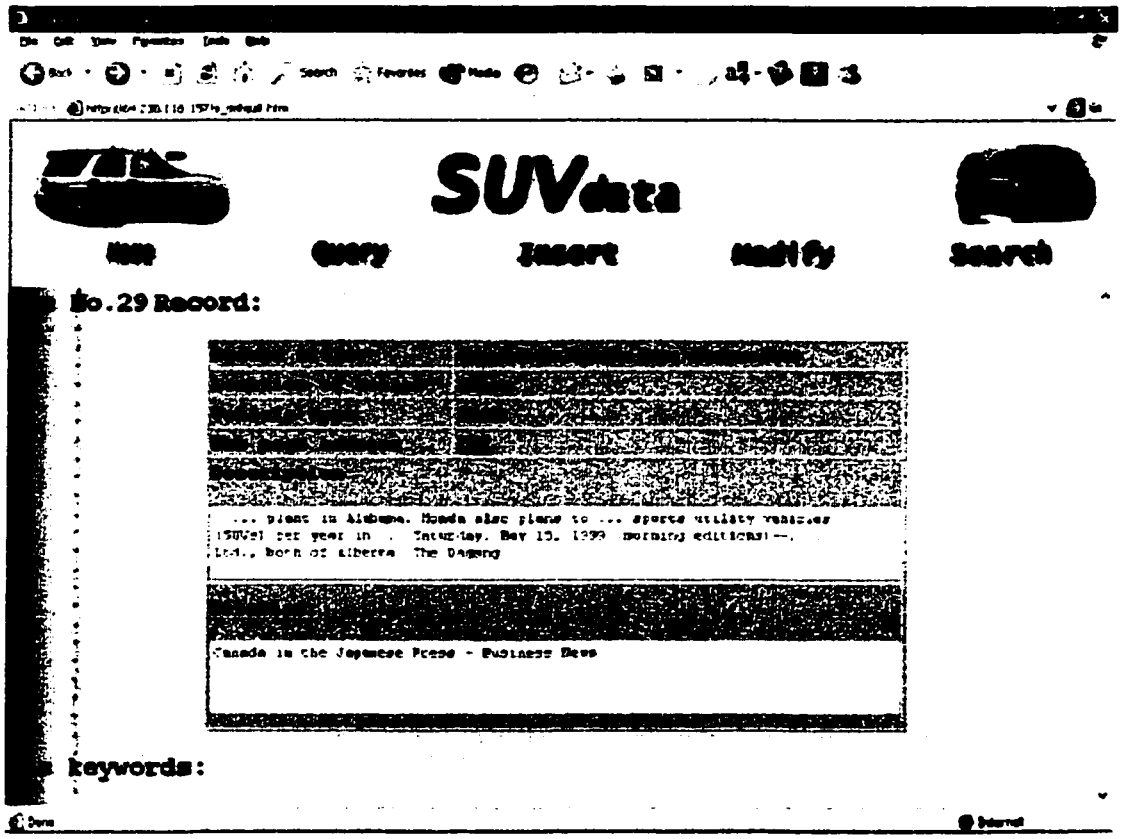


Figure 21: A Query record of SUV

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 for. 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.

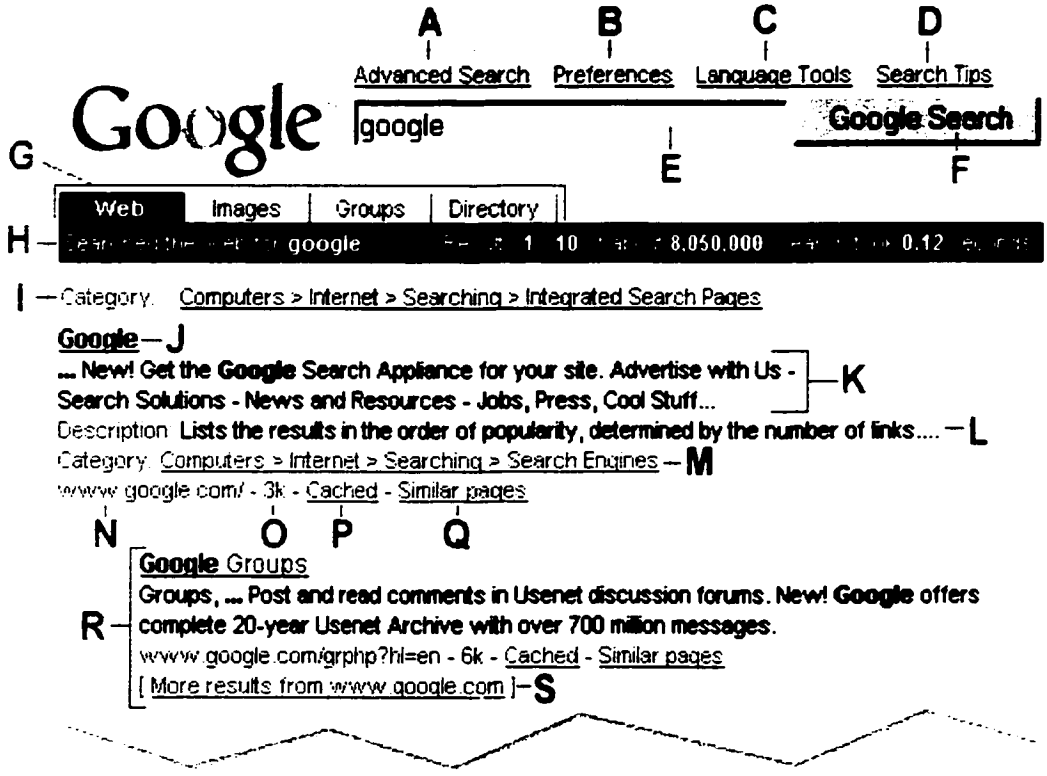


Figure 22: Interpretation of the search result (Google 2002)

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):

```
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);
.....
```

This code can insert the URL and the title of the result into different sting 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

1. About.com, Inc. 2002. The History of Sport Utility Vehicles.
2. http://4wheeldrive.about.com/library/weekly/aa012601b.htm
3. ActiveEducation. 1999. Microsoft® FrontPage® 2000 Step by Step. Microsoft Press.
4. Alta Vista search engine, 2002. http://www.altavista.com/.
5. Beck Z. 1998. MICROSOFT VISUAL C++ 6.0 PROGRAMMER'S GUIDE. Microsoft Press.
6. Braginski, L. and Powell, M. 1998. Running Microsoft® Internet Information Server. Microsoft Press.
7. Brett P. W. and Kerry A. Lehto. 1999. Official Microsoft® FrontPage® 2000 Book. Microsoft Press.
8. Brin, Sergey and Page, Lawrence. 1999. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Science Department, Stanford University. http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm
9. Buyens, J. 1999. Running Microsoft® FrontPage® 2000. Microsoft Press.
10. Catapult, Inc. 1999. Microsoft® Access 2000 Step by Step. Microsoft Press.
11. David J. K., George S., Scot W. 1998. PROGRAMMING MICROSOFT VISUAL C++, Fifth edition. Microsoft Press.
12. David P. 1999. Programming Bots, Spiders, and Intelligent Agents in Microsoft Visual C++ (Microsoft Programming Series). Microsoft Press.
13. Diversity University, http://www.du.org/
14. Diversity University, 1994. Journal of VR Education Vol. 1 No. 1 1994 http://tecfa.unige.ch/edu-comp/DUJVRE/vol1/DUJVRE.1.1.text

15. Elmasri, R. and Navathe, S. B., 2000. Fundamentals of Database Systems, Third edition. Addison-Wesley Publishing Co.

16. Enguix, C. F., 1998. Database querying on the World Wide Web: *UniGuide*, an object-relational search engine for Australian universities, http://www7.scu.edu.au/programme/docpapers/1891/com1891.htm

17. Fah-Chun C. 1996. Internet Agents: Spiders, Wanderers, Brokers, and 'Bots. New Riders.

18. FRCC, 2001. Fisheries Resource Conservation Council. http://www.dfo-mpo.gc.ca/frcc/index.htm

19. Google search engine, 2002. http://www.google.com/.

20. Google, 2002. How to Interpret your Search Results. http://www.google.com/help/interpret.html

21. Hotka, D. 2001. Sales Tracking Application Web Site -- Buy a Car, http://www.informit.com/content/index.asp

22. Hu, Yuh-Jong. 1997. Intelligent Agent. Department of Computer Science National Chengchi University. http://www.cs.nccu.edu.tw/~jong/agent/1A/ia.html

23. Hu, Yuh-Jong. 1996. Intelligent Agent & Electronic Commerce. Department of Computer Science National Chengchi University. http://www.cs.nccu.edu.tw/~jong/agent/agent.html

24. Lovato, S. 1999. MUD - a whatis definition http://whatis.techtarget.com/definition/0,289893,sid9_gci212609,00.html

25. McBryan O. A. 1994. *GENVL and WWWW: Tools for Taming the Web*. In O. Nierstarsz, editor, Proceedings of the first International World Wide Web Conference, page 15, CERN, Geneva, http://www.cs.colorado.edu/home/mcbryan/mypapers/www94.ps

26. Microsoft Corporation. 1998. MICROSOFT VISUAL C++ 6.0 REFERENCE LIBRARY. Microsoft Press.

27. Nel L.D. 2001, 95.305 (Database Management Systems) winter 2001 Lecture Notes. Carleton University. http://www.scs.carleton.ca/~ldnel/305w2001/

28. Nolan, D. J. 2001. An application of Gowin's Vee Heuristic Model to Educational Research into Multi User Simulated Environments as a Mechanism for Applying the Story Model to Transformative and Holistic Learning Situations. http://achieve.forestry.utoronto.ca/tortoise/gowinMOO.html

29. Riordan R. 1999. Designing Relational Database Systems, Microsoft Press.

30. Suv.com. 2001. Sport Utility Safety Training: How to safely drive a Sport Utility Vehicle. http://www.suv.com/EDU/

31. Suv.org. 2001. Economic Costs: Owners and Taxpayers Foot the Bill for SUVs. http://www.suv.org/economic.html

32. Viescas, J. 1999. Running Microsoft® Access 2000. Microsoft Press.

33. Viescas, J.; Microsoft Corporation. 1999. Running Microsoft® Access 2000/ Mastering Solution Set. Microsoft Press.

34. Whatis, 2000. Whatis.com Target Search. http://whatis.techtarget.com/definition/0,,sid9_gci211538,00.html

35. Yahoo search engine, 2002. http://www.yahoo.com/.

36. Yates, R. B. and Neto, B. R., 2001. Modern Information Retrieval. Addison Wesley Longman Publishing Co. Inc. http://www.sims.berkeley.edu/~hearst/irbook/1/node4.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.

---------Table of Functions---------

```
1   // SUV.CPP - Implementation file for your Internet Server
2   //    suv Extension
3
4   #include "stdafx.h"
5   #include "suv.h"
6   #include "CRobot.h"
7   #include "CRobotInternet.h"
8
9   #include <stdlib.h>
10  #include <time.h>
11
12  CCriticalSection cs, cs2, cs3;
13  CString path = "C:\\vc_temp\\suv\\";
14  CStringArray tmp_file_list;
15  bool update = true;
16
17  ////////////////////////////////////////////////////////////////
18  // The one and only CWinApp object
19  // NOTE: You may remove this object if you alter your project to no
20  // longer use MFC in a DLL.
21
22  CWinApp theApp;
23
24  ////////////////////////////////////////////////////////////////
25  // command-parsing map
26
27  BEGIN_PARSE_MAP(CSuvExtension, CHttpServer)
28          // TODO: insert your ON_PARSE_COMMAND() and
29          // ON_PARSE_COMMAND_PARAMS() here to hook up your commands.
30          // For example:
31
32
33          DEFAULT_PARSE_COMMAND(Default, CSuvExtension)
34  //      ON_PARSE_COMMAND(Default, CHelloExtension, ITS_EMPTY)
35          ON_PARSE_COMMAND(Default, CSuvExtension, ITS_PSTR ITS_PSTR ITS_PSTR
36  ITS_PSTR ITS_PSTR)
37          ON_PARSE_COMMAND_PARAMS("r_title r_key r_area r_res number")
38  //      DEFAULT_PARSE_COMMAND(Default, CHelloExtension, ITS_PSTR ITS_PSTR ITS_PSTR
39  ITS_PSTR ITS_PSTR)
40  //      DEFAULT_PARSE_COMMAND_PARAMS("r_title r_key r_area r_res number")
41          //      DEFAULT_PARSE_COMMAND(Default, CHelloExtension, ITS_PSTR)
42
43
44
45  //      DEFAULT_PARSE_COMMAND_PARAMS("r_title r_key r_area r_res number")
46
47          ON_PARSE_COMMAND(insert, CSuvExtension, ITS_PSTR ITS_PSTR ITS_PSTR ITS_PSTR
48  ITS_PSTR ITS_PSTR ITS_PSTR)
49          ON_PARSE_COMMAND_PARAMS("title keyword area resource number filename
50  selected_items")
51
52          ON_PARSE_COMMAND(my_update, CSuvExtension, ITS_PSTR ITS_PSTR)
53          ON_PARSE_COMMAND_PARAMS("username passwd")
54
55  END_PARSE_MAP(CSuvExtension)
56
```

```
57
58  ////////////////////////////////////////////////////////////////////
59  // The one and only CSuvExtension object
60
61  CSuvExtension theExtension;
62
63
64  ////////////////////////////////////////////////////////////////////
65  // CSuvExtension implementation
66
67  CSuvExtension::CSuvExtension()
68  {
69  }
70
71  CSuvExtension::~CSuvExtension()
72  {
73  }
74
75  BOOL CSuvExtension::GetExtensionVersion(HSE_VERSION_INFO* pVer)
76  {
77          // Call default implementation for initialization
78          CHttpServer::GetExtensionVersion(pVer);
79
80          // Load description string
81          TCHAR sz[HSE_MAX_EXT_DLL_NAME_LEN+1];
82          ISAPIVERIFY(::LoadString(AfxGetResourceHandle(),
83                          IDS_SERVER, sz, HSE_MAX_EXT_DLL_NAME_LEN));
84          _tcscpy(pVer->lpszExtensionDesc, sz);
85          return TRUE;
86  }
87
88  BOOL CSuvExtension::TerminateExtension(DWORD dwFlags)
89  {
90          // extension is being terminated
91          //TODO: Clean up any per-instance resources
92          return TRUE;
93  }
94
95  ////////////////////////////////////////////////////////////////////
96  // CSuvExtension command handlers
97
98  //default method used to do keyword search
99  //it is an interface to get parameters(keywords) back from user
100 void CSuvExtension::Default(CHttpServerContext* pCtxt, char* title, char* r_key, char*
101 r_area, char* r_res, char* number)
102 {
103         CString title(r_title);
104         CString keyword(r_key);
105         CString area(r_area);
106         CString resource(r_res);
107         CString return_page;
108         int num = atoi(number);
109         return_page = do_search(title, keyword, area, resource, num);
110 //produce the result page to user
111         StartContent(pCtxt);
112         WriteTitle(pCtxt);
```

```
113             *pCtxt << return_page;
114             EndContent(pCtxt);
115     }
116
117     //a help method to do keyword search
118     //it is the real one to communicate with google search engine
119     CString CSuvExtension::do_search(CString& m_title, CString& m_key, CString& m_area,
120     CString& m_resource, int Max_records)
121     {
122             CRobotInternet internet;
123             internet.m_bReadFromCache = true;
124             internet.m_bWriteToCache = true;
125             CString sData, sData_mem, sURL, sHTML, sErrMsg, test, test3, test4, sm_Html, sm_Html1,
126     mytest1 = "" ;
127             CStringArray URLs, Titles, Contents;
128             CString content_tmp;
129             int count = 0;
130             int sum_num = 0;
131             int num = 0;
132             char buff[30];
133             bool flag = false;
134             int nResult, jj;
135
136             sData = "q=";
137             sData = sData + "%22" ;
138             test4 = m_title;
139             test4.Replace(' ','+');
140             sData = sData + test4;
141             sData = sData + "%22";
142             sData = sData + '+';
143
144             sData = sData + "%22";
145             test4 = m_area;
146             test4.Replace(' ','+');
147             sData = sData + test4;
148             sData = sData + "%22";
149             sData = sData + '+';
150
151             sData = sData + "%22";
152             test4 = m_key;
153             test4.Replace(' ','+');
154             sData = sData + test4;
155             sData = sData + "%22";
156
157             sData = sData + '+';
158             sData = sData + "%22";
159             test4 = m_resource;
160             test4.Replace(' ','+');
161             sData = sData + test4;
162             sData = sData + "%22";
163
164             sData = sData + '+';
165             sData = sData +"-filetype%3Apdf+-filetype%3Aps";
166             sData = sData + "&num=100";
```

```
167             sData = sData +
168     "&btnG=Google+Search&as_epq=&as_oq=&as_eq=&lr=&as_qdr=all&as_occt=any&as_dt=i&as_sitesear
169     ch=&safe=off&start=";
170     //          sData = sData +"&start=";
171             sData_mem = sData;
172             sData = sData + itoa(count, buff, 10);
173             sData = sData + "&sa=N&filter=0";
174     //          sData = sData + "&sa=N";
175             sURL = "www.google.com/search?"+sData;
176
177     //          CFile my_file("C:\\m_result.htm", CFile::modeCreate | CFile::modeWrite);
178
179
180             if(internet.httpGet(sURL, sHTML, nResult, sErrMsg))
181             {
182                     if(sHTML.Find("did not match any documents")!=-1)
183                     {
184                             sm_Html= "<html>\n<title>Results Page</title>\n<body
185     background=\"/backgrnd.jpg\"><center><h1>Search finished without any
186     result!</h1></center></body></html>";
187                             return sm_Html;
188                     }
189                     sHTML.Delete( 0, sHTML.Find("<p>"));
190                     if(sHTML.Find("/nav_next.gif")!=-1)
191                     {
192                     flag = true;
193     //              sHTML.Delete(sHTML.Find("<div class=n"), sHTML.GetLength()-sHTML.Find("<div
194     class=n"));
195                     sHTML.Delete(sHTML.Find("<br clear=all>"), sHTML.GetLength()-sHTML.Find("<br
196     clear=all>"));
197                     }
198                     else
199                     {
200                             sHTML.Delete(sHTML.Find("<br clear=all>"), sHTML.GetLength()-
201     sHTML.Find("<br clear=all>"));
202     //                      if(sHTML.Find("<br clear=all>")!=-1)
203     //                      sHTML.Delete(sHTML.Find("<br clear=all>"), sHTML.GetLength()-
204     sHTML.Find("<br clear=all>"));
205                     }
206
207             if(sHTML.IsEmpty()==0)
208                     {
209
210                             int e, f;
211                             CString tmp;
212                             sHTML.Replace("\n", "");
213                             sHTML.TrimLeft();
214                             sHTML.TrimRight();
215                             e = sHTML.Find("<blockquote>");
216                             while(e != -1)
217                             {
218                                     f = sHTML.Find("</blockquote>");
219                                     tmp = sHTML.Mid(e, f+13-e); // change 12 to 13
220                                     sHTML.Delete(e, tmp.GetLength());
221                                     sHTML.TrimLeft();
222                                     sHTML.TrimRight();
```

```
223                                     e = sHTML.Find("<blockquote>");
224                             }
225     //      CString xixi= "here";
226     //              return sm_Html;
227                     }
228                     sm_Html = "";
229     //              my_file.Write(sHTML, sHTML.GetLength());
230
231                     sum_num += record_num(sHTML);
232                     if(sum_num>=num){
233                             flag=false;
234                     }
235                     sm_Html = sm_Html + sHTML;
236     //              if(sm_Html.
237     //              my_file.Write(sData, sData.GetLength());
238                     while(flag)
239                     {
240                             flag = false;
241                             count += 100;
242                             sData.Empty();
243                             sData = sData_mem;
244                             sData = sData + itoa(count, buff, 10);
245     //                      sData = sData + "&sa=N&filter=0";
246                             sData = sData + "&sa=N";
247                             sURL.Empty();
248                             sURL = "www.google.com/search?"+sData;
249                             sHTML.Empty();
250                             if(internet.httpGet(sURL, sHTML, nResult, sErrMsg))
251                             {
252                             if(sHTML.Find("/nav_next.gif")!=-1)
253                             {
254                                     flag = true;
255                             }
256                             sHTML.Delete( 0, sHTML.Find("<p>"));
257                             sHTML.Delete(sHTML.Find("<br clear=all>"), sHTML.GetLength()-
258     sHTML.Find("<br clear=all>"));
259     /*                      if(flag)
260                             {
261
262     //                              sHTML.Delete(sHTML.Find("<div class=n"), sHTML.GetLength()-
263     sHTML.Find("<div class=n"));
264     //                              sHTML.Delete(sHTML.Find("<center>"), sHTML.GetLength()-
265     sHTML.Find("<center>"));
266                             }
267                             else
268                             {
269                                     sHTML.Delete(sHTML.Find("<br clear=all>"), sHTML.GetLength()-
270     sHTML.Find("<br clear=all>"));
271                                     sHTML.Delete(sHTML.Find("<p><div class=n>"),
272     sHTML.GetLength()-sHTML.Find("<p><div class=n>"));
273                             }*/
274
275     //                      my_file.Write(sHTML, sHTML.GetLength());
276
277             if(sHTML.IsEmpty()==0)
278                     {
```

```
279
280                             int e, f;
281                             CString tmp;
282                             sHTML.Replace("\n", "");
283                             sHTML.TrimLeft();
284                             sHTML.TrimRight();
285                             e = sHTML.Find("<blockquote>");
286                             while(e != -1)
287                             {
288                                     f = sHTML.Find("</blockquote>");
289                                     tmp = sHTML.Mid(e, f+13-e); // change 12 to 13
290                                     sHTML.Delete(e, tmp.GetLength());
291                                     sHTML.TrimLeft();
292                                     sHTML.TrimRight();
293                                     e = sHTML.Find("<blockquote>");
294                             }
295     //      CString xixi= "here";
296     //              return sm_Html;
297                     }
298
299                     sum_num += record_num(sHTML);
300                     if(sum_num>=num){
301                             flag=false;
302                     }
303
304                             sm_Html = sm_Html + sHTML;
305                             }
306                             else
307                             {
308                                     return sErrMsg;
309                             }
310                     }
311     ////              CString tmp = itoa(count, buff, 10);
312     //                my_file.Write(sURL, sURL.GetLength());
313     //                my_file.Close();
314     ////               test3 = test3 + test;
315
316             }
317             else
318             {
319                     sm_Html = "<html>\n<title>Results Page</title>\n<body
320     background=\"/backgrnd.jpg\"><center><h1>Network error! The job of searching can not be
321     finished.</h1></center></body></html>";
322                     return sm_Html;
323             }
324
325     /*      if(sm_Html.IsEmpty()==0)
326                     {
327
328                             int e, f;
329                             CString tmp;
330                             sm_Html.Replace("\n", "");
331                             sm_Html.TrimLeft();
332                             sm_Html.TrimRight();
333                             e = sm_Html.Find("<blockquote>");
334                             while(e != -1)
```

```
335                             {
336                                     f = sm_Html.Find("</blockquote>");
337                                     tmp = sm_Html.Mid(e, f+13-e); // change 12 to 13
338                                     sm_Html.Delete(e, tmp.GetLength());
339                                     sm_Html.TrimLeft();
340                                     sm_Html.TrimRight();
341                                     e = sm_Html.Find("<blockquote>");
342                             }
343     //      CString xixi= "here";
344     //              return sm_Html;
345                     }*/
346     //              else
347     //              {
348     //                      sm_Html = "No results found!";
349     //                      return sm_Html;
350     //              }
351             int index = 0;
352             CString check1 = "<input type=\"checkbox\" name=\"";
353             CString check2 = "\" value=\"";
354             CString check3 = "\">";
355                     if(sm_Html != "")
356                     {
357                             int a, b, c, d, t, s;
358                             CString tmp, tmp1;
359     //                      sm_Html.Empty ();
360                             sm_Html1 = sm_Html;
361     //                      sm_Html.Empty ();
362                             sm_Html1.TrimLeft();
363                             sm_Html1.TrimRight();
364                             a = 0;
365                             b = sm_Html1.Find("<p>", a+3);
366                             if( b == -1)
367                             {
368     //if there is only one record????
369                                     sm_Html = "<html>\n<title>Results Page</title>\n<body
370     background=\"/backgrnd.jpg\"><center><h1>Search finished without any
371     result!</h1></center></body></html>";
372                                     return sm_Html;
373                             }
374                             else
375                             {
376                                     num = 0;
377                             }
378                             while((b != -1)&&(num < Max_records))
379                             {
380                                     tmp = sm_Html1.Mid(a, b);
381                                     sm_Html1.Delete(a, tmp.GetLength());
382                                     sm_Html1.TrimLeft();
383                                     sm_Html1.TrimRight();
384                                     tmp.TrimLeft();
385                                     tmp.TrimRight();
386                                     if(tmp.Find("Did you mean:", a) != -1)
387                                     {
388                                             b = sm_Html1.Find("<p>", a+3);
389                                             tmp = sm_Html1.Mid(a, b);
390                                             tmp.TrimLeft();
```

```
391                                     tmp.TrimRight();
392                                 }
393                                 t = tmp.Find("<a href=")+8;
394                                 s = tmp.Find(">", t);
395                                 CString url_tmp = tmp.Mid(t, s-t);
396                                 URLs.Add(url_tmp);
397                                 t = s + 1;
398                                 s = tmp.Find("</a>", t);
399                                 CString title_tmp = tmp.Mid(t, s-t);
400                                 title_tmp.Replace("<b>", "");
401                                 title_tmp.Replace("</b>","");
402                                 Titles.Add(title_tmp);
403                                 CString check4 = check1 + itoa(index+1, buff, 10) + check2 +
404     itoa(index+1, buff, 10) + check3;
405                                 tmp.Insert(3, (LPCSTR)check4);
406                                 c = 0;
407                                 d = tmp.Find("<br>", c);
408                                 content_tmp = " ";
409                                 jj = 0;
410                                 while(d != -1)
411                                 {
412                                         tmp1 = tmp.Mid(c, d+4);
413                                         tmp.Delete(c, tmp1.GetLength());
414                                         tmp.TrimRight();
415                                         tmp.TrimLeft();
416                                         tmp1.TrimLeft();
417                                         tmp1.TrimRight();
418                                         if((tmp1.Find("File Format:")==-1)&&(tmp1.Find("Your
419     browser may not")==-1)&&(tmp1.Find("Similar pages")==-1)&&(tmp1.Find("Category:")==-1))
420                                         {
421                                                 mytest1 = mytest1 + tmp1;
422                                                 if(jj != 0)
423                                                 content_tmp = content_tmp + tmp1;
424                                         }
425                                         d = tmp.Find("<br>", c);
426                                         jj++;
427                                 }
428                                 content_tmp.Replace("<br>", " ");
429                                 content_tmp.Replace("<b>", "");
430                                 content_tmp.Replace("</b>","");
431                                 content_tmp.Replace("<font size=-1>", "");
432                                 Contents.Add(content_tmp);
433                                 b = sm_Html1.Find("<p>", a+3);
434                                 index++;
435                                 num += 1;
436                         }
437                         if(num == Max_records)
438                         {
439                                 goto there;
440                         }
441
442                         sm_Html1.TrimLeft();
443                         sm_Html1.TrimRight();
444                         CString check4 = check1 + itoa(index+1, buff, 10) + check2 + itoa(index+1,
445     buff, 10) + check3;
446                         sm_Html1.Insert(3, (LPCSTR)check4);
```

```
447                             c = 0;
448                             d = sm_Html1.Find("<br>", c);
449                             jj = 0;
450                             index++;
451                             while(d != -1)
452                             {
453                                     t = sm_Html1.Find("<a href=")+8;
454                                     s = sm_Html1.Find(">", t);
455                                     CString url_tmp = sm_Html1.Mid(t, s-t);
456                                     URLs.Add(url_tmp);
457                                     t = s + 1;
458                                     s = tmp.Find("</a>", t);
459                                     CString title_tmp = tmp.Mid(t, s-t);
460                                     title_tmp.Replace("<b>", "");
461                                     title_tmp.Replace("</b>","");
462                                     Titles.Add(title_tmp);
463                                     tmp1 = sm_Html1.Mid(c, d+4);
464                                     sm_Html1.Delete(c, tmp1.GetLength());
465                                     sm_Html1.TrimLeft();
466                                     sm_Html1.TrimRight();
467                                     tmp1.TrimLeft();
468                                     tmp1.TrimRight();
469                                     content_tmp = "";
470                                     if((tmp1.Find("File Format:")==-1)&&(tmp1.Find("Your browser may
471   not")==-1)&&(tmp1.Find("Similar pages")==-1)&&(tmp1.Find("Category:")==-1))
472                                     {
473                                             mytest1 = mytest1 + tmp1;
474                                             if(jj != 0)
475                                             content_tmp = content_tmp + tmp1;
476                                     }
477                                     d = sm_Html1.Find("<br>", c);
478                                     jj++;
479                             }
480                                     content_tmp.Replace("<br>", " ");
481                                     content_tmp.Replace("<b>", "");
482                                     content_tmp.Replace("</b>","");
483                                     content_tmp.Replace("<font size=-1>", "");
484                                     Contents.Add(content_tmp);
485                     }
486
487
488   there:
489   //                CString tmp_file_name = "";
490                     srand(time(NULL));
491                     CString file_name = itoa((rand()%1000), buff, 10);
492                     file_name +=".txt";
493                     cs.Lock();
494                     CFile my_file2(path+file_name, CFile::modeCreate | CFile::modeWrite);
495                     cs.Unlock();
496
497   //                CFile my_file2("C:\\result1.htm", CFile::modeCreate | CFile::modeWrite);
498                     CString html_tmp = "<html>\n<title>Results Page</title>\n<SCRIPT
499   LANGUAGE=\"JavaScript\">\n function change()\n {\nalert('hello');\nvar j = 0; for(var i=1;
500   i<=document.f2.number.value;i++)\n {\n tmp = document.f1.elements[i-1];\n     if(tmp.checked)\n {\n
501   document.f2.selected_items.value +=i;\n document.f2.selected_items.value +=\" \";\n      j++;\n }\n }\n
502   document.f2.number.value = j;\n alert(document.f2.number.value);\n
```

```
503  alert(document.f2.selected_items.value);\n }\n function help()\n{\n var a = window.open(\"\", \"\",
504  \"width=300 height=500\");\n var b = a.document;\n b.write(\"<html>\");\n b.write(\"<head>\");\n
505  b.write(\"<title>Usage of Intelligent Agent Search</title>\");\n b.write(\"</head>\");\n b.write(\"<body
506  background=/backgrnd.jpg>\");\n b.write(\"<center><h2>Usage of Intelligent Agent
507  Search</h2></center>\");\n b.write(\"<br><hr><br>\");\n b.write(\"<li>Step 1: After receiving the search
508  results, users select records to be inserted into database by clicking on the check box corresponding to the
509  entry.\");\n b.write(\"<li>Step 2: Selected records are automatically submitted to the database by pressing
510  the Submit button at the bottom of the page.\");\n b.write(\"<br><br><hr><br>\");\n
511  b.write(\"Note:<br>\");\n b.write(\"Users can not submit the form to update the database without first
512  selecting at least one entry from the list. The Reset button at the bottom of the page cancels existing
513  selections.\");\n b.write(\"</body>\");\n b.write(\"</html>\");\n }\n</SCRIPT>\n <body
514  background=\"/backgrnd.jpg\"><Center><H1>Select Results</H1></center>\n<hr><b><h2>Usage of
515  Intelligent Agent Search</h2></b><br><li><b>Step 1:</b> After receiving the search results, users select
516  records to be inserted into database by clicking on the check box corresponding to the entry. <li><b>Step
517  2:</b> Selected records are automatically submitted to the database by pressing the Submit button at the
518  bottom of the page.<br><b>Note:</b><br>";
519       html_tmp = html_tmp + "Users can not submit the form to update the database without first selecting
520  at least one entry from the list. The Reset button at the bottom of the page cancels existing
521  selections.<hr><form method=\"POST\" name=\"f1\" action=\"--WEBBOT-SELF--\">";
522  //                  my_file2.Write(html_tmp, html_tmp.GetLength());
523  //                  my_file2.Write(mytest1, mytest1.GetLength());
524                      mytest1 = html_tmp + mytest1;
525                      html_tmp = "</form>\n";
526                      html_tmp += "<form method=\"POST\" name=\"f2\"
527  action=\"/suv_scripts/suv.dll?insert\">\n<p><input type=\"hidden\" name=\"title\" value=\"";
528                      html_tmp += m_title;
529                      html_tmp += "\">\n<input type=\"hidden\" name=\"keyword\" value=\"";
530                      html_tmp += m_key;
531                      html_tmp += "\">\n<input type=\"hidden\" name=\"area\" value=\"";
532                      html_tmp += m_area;
533                      html_tmp += "\">\n<input type=\"hidden\" name=\"resource\" value=\"";
534                      html_tmp += m_resource;
535                      html_tmp += "\">\n<input type=\"hidden\" name=\"number\" value=\"";
536                      html_tmp += itoa(index, buff, 10);
537                      html_tmp += "\">\n<input type=\"hidden\" name=\"filename\" value=\"";
538                      html_tmp += file_name;
539                      html_tmp += "\">\n<input type=\"hidden\" name=\"selected_items\" value=>";;
540                      html_tmp += "<p align=\"center\"><input type=\"submit\" value=\"Submit\" onClick =
541  change()>     <input type=\"reset\"
542  value=\"Reset\">     <input type=\"button\" value=\"Help\"
543  onClick=help()></p>\n</form>\n</body></html>";
544                      mytest1 += html_tmp;
545
546                      CString tmp = "\n";
547                      int j;
548                      for(j=0; j< index; j++)
549                      {
550                              CString tmpIndex = itoa(j+1, buff, 10);
551                              my_file2.Write(tmpIndex, tmpIndex.GetLength());
552                              my_file2.Write(tmp, tmp.GetLength());
553                              my_file2.Write(URLs[j], URLs[j].GetLength());
554                              my_file2.Write(tmp, tmp.GetLength());
555                              my_file2.Write(Titles[j], Titles[j].GetLength());
556                              my_file2.Write(tmp, tmp.GetLength());
557                              my_file2.Write(Contents[j], Contents[j].GetLength());
558                              my_file2.Write(tmp, tmp.GetLength());
```

```
559                     //          my_file2.Write(tmp, tmp.GetLength());
560                     }
561                     tmp = "EOF";
562                     my_file2.Write(tmp, tmp.GetLength());
563                     my_file2.Close();
564                     tmp_file_list.Add(file_name);
565
566         return mytest1;
567     }
568
569     //the method used to insert record selected by user into database
570     //it is an interface used to get user's selection
571     void CSuvExtension::insert(CHttpServerContext* pCtxt, char* title, char* keyword, char*
572     area, char* resource, char* number, char* filename, char* selected_items)
573     {
574
575             CString r_title(title);
576             CString r_keyword(keyword);
577             CString r_area(area);
578             CString r_resource(resource);
579             int r_number = atoi(number);
580             CString r_filename(filename);
581             CString r_selected_items(selected_items);
582             CString return_page;
583             return_page = do_insert(r_title, r_keyword, r_area, r_resource, r_number, r_filename,
584     r_selected_items);
585             StartContent(pCtxt);
586             WriteTitle(pCtxt);
587             *pCtxt << _T("<br><br>");
588             *pCtxt << return_page;
589             EndContent(pCtxt);
590     }
591
592     //a help method used to insert records into database automatically
593     //it is the real one to do insertion job
594     CString CSuvExtension::do_insert(CString& n_title, CString& n_keyword, CString&
595     n_area, CString& n_resource, int& n_number, CString& n_filename, CString&
596     n_selected_items)
597     {
598             CString tmp_index="", tmp_return="<html>\n<head>\n<title>/nInsertion Reulsts Feedback
599     page</title>\n</head>\n<body background=/backgrnd.jpg>\n<center><h1>Insertion Results
600     Feedback</h1><br><hr></center>\n", tmp1 = "\n", statement, max_index;
601             CString exist_items="";
602             CString feedback;
603             CString non_exist_items="";
604             CStringArray index, url, title, content;
605             char buff[30];
606             bool* record_status = new bool[n_number];
607             bool flag = false;
608             int c, a, b, z=0;
609             int exist_num = 0;
610             c = n_selected_items.Find(" ", 0);
611             while(c != -1){
612                             index.Add(n_selected_items.Mid(0, c));
613                             n_selected_items.Delete(0, c+1);
```

```
614                         c = n_selected_items.Find(" ", 0);
615                         z++;
616             }
617             CStdioFile f;
618             cs2.Lock();
619             CString my_file = path + n_filename;
620             cs2.Unlock();
621             f.Open(_T(my_file), CFile::modeRead);
622             c = 0;
623             CString tag("EOF");
624             while(tmp_index.Compare(tag) != 0 ){
625             if(tmp_index.Compare(index.GetAt(c)) == 0){
626                     f.ReadString(tmp_index);
627                     url.Add(tmp_index);
628                     f.ReadString(tmp_index);
629                     title.Add(tmp_index);
630                     f.ReadString(tmp_index);
631                     content.Add(tmp_index);
632                     c++;
633             }
634                     f.ReadString(tmp_index);
635             }
636                     f.Close();
637                     f.Remove(_T(my_file));
638
639             CRobotDatabase db;
640     //establish the connect with database through odbc
641             if(!db.Open("suv", "Record","sa","")){
642                     tmp_return += "<p>Sorry!, cannot open database \"Record\"!<br>\n";
643             }
644             else{
645                     db.m_nFields = 8;
646                     db.m_sFieldName[0] = "RecordID";
647                     db.m_bKeyField[0] = true;
648                     db.m_bNumeric[0] = true;
649
650                     db.m_sFieldName[1] = "Title";
651                     db.m_bKeyField[1] = false;
652                     db.m_bNumeric[1] = false;
653
654                     db.m_sFieldName[2] = "URL";
655                     db.m_bKeyField[2] = false;
656                     db.m_bNumeric[2] = false;
657
658                     db.m_sFieldName[3] = "Description";
659                     db.m_bKeyField[3] = false;
660                     db.m_bNumeric[3] = false;
661
662                     db.m_sFieldName[4] = "g_area";
663                     db.m_bKeyField[4] = false;
664                     db.m_bNumeric[4] = false;
665
666                     db.m_sFieldName[5] = "resource";
667                     db.m_bKeyField[5] = false;
668                     db.m_bNumeric[5] = false;
669
```

```
670                     db.m_sFieldName[6] = "citation";
671                     db.m_bKeyField[6] = false;
672                     db.m_bNumeric[6] = false;
673
674                     db.m_sFieldName[7] = "date";
675                     db.m_bKeyField[7] = false;
676                     db.m_bNumeric[7] = false;
677
678                     for(int k=0; k<n_number;k++){
679
680                             statement = "URL=\'" + _T(url.GetAt(k)) +"\'";
681                             if(db.SelectMatchingRecords(statement)){
682                                     record_status[k] = true;
683                             }
684                             else{
685                                     record_status[k] = false;
686                                     flag = true;
687                             }
688                     }
689             }
690                             for(int j = 0; j < n_number; j++){
691                             if(record_status[j]==true){
692                             exist_items += "<p>";
693                             exist_items +=url[j];
694                             exist_items += "<br>\n";
695                             exist_items += title[j];
696                             exist_items += "<br>\n";
697                             exist_items += content[j];
698                             exist_num++;
699                             }
700                             else{
701                             non_exist_items += "<p>";
702                             non_exist_items +=url[j];
703                             non_exist_items += "<br>\n";
704                             non_exist_items += title[j];
705                             non_exist_items += "<br>\n";
706                             non_exist_items += content[j];
707                             }
708                     }
709                     statement = "recordid = (select max(recordid) from record)";
710                     bool bHaveData = db.SelectMatchingRecords(statement);
711                      while(bHaveData){
712                             bHaveData = db.NextMatchingRecord();
713                      }
714                     max_index = db.m_sFieldValue[0];
715                     feedback += db.m_sFieldValue[1];
716                     feedback += max_index;
717                     a = atoi(_T(max_index));
718                     b = a;
719     //produce the sql command
720       for(int i=0; i<z;i++){
721                     if(record_status[i]==false){
722                             a++;
723                             statement = "insert into Record values(";
724                             statement += itoa(a, buff, 10);
725                             statement += ", ";
```

```
726                         statement += "\'";
727                         if(title[i].Find("\'",0)){
728                                 title[i].Replace("\'", "");
729                         }
730                         statement += title[i];
731                         statement += "\'";
732                         statement += ", ";
733                         statement += "\'";
734                         if(url[i].Find("\'",0)){
735                                 url[i].Replace("\'", "");
736                         }
737                         statement += url[i];
738                         statement += "\'";
739                         statement += ", ";
740                         statement += "\'";
741                         if(content[i].Find("\'",0)){
742                                 content[i].Replace("\'", "");
743                         }
744                         statement += content[i];
745                         statement += "\'";
746                         statement += ", ";
747                         statement += "\'";
748                         statement += n_area;
749                         statement += "\'";
750                         statement += ", ";
751                         statement += "\'";
752                         statement += n_resource;
753                         statement += "\'";
754                         statement += ", ";
755                         statement += "\'...\'";
756                         statement += ", ";
757                         statement += "date()";
758                         statement += ")";
759                         if(!db.Execute(statement)){
760                                 tmp_return += "db error msg: ";
761                                 tmp_return += db.m_sErrMsg;
762                                 tmp_return +="<br>\n";
763                         }
764
765                         statement = "insert into have_keyword values(";
766                         statement += itoa(a, buff, 10);
767                         statement += ", \'";
768                         statement += n_keyword;
769                         statement += "\')";
770                         if(!db.Execute(statement)){
771                                 tmp_return += "db1 error msg: ";
772                                 tmp_return += db.m_sErrMsg;
773                                 tmp_return +="<br>\n";
774                         }
775                 }
776         }
777         db.Close();
778         tmp_return += "<p>";
779         tmp_return += itoa(z, buff, 10);
780         tmp_return += " URLs selected.";
781         tmp_return += "<p>";
```

```
782             tmp_return += itoa(z-exist_num, buff, 10);
783             tmp_return += " URLs inserted.";
784             tmp_return += "<p><h3>Following records already exist in database:</h3>";
785             tmp_return += exist_items;
786             tmp_return += "<p><h3>Following records were inserted into database successfully:</h3>";
787             tmp_return += non_exist_items;
788             tmp_return += feedback;
789             tmp_return += "<br>\n</body></html>";
790            delete [] record_status;
791            return tmp_return;
792    }
793
794    //the method used to update all records in database
795    //delete those records whose url doesn't exist anymore.
796    void CSuvExtension::my_update(CHttpServerContext* pCtxt, char* username, char*
797    passwd)
798    {
799            CRobotInternet internet;
800            CString sURL, sHeader, sErrMsg, index, return_page;
801            int nResult;
802            int del_num = 0;
803            int sum = 0;
804            char buff[30];
805            CString statement;
806            CString my_username = "lane";
807            CString my_passwd = "q23456";
808            CString tmp_username(username);
809            CString tmp_passwd(passwd);
810            CString msg_f = "<html>\n<head>\n<title>/nReulst page\n</title>\n</head>\n<body
811    background=/backgrnd.jpg>\n<h2>Sorry, there is something wrong with your username and passwd,
812    please try again!</h2>\n</body>\n</html>\n";
813            CString msg_s1 = "<html>\n<head>\n<title>/nReulst page\n</title>\n</head>\n<body
814    background=/backgrnd.jpg>\n<h2>Updating of database is finished successfully!</h2><br><p>There are
815    now ";
816            CString msg_s2 = " records in the database. ";
817            CString msg_s3 = " records have been deleted.\n</body>\n</html>\n";
818            CString msg_d = "<html>\n<head>\n<title>/nReulst page\n</title>\n</head>\n<body
819    background=/backgrnd.jpg>\n<h2>Sorry, We can not finish the Update operation at this moment, please
820    try again later! The error message is:</h2><br>\n";
821
822            if((my_username != tmp_username)||(my_passwd != tmp_passwd)){
823                    return_page = msg_f;
824            }
825            else{
826                    CRobotDatabase db;
827                    if(!(db.Open("suv","Record","sa","""))){
828                            return_page = msg_d + db.m_sErrMsg;
829                            return_page += "</body>\n</html>\n";
830                            goto there;
831                    }
832                    else{
833                     db.m_nFields = 8;
834                     db.m_sFieldName[0] = "RecordID";
835                     db.m_bKeyField[0] = true;
836                     db.m_bNumeric[0] = true;
837
```

```
838                     db.m_sFieldName[1] = "Title";
839                     db.m_bKeyField[1] = false;
840                     db.m_bNumeric[1] = false;
841
842                     db.m_sFieldName[2] = "URL";
843                     db.m_bKeyField[2] = false;
844                     db.m_bNumeric[2] = false;
845
846                     db.m_sFieldName[3] = "Description";
847                     db.m_bKeyField[3] = false;
848                     db.m_bNumeric[3] = false;
849
850                     db.m_sFieldName[4] = "g_area";
851                     db.m_bKeyField[4] = false;
852                     db.m_bNumeric[4] = false;
853
854                     db.m_sFieldName[5] = "resource";
855                     db.m_bKeyField[5] = false;
856                     db.m_bNumeric[5] = false;
857
858                     db.m_sFieldName[6] = "citation";
859                     db.m_bKeyField[6] = false;
860                     db.m_bNumeric[6] = false;
861
862                     db.m_sFieldName[7] = "date";
863                     db.m_bKeyField[7] = false;
864                     db.m_bNumeric[7] = false;
865                     statement = "Title<>'";
866                     bool bHaveData = db.SelectMatchingRecords(statement);
867                     while(bHaveData){
868                             sum++;
869                             index = db.m_sFieldValue[0];
870                             sURL = db.m_sFieldValue[2];
871                             if(!internet.httpHeader(sURL, sHeader, nResult, sErrMsg)){
872                                     statement = "delete * from Record where RecordID=";
873                                     statement += index;
874                                     db.Execute(statement);
875                                     del_num++;
876                             }
877                             bHaveData = db.NextMatchingRecord();
878                     }
879                     }
880             return_page = msg_s1 + _itoa(sum-del_num, buff, 10);
881             return_page += msg_s2;
882             return_page += _itoa(del_num, buff, 10);
883             return_page += msg_s3;
884             }
885     there:
886             StartContent(pCtxt);
887             WriteTitle(pCtxt);
888             *pCtxt << return_page;
889             EndContent(pCtxt);
890     }
891
892     int CSuvExtension::record_num(CString& page)
893     {
```

```
894             int pos = 0;
895             int number = 0;
896             while(page.Find("<p>",pos)!=-1)
897             {
898                     number++;
899                     pos += 3;
900             }
901     return number;
902
903     }
904
905
906
907     // Do not edit the following lines, which are needed by ClassWizard.
908     #if 0
909     BEGIN_MESSAGE_MAP(CSuvExtension, CHttpServer)
910             //{{AFX_MSG_MAP(CSuvExtension)
911             //}}AFX_MSG_MAP
912     END_MESSAGE_MAP()
913     #endif   // 0
914
915
916
917     ////////////////////////////////////////////////////////////////////
918     // If your extension will not use MFC, you'll need this code to make
919     // sure the extension objects can find the resource handle for the
920     // module.  If you convert your extension to not be dependent on MFC,
921     // remove the comments arounn the following AfxGetResourceHandle()
922     // and DllMain() functions, as well as the g_hInstance global.
923
924     /****
925
926     static HINSTANCE g_hInstance;
927
928     HINSTANCE AFXISAPI AfxGetResourceHandle()
929     {
930             return g_hInstance;
931     }
932
933     BOOL WINAPI DllMain(HINSTANCE hInst, ULONG ulReason,
934                                                     LPVOID lpReserved)
935     {
936             if (ulReason == DLL_PROCESS_ATTACH)
937             {
938                     g_hInstance = hInst;
939             }
940
941             return TRUE;
942     }
943
944     ****/
```

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 of Functions----------

```
<SCRIPT LANGUAGE="JavaScript">
// 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=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=/backgrnd.jpg>");
        b.write("<center><h2>Usage of Agent Search</h2></center>");
        b.write("<br><hr><br>");
        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
comboboxes.");
        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><br><hr><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 waitting 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.f1.elements[0].value == ""||document.f1.elements[0].value=="null")
        {
        document.f1.elements[0].value = prompt("Search Title can not be blank!\nPlease 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=/backgrnd.jpg>");
        b.write("<center><h2>Update Database</h2></center>");
        b.write("<hr><br>");
        b.write('<form method="post" action="/suv_scripts/suv.dll?my_update"');
        b.write('<p>Username: <input type=text name="username" value="" size="20">');
        b.write('<p>Password: <input type=password name="passwd" value="" size="20">');
```

```
53              b.write('<p><input type="submit" value="Submit">  <input type="reset"
54      value="Reset">')
55              b.write("</form>");
56              b.write("<br><br><hr><br>");
57              b.write("</body>");
58              b.write("</html>");
59      }
60
61      </SCRIPT>
62      <SCRIPT LANGUAGE="JavaScript">
63
64      //the method used to let user cancel all selection made before

65      function re_set()
66       { document.f1.reset();
67       }
68
69      //the method used to record user s selection and produce the parameters which will be //passed to server

70       function change()
71       {
72      alert('hello');
73      var j = 0; for(var i=1; i<=document.f2.number.value;i++)
74       {
75      tmp = document.f1.elements[i-1];
76              if(tmp.checked)
77       {
78      document.f2.selected_items.value +=i;
79      document.f2.selected_items.value +=" ";
80              j++;
81       }
82       }
83      document.f2.number.value = j;
84      alert(document.f2.number.value);
85      alert(document.f2.selected_items.value);
86       }
87
88      // the method used to prompt out a help windows to display the contents of the usage of //Agent Search

89       function help()
90      {
91       var a = window.open("", "", "width=300 height=400");
92       var b = a.document;
93       b.write("<html>");
94       b.write("<head>");
95       b.write("<title>Usage of Spider Search</title>");
96       b.write("</head>");
97       b.write("<body background=/backgrnd.jpg>");
98       b.write("<center><h2>Usage of Spider Search</h2></center>");
99       b.write("<br><hr><br>");
100      b.write("<li>Step 1: After receiving the search results, users select records to be inserted into database by
101     clicking on the check box corresponding to the entry.");
102      b.write("<li>Step 2: Selected records are automatically submitted to the database by pressing the Submit
103     button at the bottom of the page.");
```

```
104     b.write("<br><br><hr><br>");
105     b.write("Note:<br>");
106     b.write("Users can not submit the form to update the database without first selecting at least one entry from
107    the list. The Reset button at the bottom of the page cancels existing selections.");
108     b.write("</body>");
109     b.write("</html>");
110     }
111    </SCRIPT>
```

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.

```
1   //bring add-drop page to user and retrieval related information from database and display
2   //them in corresponding combo box
3   s_add_drop.idc
4   s_add_drop.htx
5   //bring the location template to user and let user to input record manually
6   s_add_new_area.idc
7   s_add_new_area.htx
8   //insert keyword into database manually and display result of insertion
9   s_add_new_keyword.idc
10  s_add_new_keyword.htx
11  //insert record into database manually and display result of insertion
12  s_Add_new_record.idc
13  s_Add_new_record.htx
14  //bring the vehicle type template to user and let user to input record manually
15  s_add_new_resource.idc
16  s_add_new_resource.htx
17  //bring the record template to user and let user to insert record into database manually
18  s_add_record.idc
19  s_add_record.htx
20  // let user to establish the connection between keyword and record
21  s_add_relation.idc
22  s_add_relation.htx
23  ////bring the delete page to user, retrieval related information from database and display
24  //them in corresponding combo box
25  s_delete.idc
26  s_delete.htx
27  //retrieval locations out from database and let user to select one to delete
28  s_delete_area.idc
29  s_delete_area.htx
30  //retrieval keywords out from database and let user to select one to delete
31  s_delete_keyword.idc
32  s_delete_keyword.htx
33  //retrieval records out from database and let user to select one to delete
34  s_delete_record.idc
35  s_delete_record.htx
36  //retrieval vehicle type out from database and let user to select one to delete
37  s_delete_resource.idc
38  s_delete_resource.htx
39  // let user to drop the connection between keyword and record
40  s_drop_relation.idc
41  s_drop_relation.htx
42  //bring the modify page to user, retrieval related information from database and display
43  //them in corresponding combo box
44  s_modify.idc
45  s_modify.htx
46  //make change of keyword information and display the result of change to user
```

```
47  s_modify_keyword.idc
48  s_modify_keyword.htx
49  //make change of record information and display the result of change to user
50  s_modify_record.idc
51  s_modify_record.htx
52  //retrieval corresponding keyword out from database to let user make change
53  s_need_modify_keyword.idc
54  s_need_modify_keyword.htx
55  //retrieval corresponding record out from database to let user make change
56  s_need_modify_record.idc
57  s_need_modify_record.htx
58  //bring the query page to user, retrieval related information from database and display
59  //them in corresponding combo box
60  s_query.idc
61  s_query.htx
62  //do query by area in database and display result to user
63  s_querybyarea.idc
64  s_querybyarea.htx
65  //do query by input date in database and display result to user
66  s_querybydate.idc
67  s_querybydate.htx
68  //do query by keyword in database and display result to user
69  s_querybykid.idc
70  s_querybykid.htx
71  //do query by vehicle type in database and display result to user
72  s_querybyres.idc
73  s_querybyres.htx
74  //do query by record title in database and display result to user
75  s_querybyrid.idc
76  s_querybyrid.htx
77  //do query by input word in record title in database and display result to user
78  s_querybytitle.idc
79  s_querybytitle.htx
80  //bring search page to user and retrieval related information from database and display
81  //them in corresponding combo box
82  s_search_page.idc
83  s_search_page.htx
```