An Intelligent Voice Driven Intranet Search Engine

(AIVDISE)

by

Salaheldin Ali Aboulkhasam

B.Sc., El_Fateh University, Libya, 1991

Thesis

Submitted in partial fulfillment of the requirement
for the degree of Master of Science (Computer Science)

Acadia University
Spring Convocation 2000

Abstract

The volume of website information accumulates rapidly, and thus, search engines are indispensable. The main user demand from the search engine is to efficiently retrieve accurate and relevant results. This thesis presents a new and improved search engine called Intelligent Voice Driven Intranet Search Engine (AIVDISE) which accepts natural language queries. AIVDISE can be used to search a personal or business website, e.g. Acadia's website. As input, AIVDISE can accept both spoken and typed English language sentences. Together with a number of uniquely powerful search engine technologies, AIVDISE can deliver the accuracy people want from their intranet search engine without the need for expert training in search techniques.

Acknowledgements

Although only my name appears on this thesis, this work would not be possible without the help of many people. First, I would like to thank Dr. Andre Trudel, my advisor, who helped me at every stage of this work. His suggestions and innovative ideas were invaluable for the completion of this work. I am also grateful to Dr. Scott Goodwin and Dr. Daniel L. Silver for their valuable suggestions and comments during thesis discussions. I would also like to thank the other faculty members of the Jodrey School of Computer Science for helping me gain better knowledge in the different fields of computer science. I would like to thank Carol Luczak VanIderstine, a master student in The English department, for spending time proofreading. Finally, I would like to thank my parents and my wife for their support and encouragement.

Table of Contents:

List Of Figures:

List of Tables:

To my son Amgid

Chapter 1 Introduction

As the amount of information increases on personal homepages, specific information may be too difficult for many users to find without a special tool. These tools are called intranet search engines. Existing search engines use keyword searching which often does not provide the user with the right information. For example, if we search using the keyword "agent" on the search engine Yahoo we get 7051 hits.

The ultimate goal of intranet search engine programmers is to make querying a website so simple that anyone can easily access information. Natural language search engines are working towards that goal. Natural language (NL) is defined as a language spoken or written by humans. NL input is one of the most difficult problems for artificial intelligence because human language is complex, irregular, and diverse.

In a natural language query, a searcher uses a non-structured language, such as English, to conduct a search by describing the desired information. An example of a natural language query is "I need to know how search engines work." A natural language search does not require the user to use boolean logic. A natural language search engine uses relevance ranking and other AI techniques to extract the essence of the query, conduct the search, and return the relevant information.

NL searching has certain overall advantages over Boolean keyword searching. For example, Turtle [Turt 94] compared the performance of natural language and

Boolean query formulations. Natural language formulations worked better and scaled up better. Experienced searchers find that NL searches work better than keywords [Hersh and Day 97].

To encourage people to make full use of resources on homepages, the search engine is required to be easy to use because the average user will likely never become an expert with search software. At the same time, if we are to expect users to visit our homepage and continue using our search engine, we must provide accurate and relevant search results.

This thesis introduces An Intelligent Voice Driven Intranet Search Engine (AIVDISE) for a personal or business website e.g., Acadia's website. AIVDISE can accept spoken and typed English language sentences as input. To increase the accuracy and speed of the search engine, it will store and use specialized knowledge about the Acadia website. We built our search engine using a single and easy user interface to encourage people to keep using the search engine.

We chose Java to implement AIVDISE because Java is designed to run on a variety of platforms. Most computer programs are written for specific kinds of hardware and operating systems. Thus, some applications written for Windows will not run on a Macintosh and vice versa. Java is suitable because once it is written, Java runs on any computer that has the Java Virtual Machine. Because Java is so widely accepted, it has become a popular tool on the internet. Java is guaranteed to work with virtually any browser.

1.1 Thesis Overview

Chapter 2: **What are Search Engines?** We introduce the reader to search engines, their history, where we can use them, and why we need them. We discuss some popular intranet and internet search engines, such as ht://dig, Phantom, Altavista, Yahoo, WebEnhancer, Searchlink, etc.

Chapter 3: **How Search Engines Work?** In this chapter, we discuss different types of search engines, examine how search engines work, the hidden mechanics, and how a page can be found. Furthermore, we discuss why there are so many dead links (sites are no longer in existence) returned from popular search engines.

Chapter 4: **AIVDISE Overview**. In this chapter, we present the strategies used to build AIVDISE and our system design. We begin with a brief overview of what we want to accomplish. Then, we describe the high level architecture of our system followed by a step-by-step presentation of our strategies.

Chapter 5: **Results.** In this Chapter, we demonstrate the performance of AIVDISE through examples.

Chapter 6: **Implementation.** This chapter describes the system implementation, we discuss the most important classes used by AIVDISE through code fragments.

Chapter 7: **Conclusion.** This chapter presents conclusions and directions for future work.

Chapter 2. What are Search Engines?

Introduction

In this chapter, we introduce the reader to the description of search engines, their history, where we can use them, and why we need them. We discuss some popular intranet and internet search engines, such as ht://dig, Phantom, Altavista, Yahoo, WebEnhancer, Searchlink, etc.

2.1 The history of search engines

The first attempts at organizing information on the net began in 1990 when Alan Emtage of McGill University developed a search engine named Archie [Corn 94]. At the time, Archie was the most popular repository of internet files and anonymous FTP (File Transfer Protocol) sites.

After the creation of Archie, the University of Nevada created the Very Easy Rodent-Oriented NetWare Index to Computerized Archives (Veronica) [Maci98]. Veronica works in a manner similar to that of Archie. In 1993, Jughead [Maci98] was developed at the University of Utah (USA). Jughead, another Gopher search facility, is similar to Veronica. However, Jughead is less sophisticated and is intended for searching a smaller Gopher area. As the files on Gopher servers are gradually converted to HTML

files. Gopher (and therefore Veronica and Jughead) will become less important. Currently, many valuable files can be found only on Gopher servers. Gopher servers contain plain text documents, but no images or hypertext.

The web search engines did not come into existence until April 1994 [Maci98]. David Fill and Jerry Yang, two Stanford University Ph.D. students, built the first search engine to organize their personal interests on the internet. Fill and Yang found it was fun searching the internet with the search engine because it was so easy and they would not have to remember all of the addresses. Originally, the project was called Guide to the World Wide Web but later was renamed "Yahoo" which stands for "yet another hierarchical officious oracle." Yahoo was converted into a database to help others locate and identify sites. In April 95, Fill and Yang received one million dollars from Sequoia Capital to develop Yahoo into a business. The students left graduate school and are now part of a new breed of entrepreneurs who became overnight millionaires.

Lycos was founded in July 1994. The word Lycos stems from the first five letters of the Latin, Lycosidae (the Wolf Spider). Lycos was originally designed and implemented at Carnegie Mellon University, USA. America Online bought Lycos and became Lycos Inc.

In December 1995, Altavista was released from Digital's Research Laboratories in Palo Alto, California.

Currently, there are a myriad of search engines available on the web, such as Harvest 1996, Excite 1995, Looksmart 1996, and Copernic1999.

2.2 Search Engine

The World Wide Web is the largest existing collection of documents. . According to Search Engine Watch, the Web had 800 million pages in August 1999. Information is obtained as follows. If the URL is known, the user enters the URL in the "location" box on a browser, such as Netscape Navigator or Internet Explorer. However, if the URL is not known, a search engine is used.

A search engine is software that searches for information. The search engine has an interface that allows you to type keywords about the required information into a blank field, then gives you a list of addresses. These addresses are presented in hypertext, which allows the user to click on any address. If the contents of the address selected do not have the correct information, the user can click the back button on the browser and return to the original list. There are different types of search engines which can be used in either an internet or intranet environment.

2.2.1 Environments of the search engine

We can use the search engine in two environments (internet or intranet) see Figure 2.1.We discus them in detail next.

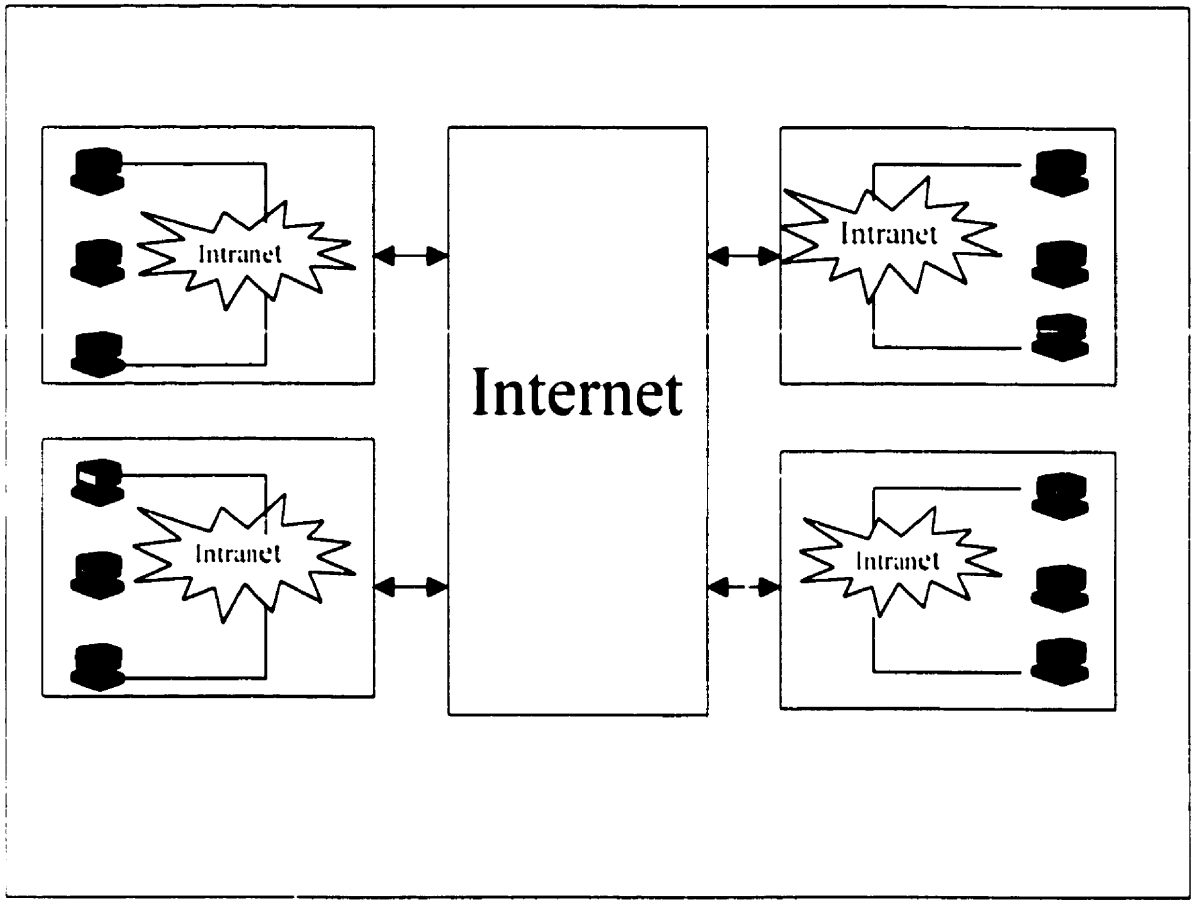

Figure 2.1 Search Engine environments

2.2.1.1 Internet Search Engines

An internet search engine is software designed to search the internet (see Figure 2.1). Although there are hundreds of different internet search engines, the most popular search engines are Altavista, Lycos, and Excite

2.2.1.1.1 Altavista Search Engine

Altavista contains over 150 million web pages in an index file. It is a full-text index that searches the entire HTML file. Altavista has both a simple search (Figure 2.2), and advanced search user interfaces (Figure 2.3). Altavista uses boolean logic to search. In simple searching, the default is OR logic between words. For example, assume the user wanted to find information on "hard rock music" and typed in the keywords:

hard rock.

The search engine presents the user with files that contain only "hard," "rock," and "hard rock". The user is potentially only interested in files containing both "hard" and "rock". In advanced searching, OR, AND, NOT may be used. Case sensitivity, in which both upper and lower case characters are searched, may also be used.

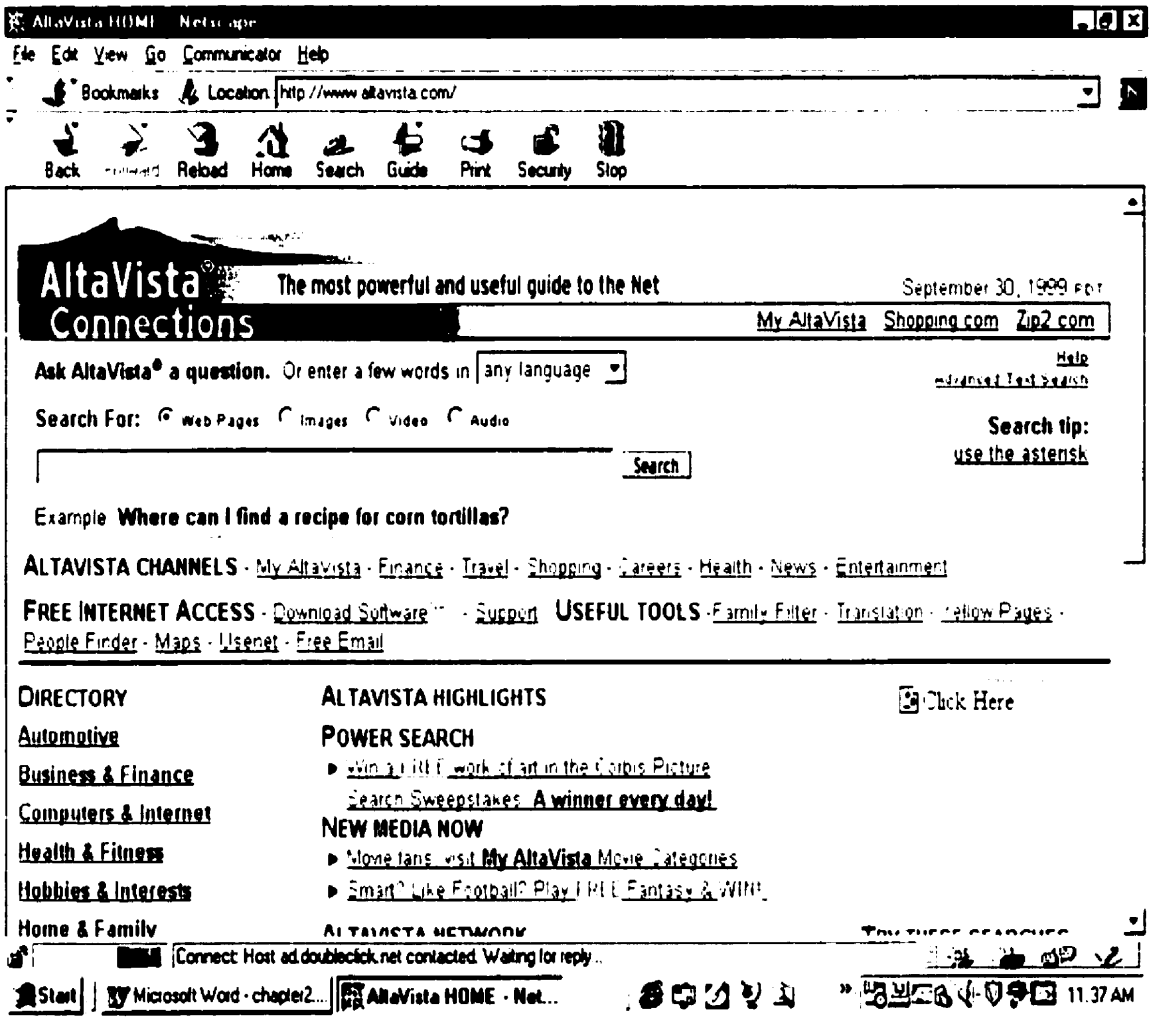

Figure 2.2 Altavista simple user interface

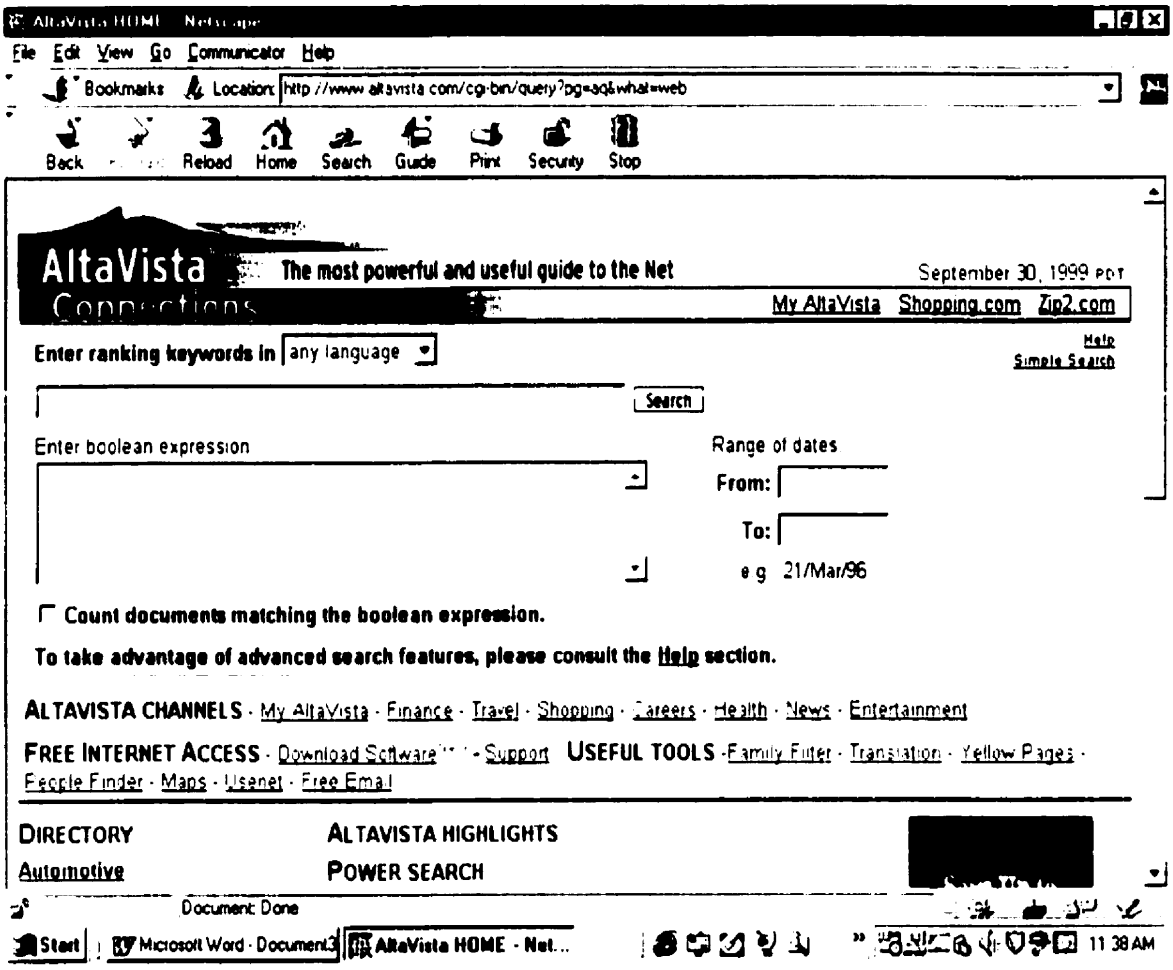

Figure 2.3 Altavista advanced user interface

2.2.1.1.2 Northern Light Search Engine

In its index file, Northern Light contains over 160 million web pages. The default between keywords is AND. For example, assume the user wanted to find information on "hard exam" and typed in the keywords:

hard exam.

The search engine presents the user with files that contain both "hard" and "exam". You can also use boolean logic OR and NOT. Phrase searching requires quotation marks around each phrase. Users can search Northern Light's special collection which is an

index of articles from 3000 journals and periodicals. Searching the special collection is free but retrieving the text of an article requires payment.

2.2.1.1.3 Excite Search Engine

There are 55 million pages in the Excite database. Excite has simple and advanced interfaces as shown in Figures 2.4 and 2.5, respectively. In the simple interface, OR is used as a default. In the advanced interface, AND and NOT are used but they must be entered in capital letters.

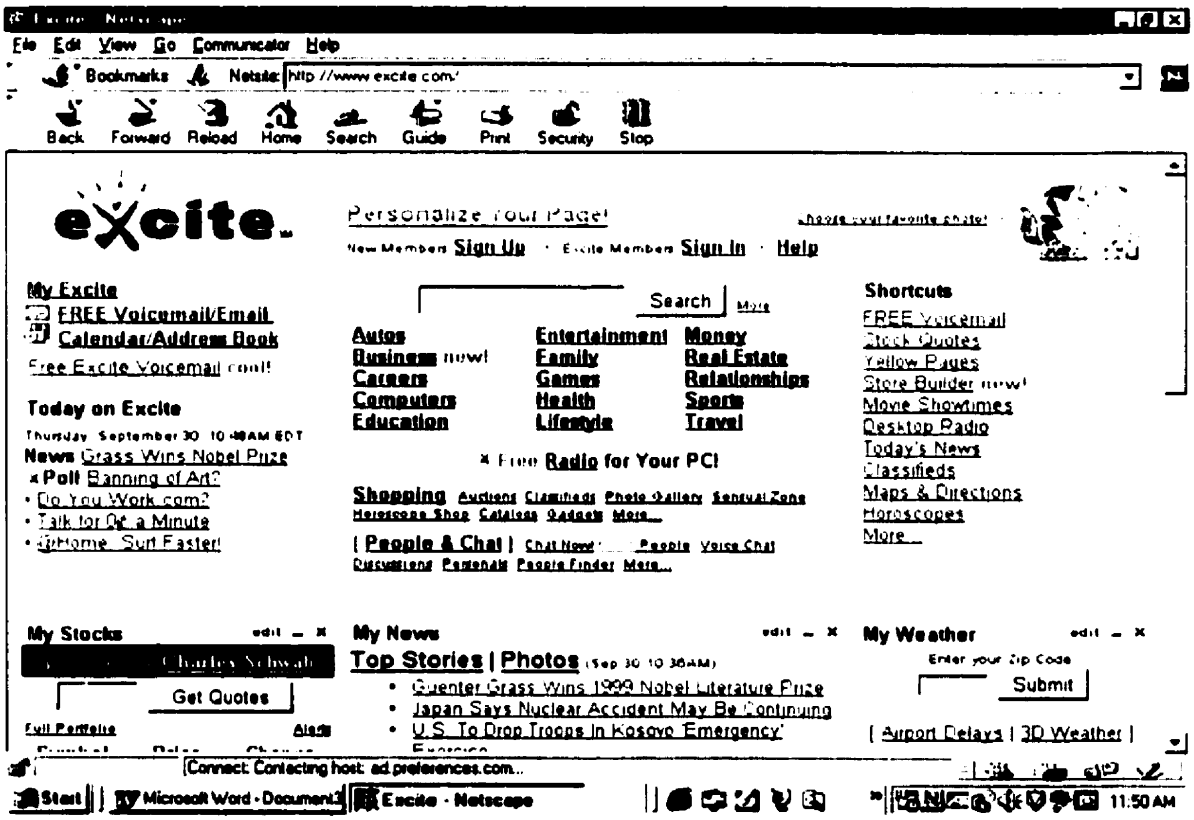

Figure 2.4 Excite simple user interface

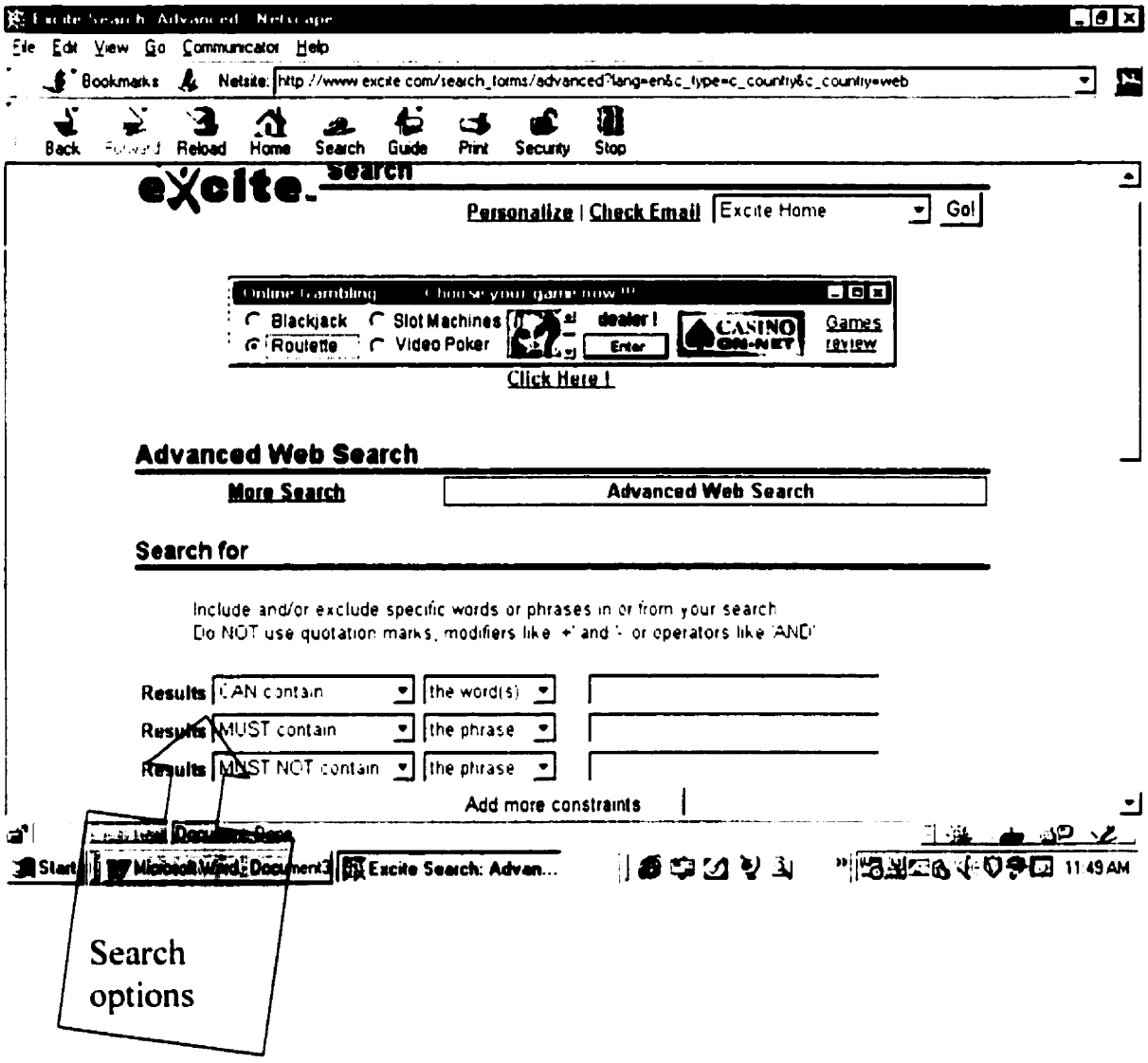

Figure 2.5 Excite advanced user interface

2.2.1.1.4 HotBot Search Engine

HotBot contains over 110 million sites. HotBot has a single interface which allows the user to modify a search term from a drop-down list and choose advanced search options (Figure 2.6). Boolean searches are supported through the menu options or by using AND, OR, and NOT.

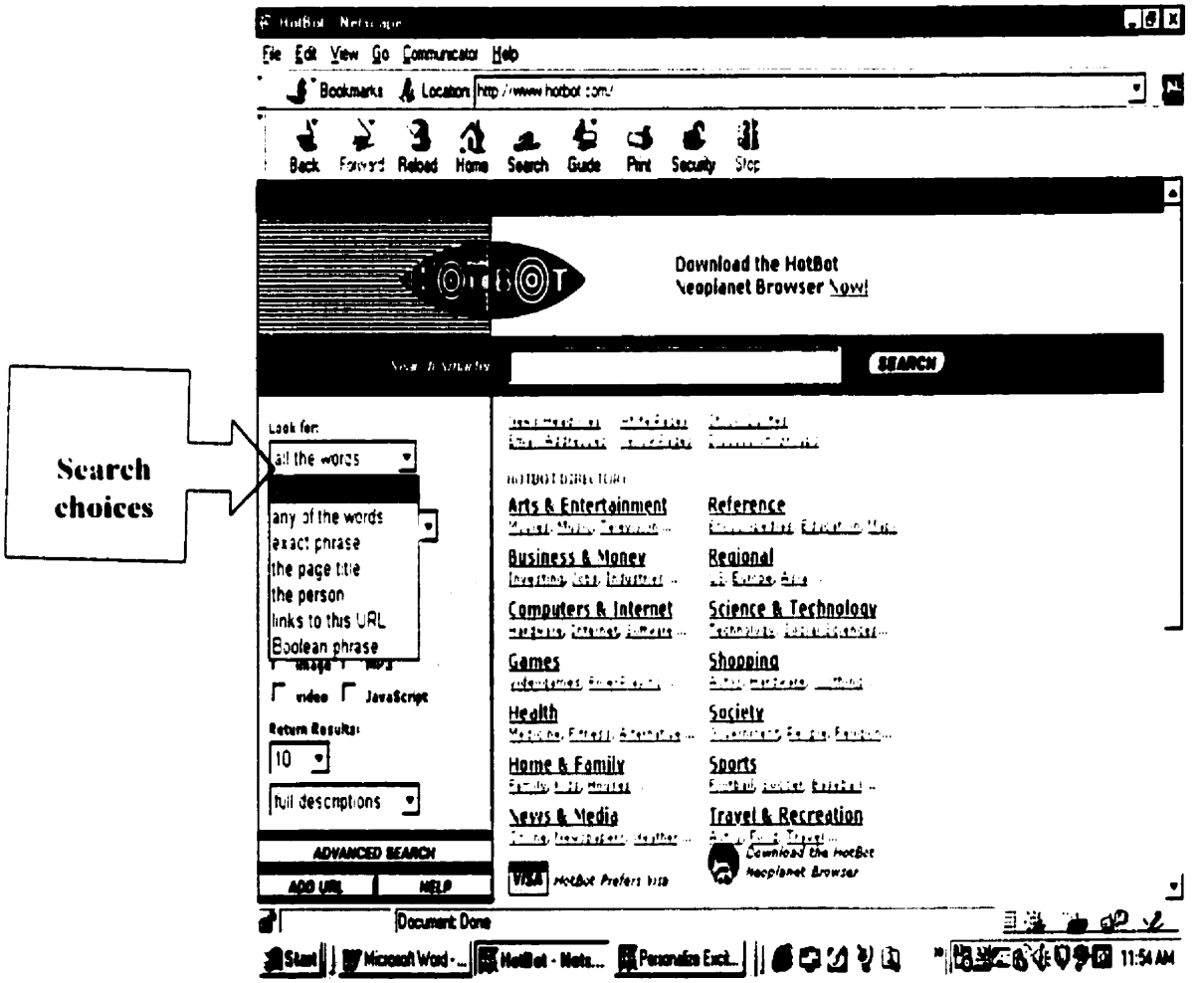

Figure 2.6 HotBot User interface

2.2.1.1.5 Infoseek Search Engine

There are 75 million sites in full-text. Infoseek has two user interfaces, simple search interface (Figure 2.7) and advanced search interface (Figure 2.8.) InfoSeek uses Boolean OR as the default. In the advanced search, Infoseek allows the use of AND and NOT. The user can search for phrases by enclosing them in quotation marks.

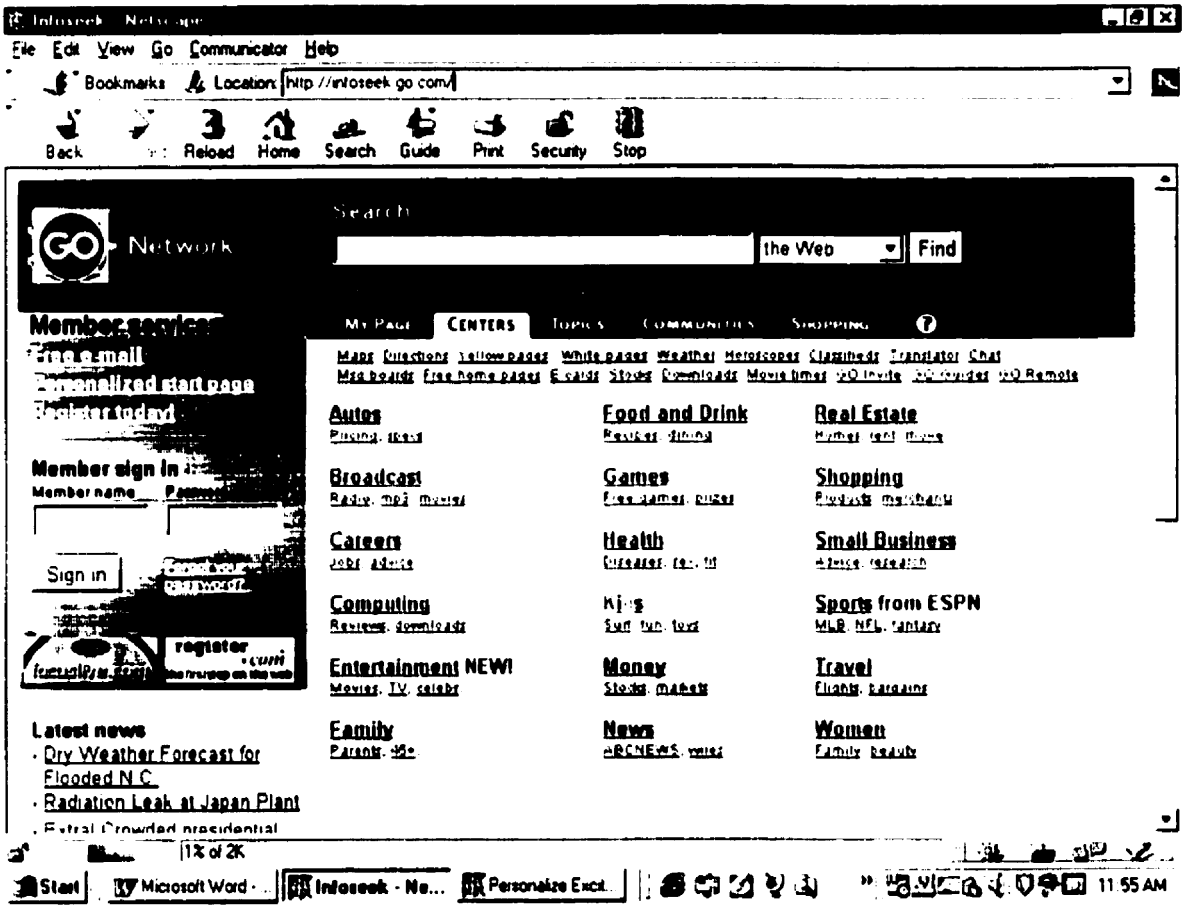

Figure 2.7 Infoseek Simple User interface

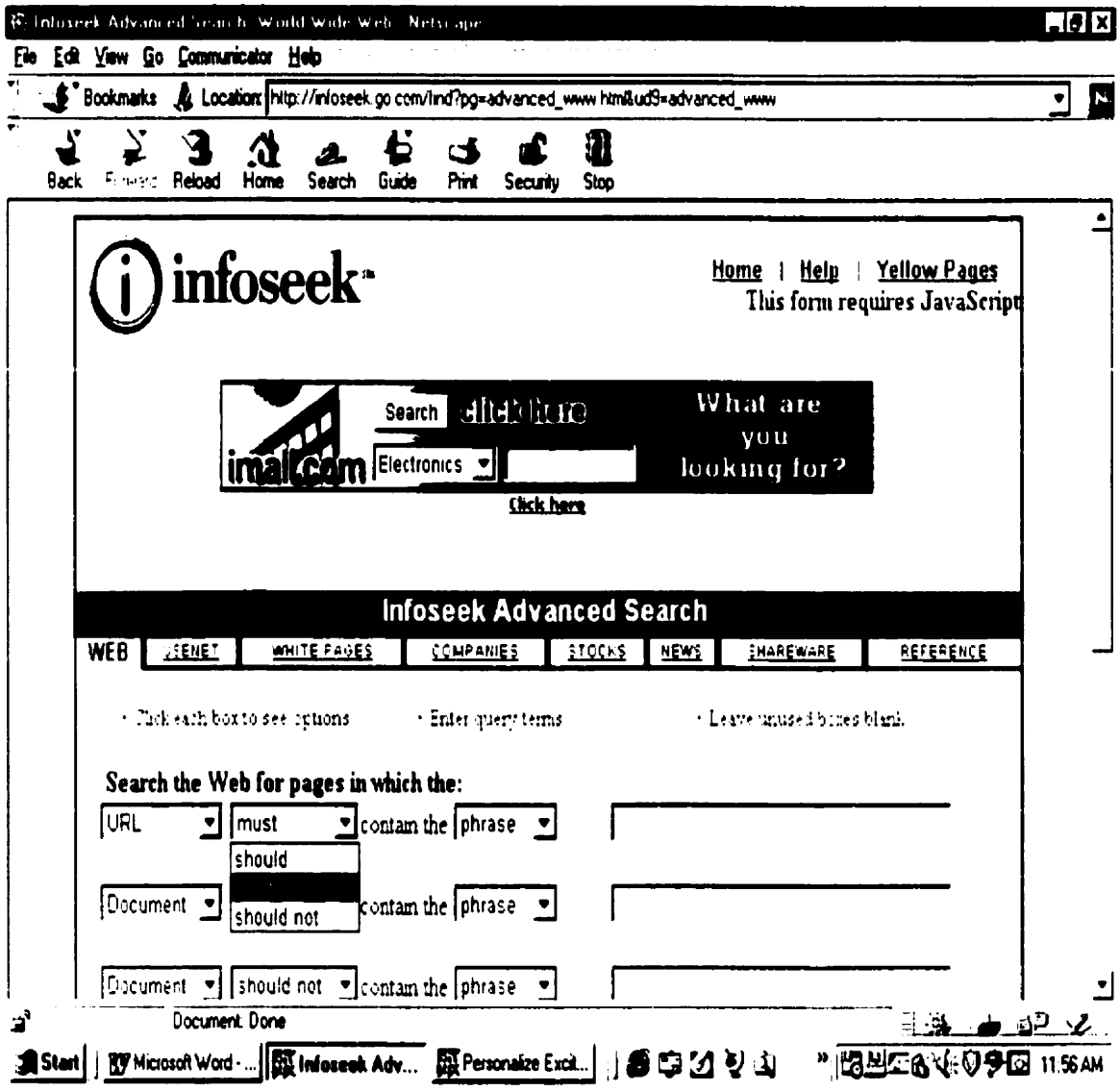

Figure 2.8 Infoseek Advanced User interface

2.2.1.2 Intranet search engines

An intranet search engine is a software program that can only search a website (see Figure 2.1). There are many intranet search engines, such as ht://dig and Phantom. The intranet search engines are not meant to replace the need for internet search engines like Yahoo, Lycos, Infoseek, Webcrawler, and AltaVista. Instead, the intranet search engines are meant to cover the search needs for a single company, campus, or even a particular sub section of a website.

2.2.1.2.1 Why we need intranet search engines

We need an intranet search engine because no internet search engine can access an intranet if it is behind a firewall (see section 2.3.1.2.2). Moreover, internet search engines are not searching the web directly (see section 3.3) but are using a database file, usually called an index file. The search engines search through the index file and provide the user with the addresses of search results. Most internet search engines cannot visit the website every day. Updating by a search engine [webp 98] takes time and that length of time depends on the speed of the search engine tools. For example, Yahoo can update its index database in four to eight weeks. Northern Light can improve its index database in three weeks. Lycos improves its index database in two to four weeks. If the content of a website changes often and a user is using an internet search engine to find specific information in a website, many of the links in the website will be dead links (see section 3.3.1).

2.2.1.2.2 Firewall

A firewall is a program which sits between a private network (an intranet) and public networks (the internet). The firewall can restrict access between the intranet and internet. Firewalls are generally used to protect an intranet from intruders, as well as to restrict access to the internet resources by local users of the intranet. For example, the computers in box A in figure 2.9 can communicate amongst themselves. But, they are isolated from the computers in B, C and D.

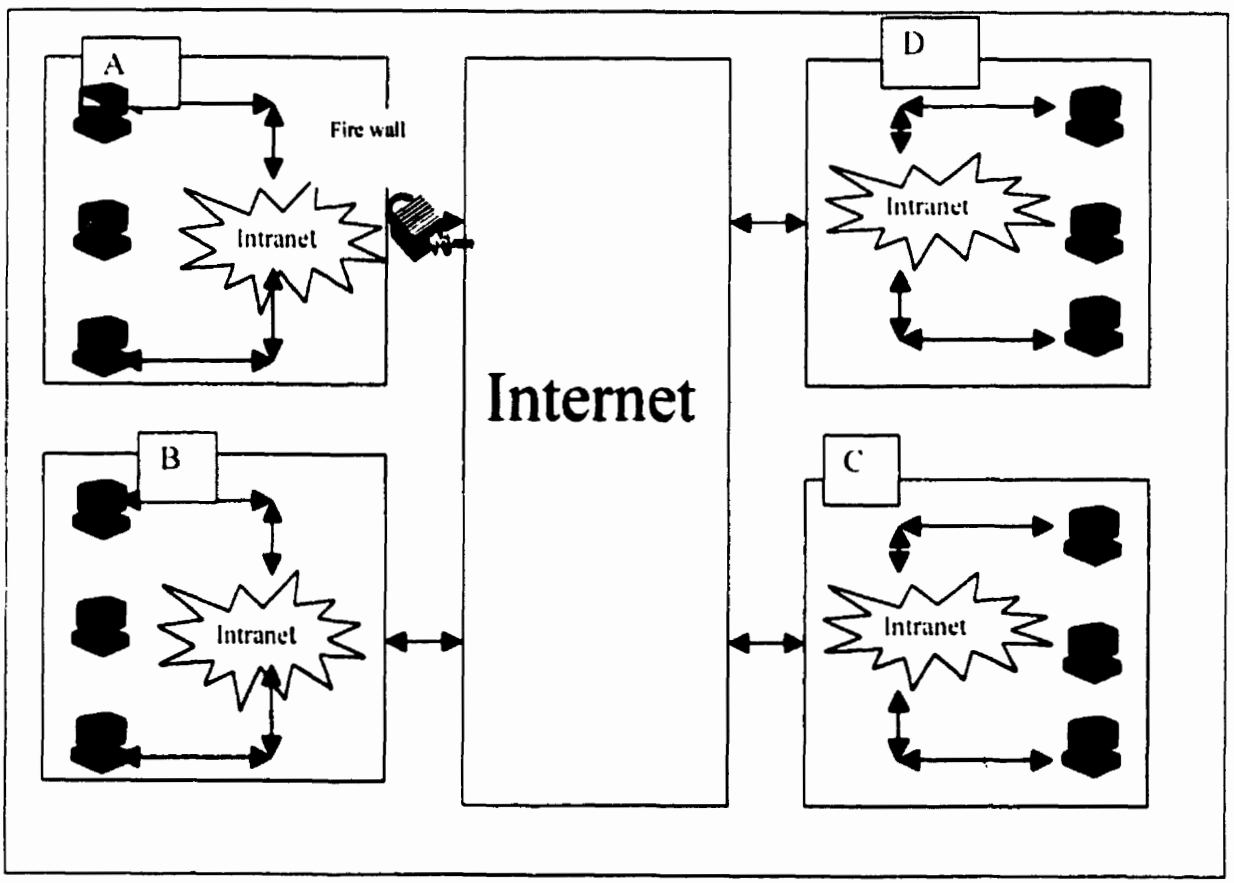

Figure 2.9 Firewall

2.2.1.2.3 ht://dig Search Engine

ht://dig [Sche 99] is a searching system for an intranet. The search engine uses keywords and was developed at San Diego State University as a way to search the on-campus website network.

2.2.1.2.4 Phantom Search Engine

Phantom [Phan 99] is another intranet search engine that can search a website. Phantom works in a manner similar to Altavista.

2.2.1.2.5 WebEnhancer Search Engine v 5.0

WebEnhancer [Deltix 99] is a personal search engine for a website. It works in a similar manner to Altavista or Yahoo. But, WebEnhancer has been developed to search a website, rather than the entire internet. WebEnhancer supports keyword search by using boolean logic for searching.

2.2.1.2.6 Searchlink Search Engine V 3.0

You can use Searchlink [Silk 97] in a webpage (Figure 2.10). A website visitor uses the search engine in a similar way to other internet search engines. Once a list is returned, the links can then be followed with a single click over the appropriate link name. The links themselves are stored and formatted in a text file which allows for easy

editing and fast access. Like many popular search engines, Searchlink uses "all words" which is AND logic, or "any word" which is OR logic searches.

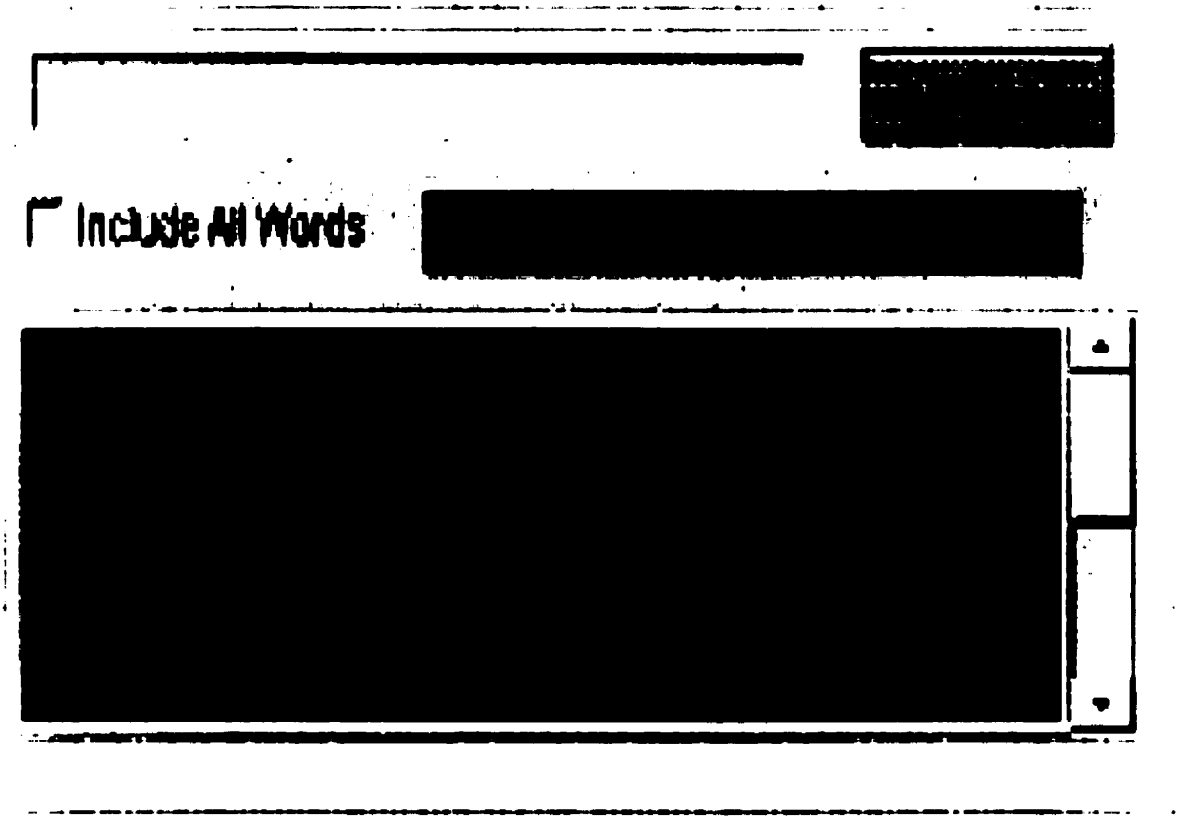

Figure 2.10 Searchlink search engine

2.2.1.2.7 Home page Search Engine v 1.5

The Home Page Search Applet [Rich 98] is a Java applet written to search the pages of a website (Figure 2.11). The applet starts searching at the index.html (or index.htm) file and then follows all local links (i.e all HTML pages) on a website. The applet uses both case sensitive and insensitive search strategies.

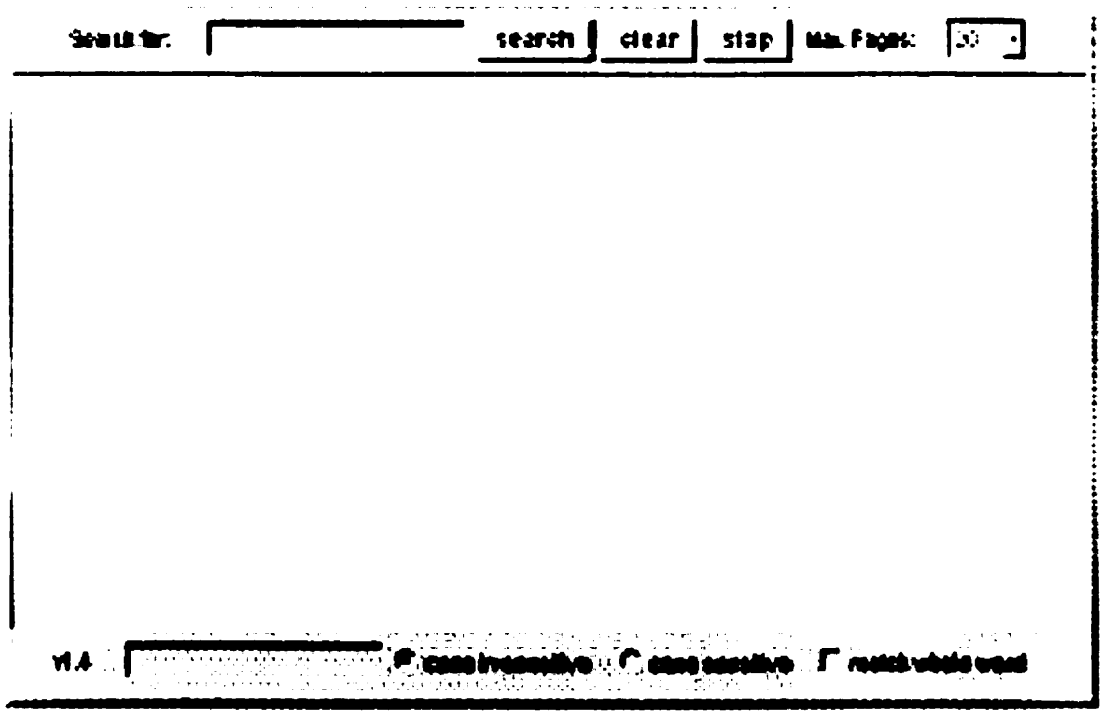

Figure 2.11 Home page search engine

2.2.1.2.8 SearchLite Search Engine v 2.0

SearchLite [leen 99] gives the user a choice of three search techiques: "the word" for single word search,"any of the words" which means OR logic, and "all of the words" which means AND Logic. SearchLite scans the site and presents the user with the pages that match the keywords.

Chapter 3 How Search Engines Work

Introduction

In this chapter we discuss different types of search engines, examine how search engines work, the hidden mechanics, and how a webpage can be found. Further, we discuss why there are so many dead links (sites are no longer in existence) returned from popular search engines.

3.1 Types of Search Engines

There are three different types of search engines: directory, index, and Meta search engines.

3.1.1 Directory Search Engines

Directory search engines catalogue all the webpages into categories. Queries go through the categories, not the individual pages. This type of search engine is best for searching a general subject. Yahoo is the most popular directory search engine.

3.1.2 Index Search Engines

Index search engines search webpages and index them according to the number of occurrences of certain words. When a request is typed to the search engine, the page that has the most words matching words in the query shows up first. The most popular index search engines are Altavista, Infoseek, Excite, Lycos, and Webcrawler.

3.1.3 Meta Search Engines

Meta search engines initially call two or more different search engines to perform a search. The meta search engine then collects, prunes, and ranks the results returned by the various search engines. There are many meta search engines such as Copernic 2000 and Metacrawler.

3.2 How Search Engines Work

There are two primary methods of searching that can be used by search engines.

3.2.1 Keyword searching

Most search engines do their query and retrieval using keywords [Barl 96]. We usually see basic and advanced options in these search engines. They try to narrow the results by using boolean logic and other techniques to provide the user with good results. One problem is that sometimes they cannot distinguish between words that are spelled the same way, but have different meaning, for example, "hard stone" or "hard exam". In either of these queries, the user may receive results related to hard rock music, which is completely irrelevant to user expectations.

Boolean logic and other techniques are discussed in the next section.

3.2.1.1 Boolean logic

Boolean logic refers to a system of logical thought developed by George Boole (1815-64) who was an English mathematician. People using boolean logic based search engines [Libr 99] must understand the meaning of each operator. The user also has to use the correct keywords in the search. If the keywords are too general, or have multiple meanings, users may receive too many results which are not related to their request [garb 99].

3.2.1.1.1 Boolean logic OR

When the user combines keywords using the boolean logic OR, the search engine is instructed to retrieve all the files that contain at least one of those keywords. Both words do not have to appear. If either word is present, the search engine presents the user with the file. For example, if the user wanted to find all the information the search engine had on either hard or rock, this search might be used: hard or rock.

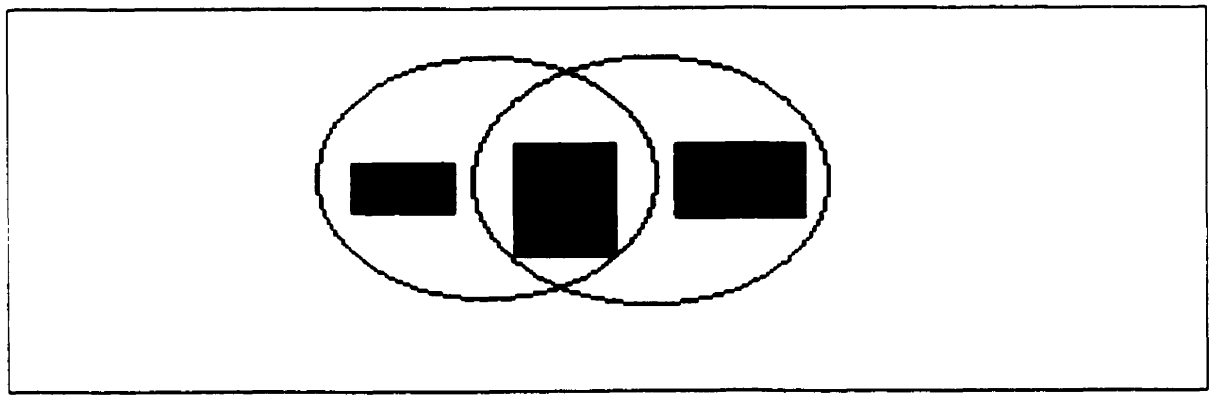

Figure 3.1 Boolean logic OR

As Figure 3.1 shows, the search engine goes through its database and locates every file with the word "hard" and every file with the word "rock." The search engine

presents the user with every file found that contains at least one of the words specified in the search.

3.2.1.1.2 Boolean logic AND

The boolean logic AND tells the search engine to search its database for every file that has both of the words somewhere in the same file. For example, if we want to find information on hard rock, we might search the appropriate search engine in this manner: hard and rock

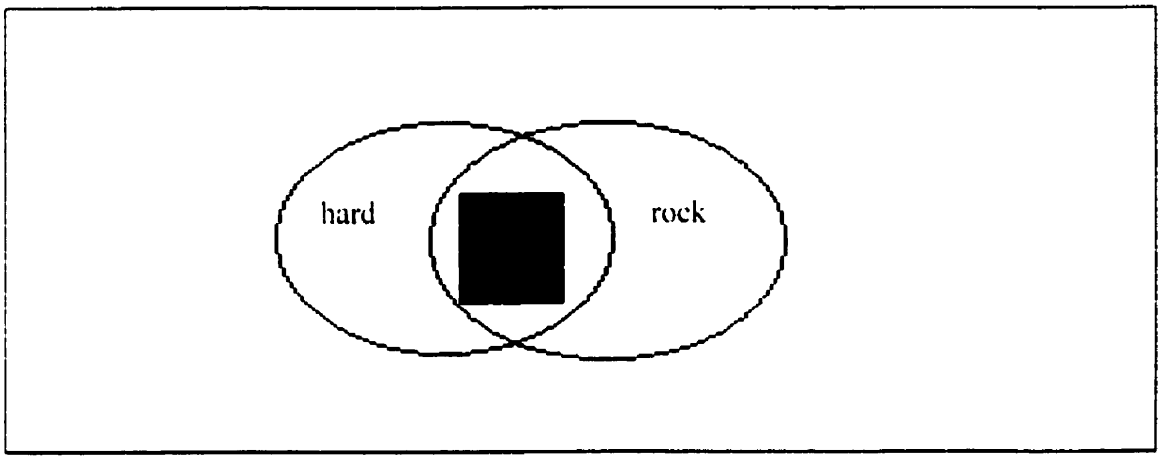

Figure 3.2 Boolean logic AND

The search engine goes through its database and retrieves every file found with the word "hard" and every file with the word "rock." Then, the search engine gives only

those files in which both words appear, as indicated by the shaded area where the circles intersect in Fig 3.2.

3.2.1.1.3 Boolean logic NOT

The final operator is NOT. Combining the search terms with this operator allows the user to strip a term out of the search. This tells the search engine to retrieve everything with the first word but nothing that mentions the second word. For example, if the user is looking for something with the word hard, any mention of the exam should be removed. We might try this search:

hard not exam

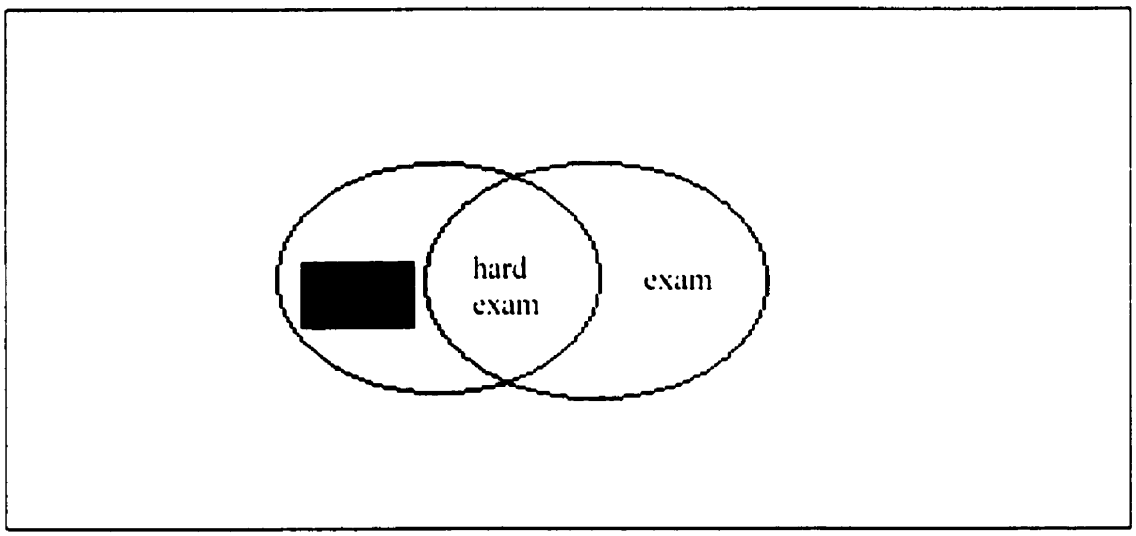

Figure 3.3 Boolean logic NOT

The search engine goes through its database and extracts every file with the word "hard" from that group of files. The search engine will display only those files in which

the word "exam" does not appear, as Fig 3.3 illustrates. We will not see any file that has both words.

3.2.1.2 Phrase Strategy

The phrase strategy tells the search engine to search its database for every file that has a group of words which appear side-by-side in the same file. For example, if we want to find information on hard rock music, we can pose the query:

"hard rock music"

The search engine goes through its database and retrieves every file it finds with the words hard and rock and music appearing side by side.

3.2.1.3 Case sensitivity strategy

We divide this into two sub-strategies

3.2.1.3.1 Case sensitive strategy

The case sensitive strategy tells the search engine to search its database for every file that has a group of words whose upper and lower case characters appear exactly as they do in the query.

3.2.1.3.2 Case insensitive strategy

The Case insensitive strategy tells the search engine to search its database for every file that has a group of words in the same file without paying any attention to whether the word is in upper or lower case. For example, if we want to find information on hard rock music, we might ask the search engine in this way :

Hard rOcK mUsIc

The search engine goes through its database and retrieves every file with the words hard rock music or Hard rOcK mUsIc or HARD ROCK MUSIC.

3.2.1.4 Wildcard Strategy

The wildcard strategy uses the asterisk (*) symbol which tells the search engine to return alternate spellings for a word at the point where the asterisk appears. For example, if we want to find the word hard, we might search the appropriate search engine in this way:

hard*

The search engine goes through its database and retrieves every file found with the word hard, harder, and hardest.

3.2.2 Natural language Strategy

The natural language strategy is the most powerful strategy because the search engine is told, in plain English, to search its database for every file that has the information for which the user is looking. For example, if we want to find information on hard rock music, we might ask the search engine in either of the following ways:

Where I can find any information about hard rock music.

Give me a list of hard rock music.

In both questions, the search engine goes through its database and retrieves every file it finds on the subject of hard rock music.

3.3 Spiders

When you use a search engine to search the web, you are not searching the web directly as mentioned in section 2.3.1.2.1. A search engine searches index files that contain some addresses and their contents. Some search engines like Altavista use spiders which are programs that can search the internet to update or add new web files, read their pages, and place their addresses and content-related information in an index file. Then, the search engine can search that index file. A spider is also known as a "crawler" or a "bot." These programs normally start with a historical list of links such as server lists which list the most popular sites. The links on these pages are then followed to find more links and add them to the index file. Most search engines have their own spider. For example, Altavista uses its own spider which is called Scooter [whatis 97].

3.3.1 Dead links

All information on the web is subject to change without notice. If a site is changed today, it will take sometime before the search engines notice the changes. Spiders cannot visit and update the web site every day. For example, the most powerful spider which is based in Altavista, can visit only three million websites per day. According to Search Engine Watch, the Web had 800 million pages in August 1999. For this reason, there are dead links that generate a browser window message stating that the requested document cannot be found.

A report by Greg R. Notess [Note99] shows in Figure 3.5a and figure 3.5b the percentage of dead links generated in the first one hundred hits from three separate searches. The dead links percentage columns include the 404 file not found error message, 401 access denied, and other errors that do not let the user connect to the result (see figure 3.5 a). Figure 3.5 b shows the percentage of dead links that resulted from the 404. Figure 3.6 shows the file not found error message. Figure 3.7 shows the file access denied error message 401 and Figure 3.8 shows the error message 403 forbidden errors. These examples exclude all the connection errors which could represent only temporarily dead links.

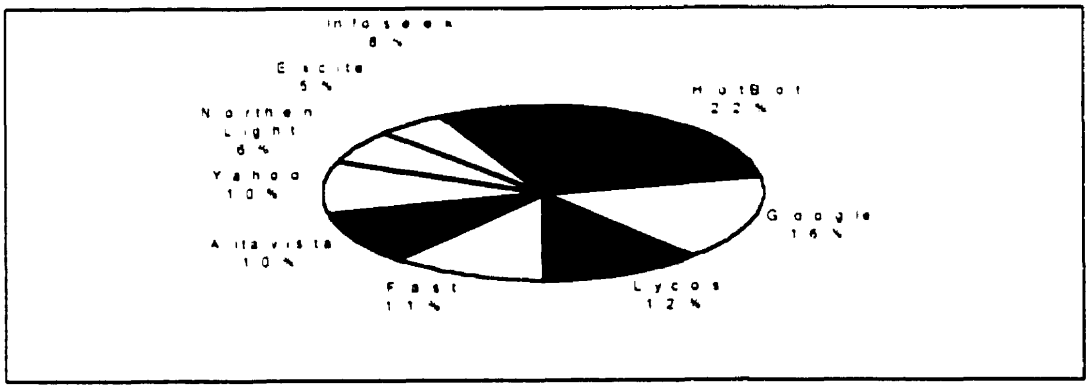

Figure 3.5 a Error messages

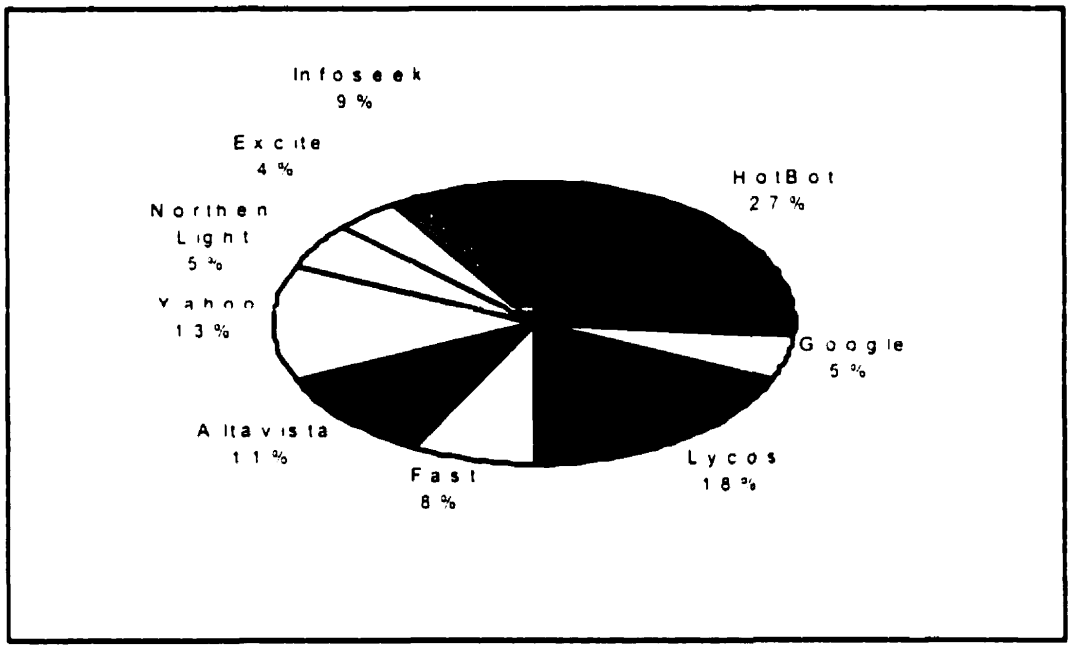

Figure 3.5 b 404 Error message

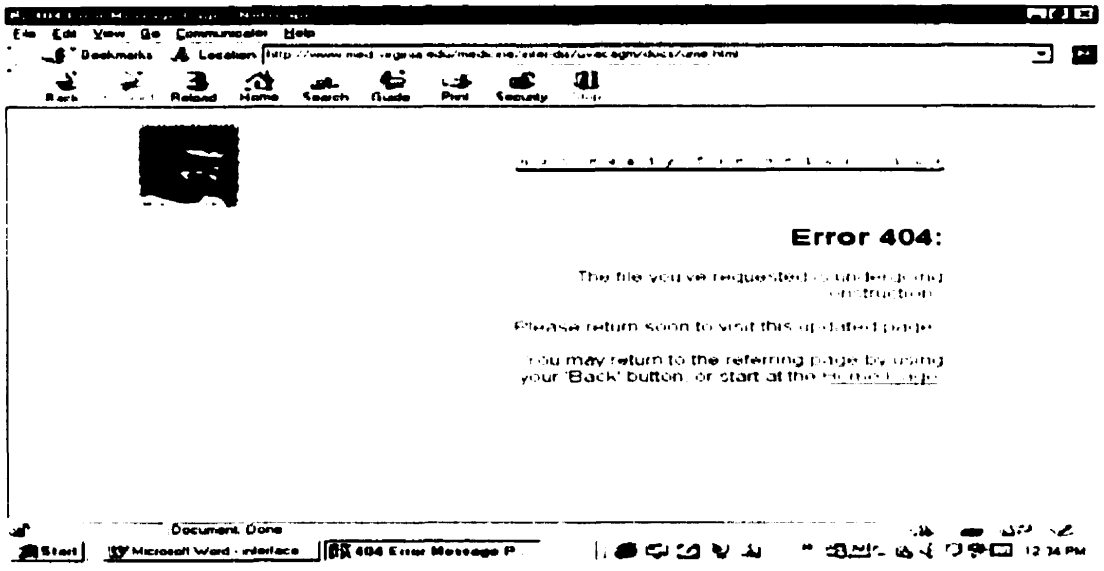

Figure 3.6 Error message 404

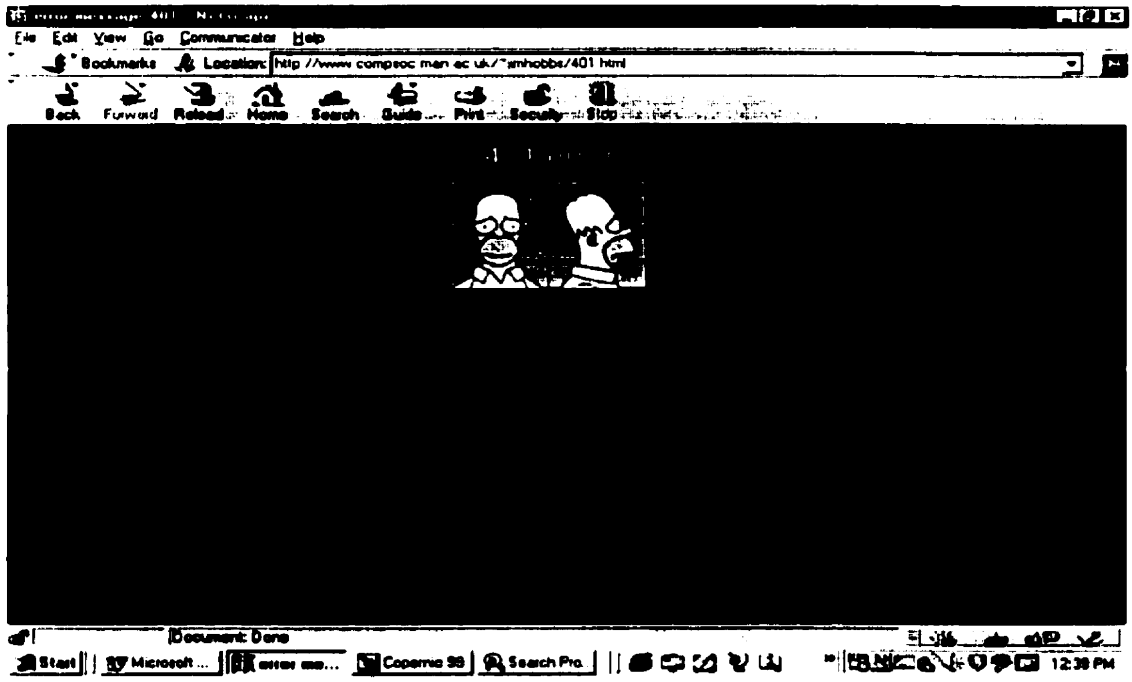

Figure 3.7 Error message 401

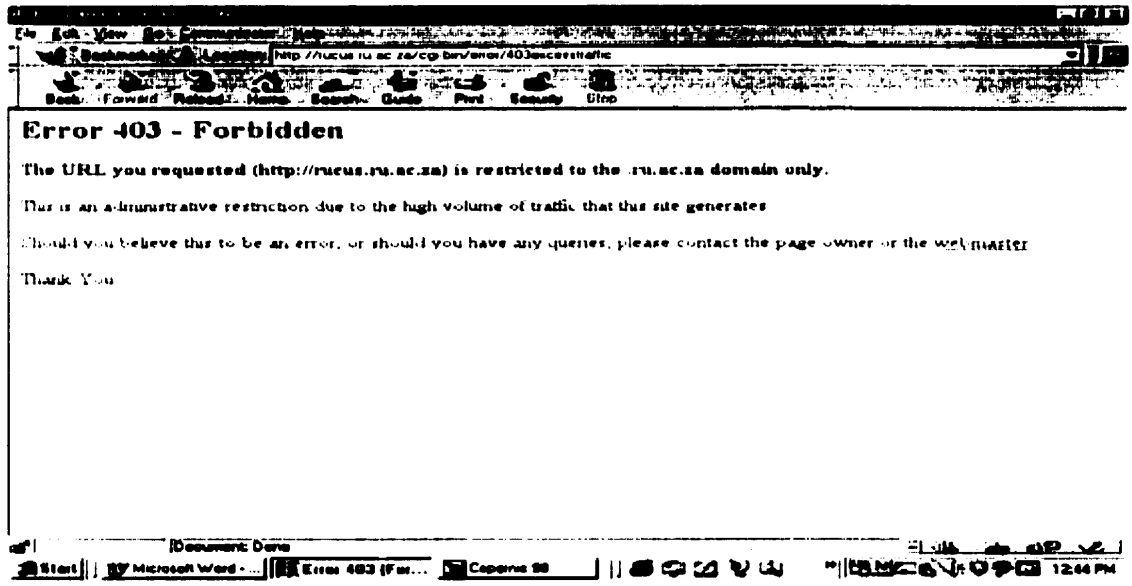

Figure 3.8 Error message 403

3.4 Conclusion

Most search engines use boolean logic strategies for searching. These strategies are sometimes complicated to use. If users are not familiar with boolean logic, it is sometimes difficult to find the information they want. Therefore, we decided to design an intranet search engine that would be easy to use with a single user interface and that would understand natural language (both typed and spoken).

Chapter 4 AIVDISE Overview

Introduction

In this chapter we present the strategies used to build AIVDISE and our system design. We begin with a brief overview of the goals we wanted to accomplish. Then, we detail the high level architecture of our system. Next, we give an overview of AIVDISE followed by a discussion of our strategies in a step-by-step presentation. We describe how we used those strategies in our system design. Finally, we describe how AIVDISE works using natural language.

4.1 Goals

Our goal was to build an intelligent voice driven intranet search engine (AIVDISE). Using Java, AIVDISE will accept natural language as input and provide the user with accurate results. In a single user interface, the user can use spoken English for his/her request. AIVDISE translates the verbal request into written language, searches its database, and returns results.

4.2 High level Architecture of AIVDISE

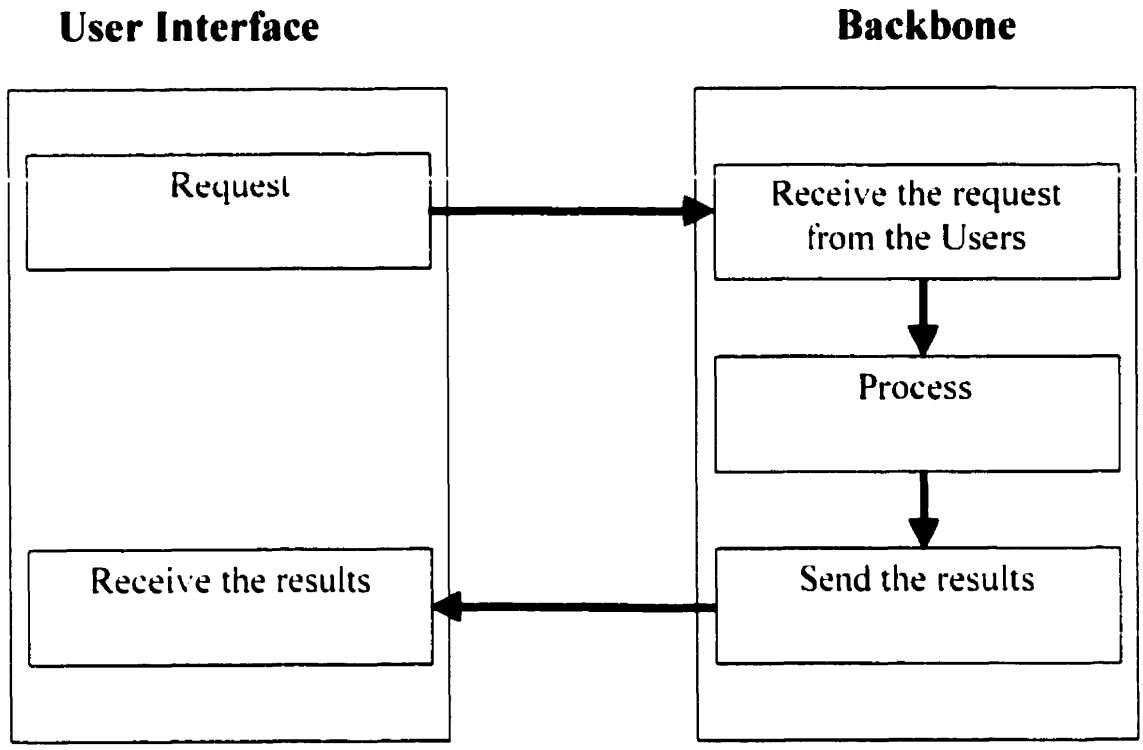

Figure 4.1 High Level Architecture of AIVDISE

The high level architecture of AIVDISE contains the user interface and backbone components. The former interacts with the user and is responsible for user requests (see Figure 4.1). Also, the user interface component holds received results so that the user can see the results of their request. The backbone components interact with the user interface and receives the request through the Request component. The Process component is a significant process and its strategies are described in section 4.3. The Send Results component receives the result from the Process component and passes it on to the

'Receive the results' component in the user interface. Finally, the user can interact with the user interface through the Receive the results component.

4.3 Overview of AIVDISE

A brief overview of the components of the AIVDISE system is given. The most important components, such as Stop word, concept-based searching, case sensitivity, and AND logic will be described in detail in the following section. In order to develop a user-friendly system, AIVDISE accepts queries in the form of English sentences. The question needs to be changed to lower case. All the stop words must be removed from the question. AIVDISE uses concept-based searching strategy in case there are some words which need to be changed. Then, the search will start by using AND logic strategy. An overview of AIVDISE is given in Figure 4.2.

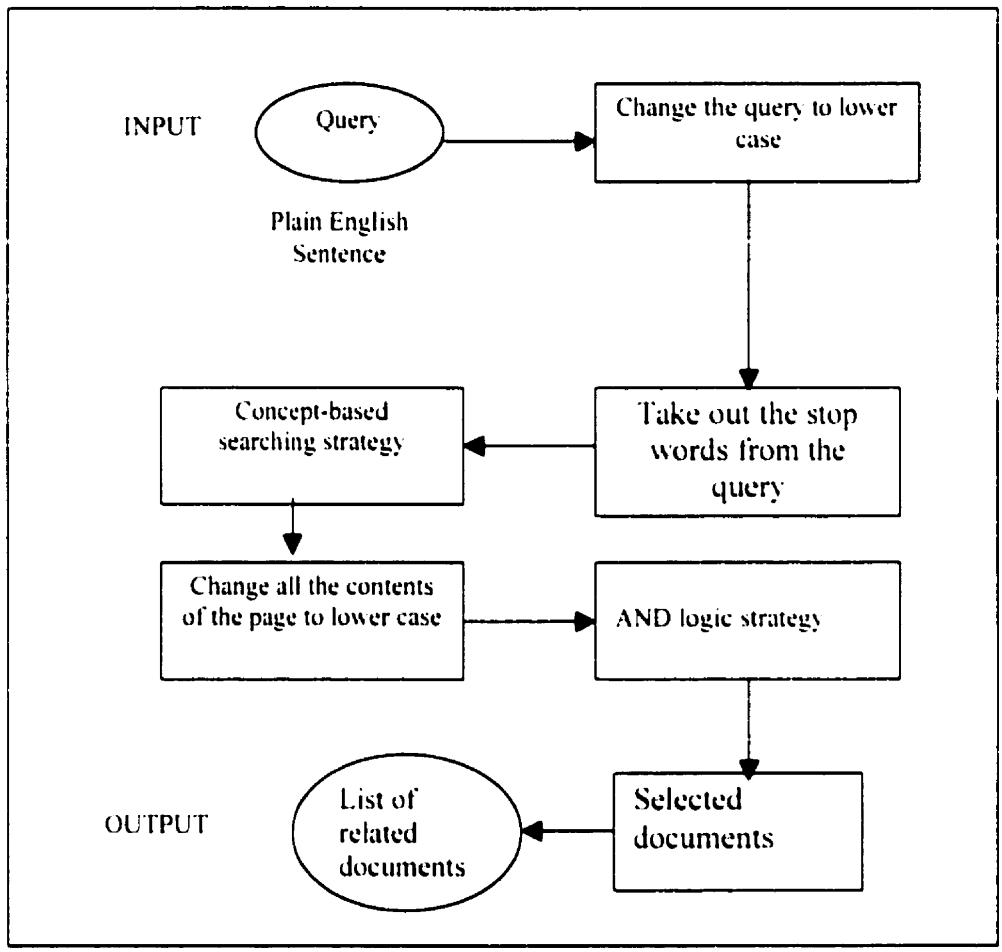

Figure 4.2 Overview of AIVDISE system

4.4 Strategies

In this section, we review the strategies and the methodologies used by AIVDISE.

4.4.1 Stop Words Strategy

In written text, some words are very common but they add no additional meaning to the actual content of the text. A considerable amount of processing time and working memory can be saved if the words that do not contribute to the actual content of the corpus are removed. The percentage that the corpus is being decreased is often 70-75 %. This is done by filtering the text using "stop words" [Rijs 79]. We constructed our own list of stop words which is shown in table 4.1.

For Example, consider this question:

Who is the director?

Since the most important word in this question is "director", we ignore the rest of the words. Thus, in this situation we are cutting processing time by 75% because if we want to search for each word, the search engine has to make four passes to find each of the above words. The search engine could potentially look at all matches of "who," "is," "the," and "director". The chances are that just looking for the last word is enough to find the relevant page. Therefore, to save time, AIVDISE will exclude searching for the other words.

Who	Is	Where	I
Show	What	The	Give
Can	Me	All	Find
List	A	Of	Are
Information	About	Were	Was
In	Dr.	By	And
Who's	Which	Offered	Doctor
Need	Some	Want	Search
Could	Please	Tell	You
Does			

Table 4.1 Stop Words

4.4.2 Concept-based searching Strategy

Now we are faced with another problem: how can AIVDISE understand the meaning of the question that will be asked. We used a concept-based searching strategy which allows the user to retrieve documents based on concepts. Concept-based searching attempts to determine what is meant, not just what is said. In the best circumstances, a concept-based search returns hits on documents that are "about" the subject/theme the user is exploring, even if the words in the document do not precisely match the words entered in the query. For example, a search for "teaching " could also automatically retrieve documents containing equivalent keywords. The exact equivalent keywords used depends on the particular context.

A list of equivalent keywords is shown in table 4.2.

Keyword	Equivalent keywords
Courses	Instructors or professor or professors or instructor
Fall	September
Teaching	Instructors or professor or professors or instructor or car 310 or car 416 or car 313
Teach	Instructors or professor or professors or instructor
Taught	Instructors or professor or professors or instructor
Teaches	Instructors or professor or professors or instructor or car 310 or car 416 or car 313
Director	Faculty at the Jodrey School of Computer Science
Trudel	Course Outline
Giles	Course Outline
Oliver	Course Outline
first	1st
Second	2nd
Third	3rd
Fourth	4th

Table 4.2 Equivalent Keywords

4.4.3 AND Logic Strategy

The next strategy AIVDISE uses is the AND logic strategy which narrows the search by retrieving only those references containing at least one term from each concept. The AND logic strategy is good for narrowing a search to the specific topic being researched. If we were doing a search for documents about "Computers and Society" by using the AND logic strategy, all the resulting documents would contain both of the terms "Computers" and "Society". Any documents containing one of the two terms, and not the other, would be excluded.

4.4.4 Case insensitive strategy

Next, AIVDISE uses the case insensitive strategy which allows AIVDISE to search its database for every file that has a group of words in the same file without consideration of case. For example, if we want to find information on "computers and society". The search engine goes through its database and retrieves every file it finds with the words "computers and society" or "Computers And Society." AIVDISE will do that by changing the user query and the file to be searched to lower case.

4.5 How AIVDISE Works

We illustrate how AIVDISE works through the use of screenshots. To start working with AIVDISE from a browser, open the file AIVDISE.hmtl and then press the open button (Figure 4.3) to get the main user interface. Then, the user can interact with the search engine. Alternatively, the "open" button can be embedded in another web page.

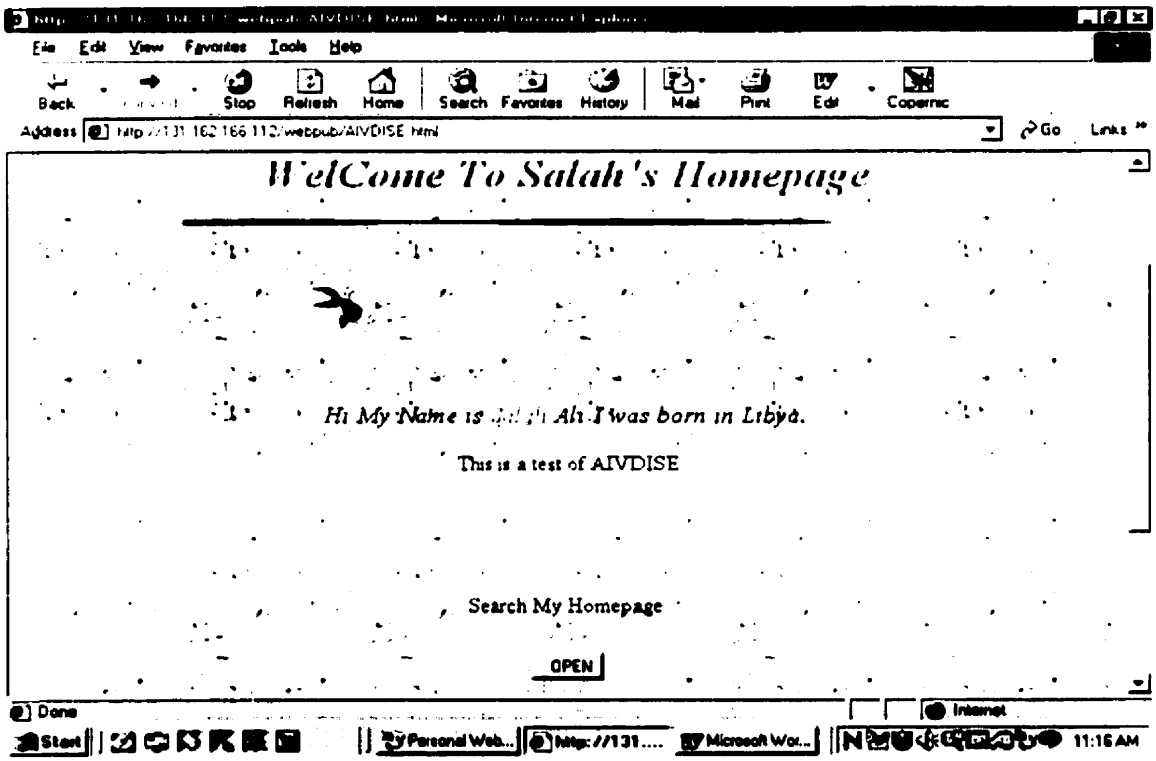

Figure 4.3 Opens AIVDISE

4.5.1.1 Main user interface

Once the user presses the open button, the system opens the main user interface. Figure 4.4 shows the main user interface. We divided the main user interface into three section menu bars: User requests, Results, and Description.

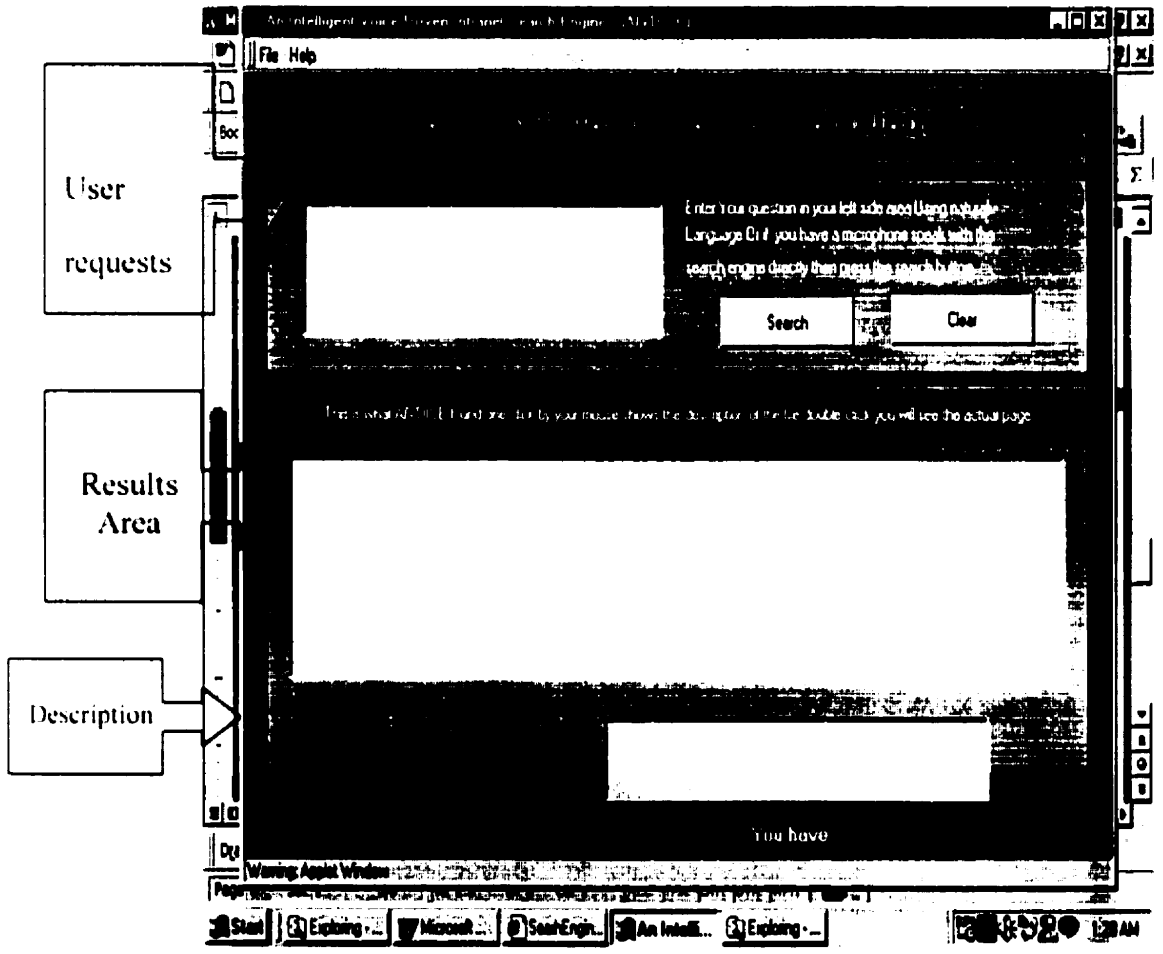

Figure 4.4 Main User Interface

4.5.1.1.1 User requests

The User request section is responsible for the requests from the user and sends the commands to the search engine.

4.5.1.1.1.1 File drop menu

File is a drop menu that lets the user exit from AIVDISE. The File drop menu contains one menu item which is exit (figure 4.5).

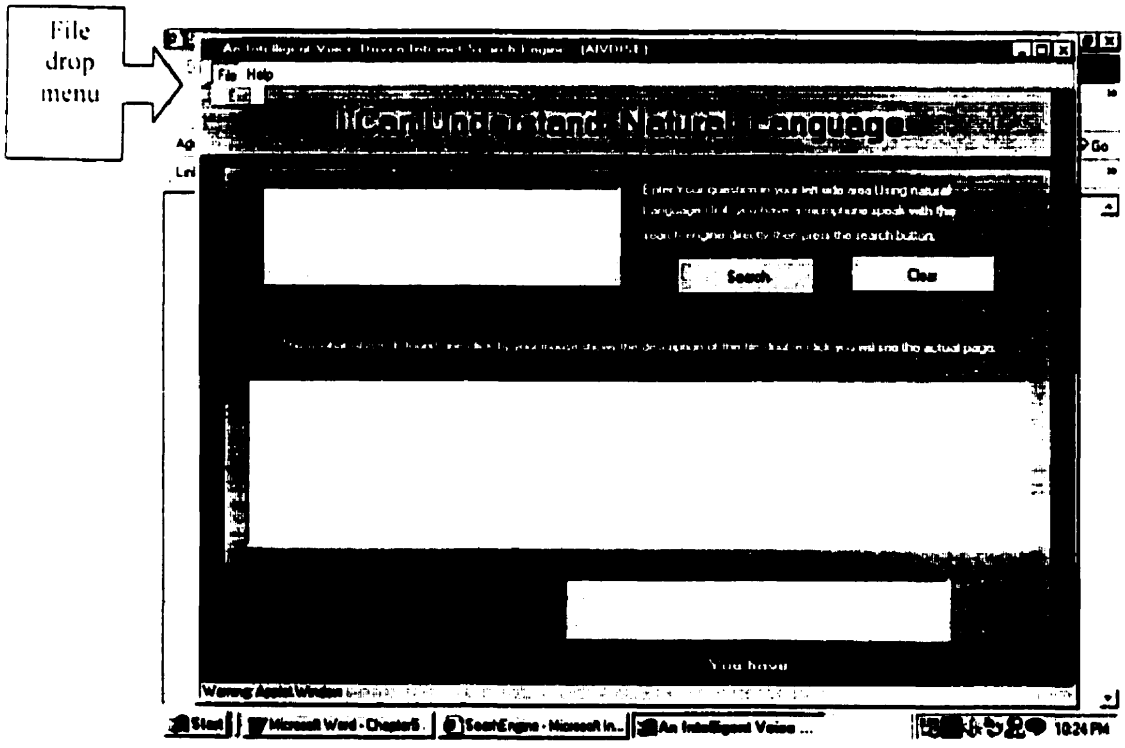

Figure 4.5 File Drop Menu

When the user finishes working with AIVDISE, the user can click on File. AIVDISE will open the drop menu for the user. Then the user can click on exit and AIVDISE will open an option window to exit or go back to the main user interface (figure 4.6).

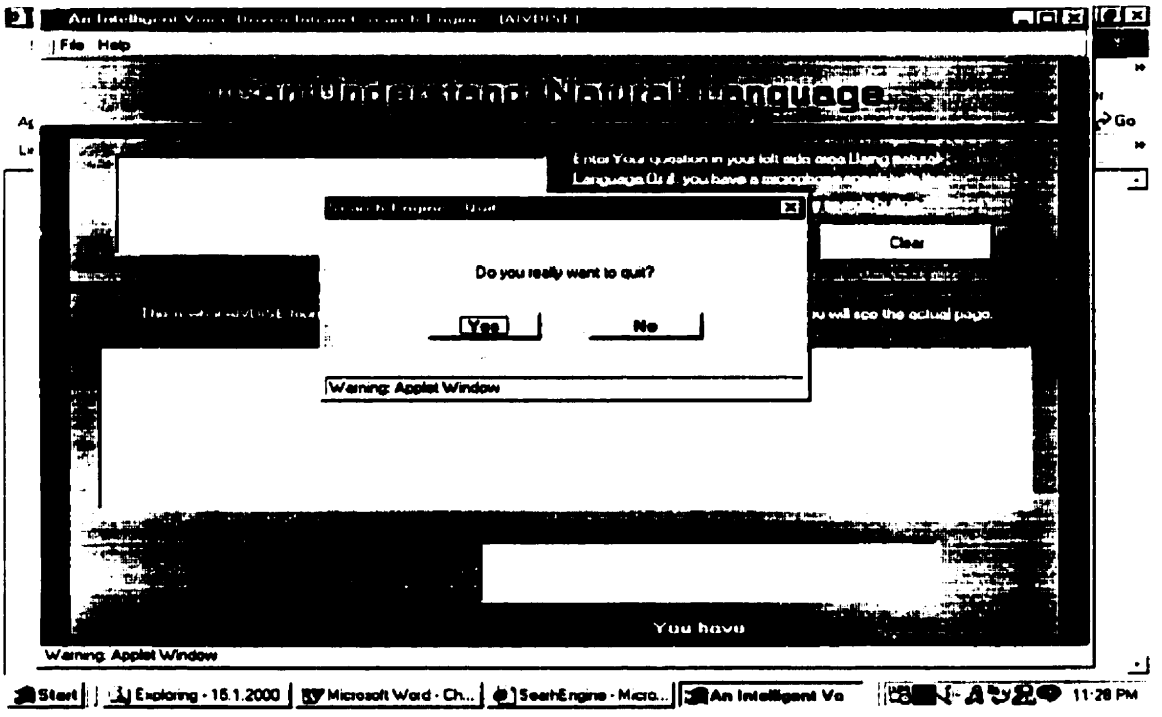

Figure 4.6 Exit Menu Item

4.5.1.1.1.2 Help drop menu

The Help drop menu contains the About and Instructions menu items (figure 4.7).

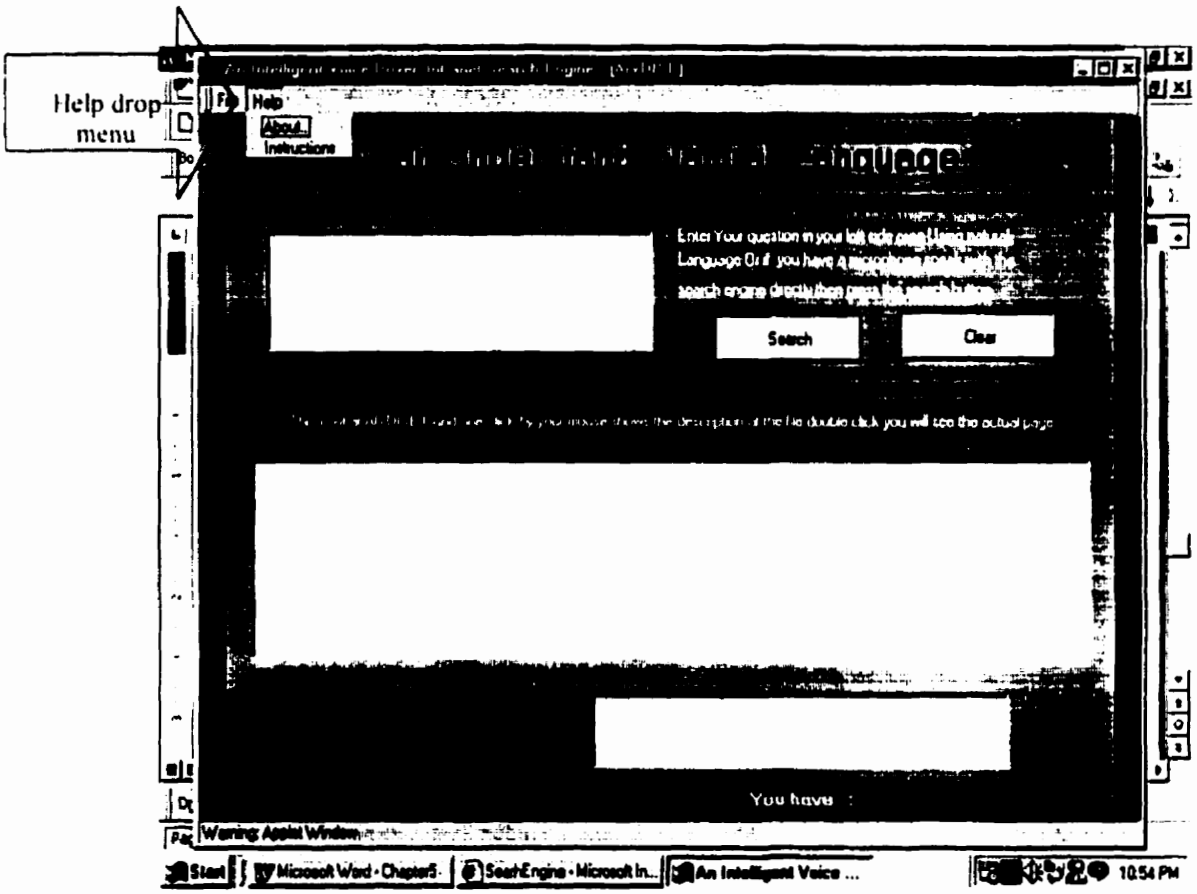

Figure 4.7 Help Drop Menu

4.5.1.1.1.3 Instructions Menu Item

Help is a drop menu that lets the user get help from AIVDISE (see Figure 4.7). Clicking on Instructions opens the window shown in Figure 4.8. It will guide the user in the use of AIVDISE.

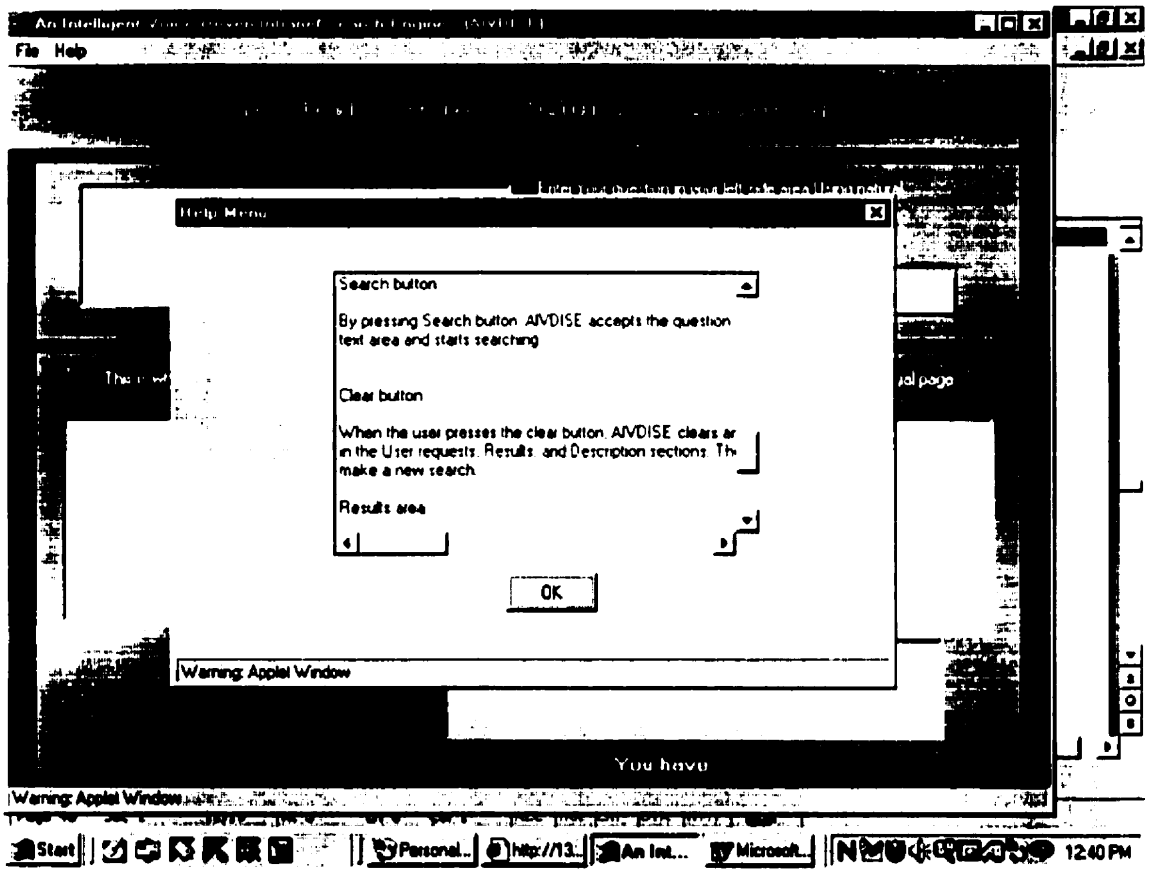

Figure 4.8 Instructions Menu Item

4.5.1.1.1.4 About Item Menu

This menu item shows information about search engine implementation. (figure 4.9)

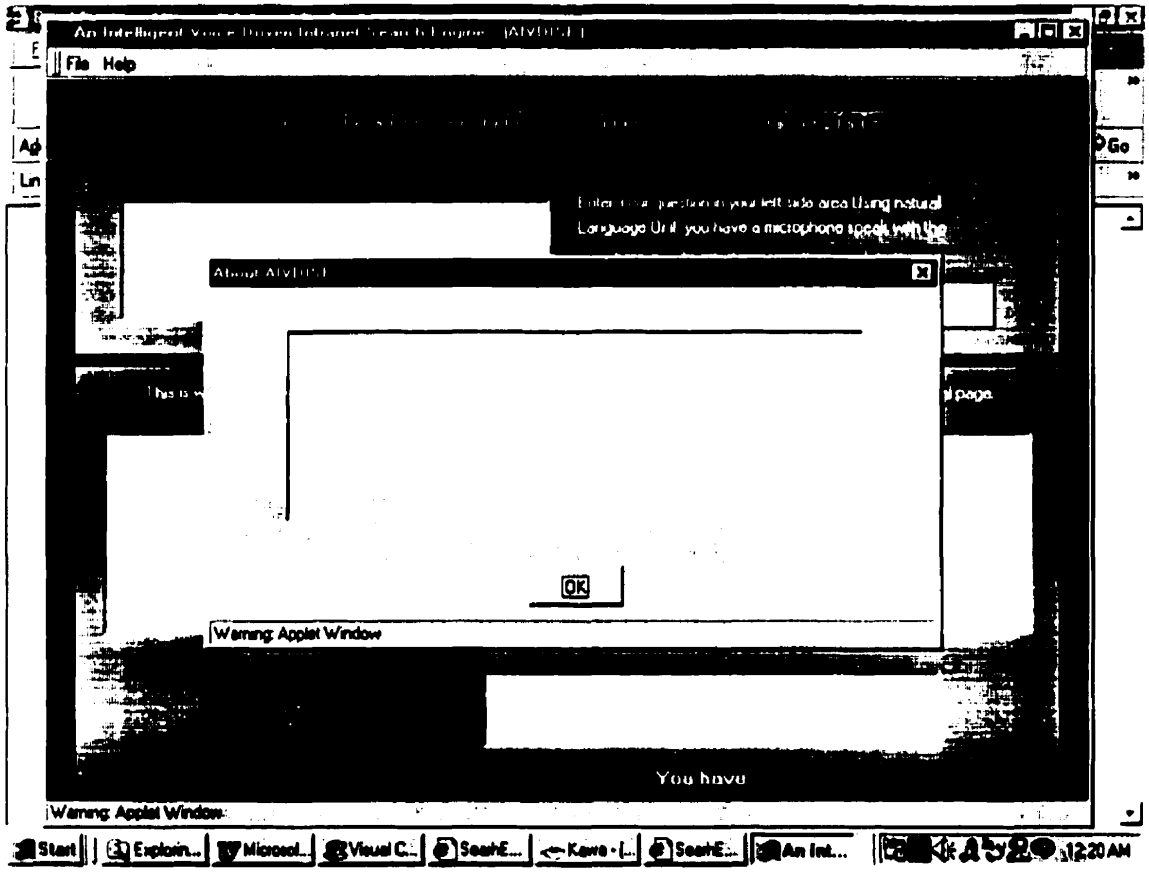

Figure 4.9 About Menu Item

4.5.1.1.1.5 Text area

The text area in figure 4.10 is responsible for accepting the request from the user in either typed or spoken English. For example, the text area in figure 4.11 shows the question "who is teaching comp 2903?".

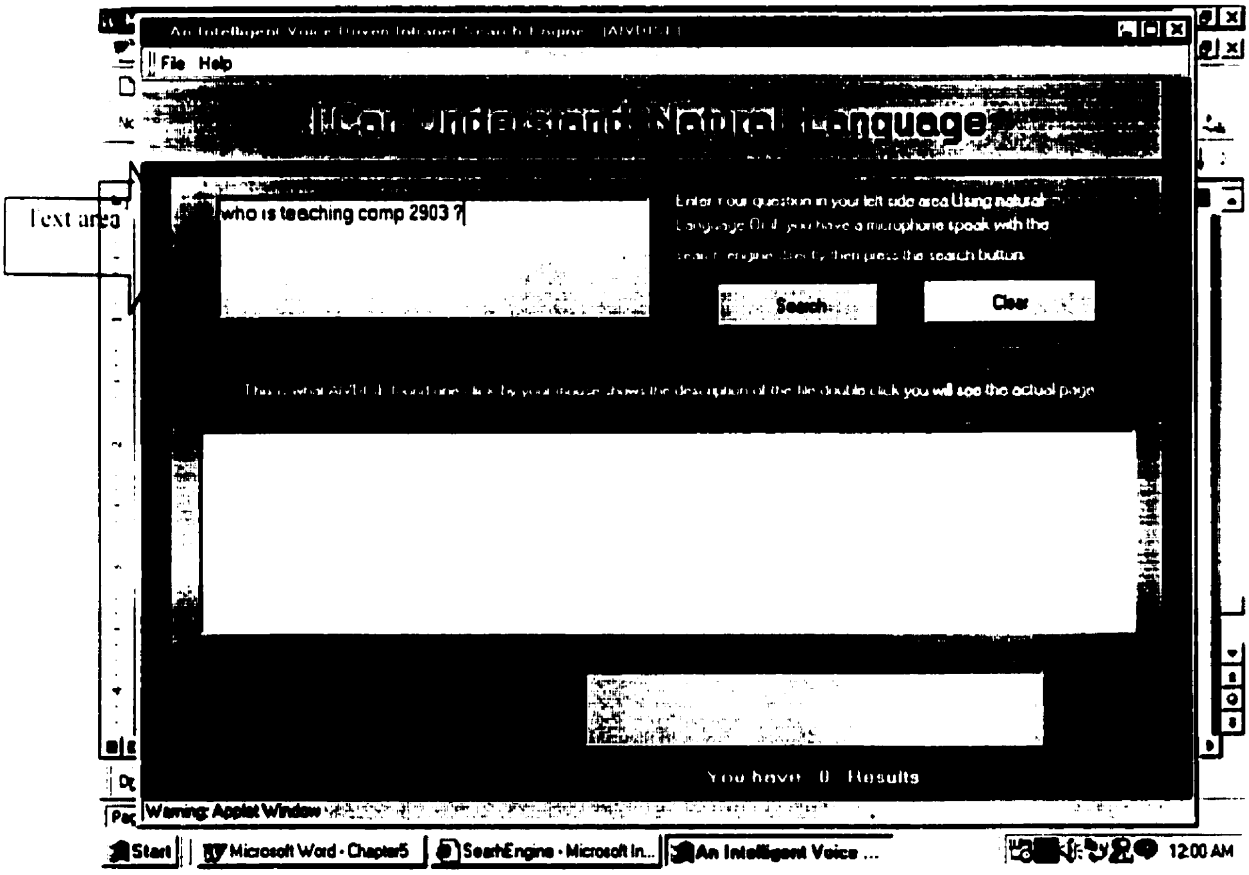

Figure 4.10 Text Area

4.5.1.1.1.6 Search button

By pressing the search button, AIVDISE accepts the question from the text area and starts searching. First, AIVDISE changes the question to lower case. Second, all the stop words must be removed. In the example in Figure 4.11, the question is "Who is teaching comp 2903?" and the stop words are "Who" and "is". Third, the word "teaching" has to change because we want the name of the professor who is teaching comp 2903. We replace the keyword "teaching" with "professor or professors or instructor or instructors". Fourth, AIVDISE changes each file to be searched to lower case. We then start searching for the words " professor or professors or instructor or instructors" and "comp" and "2903".

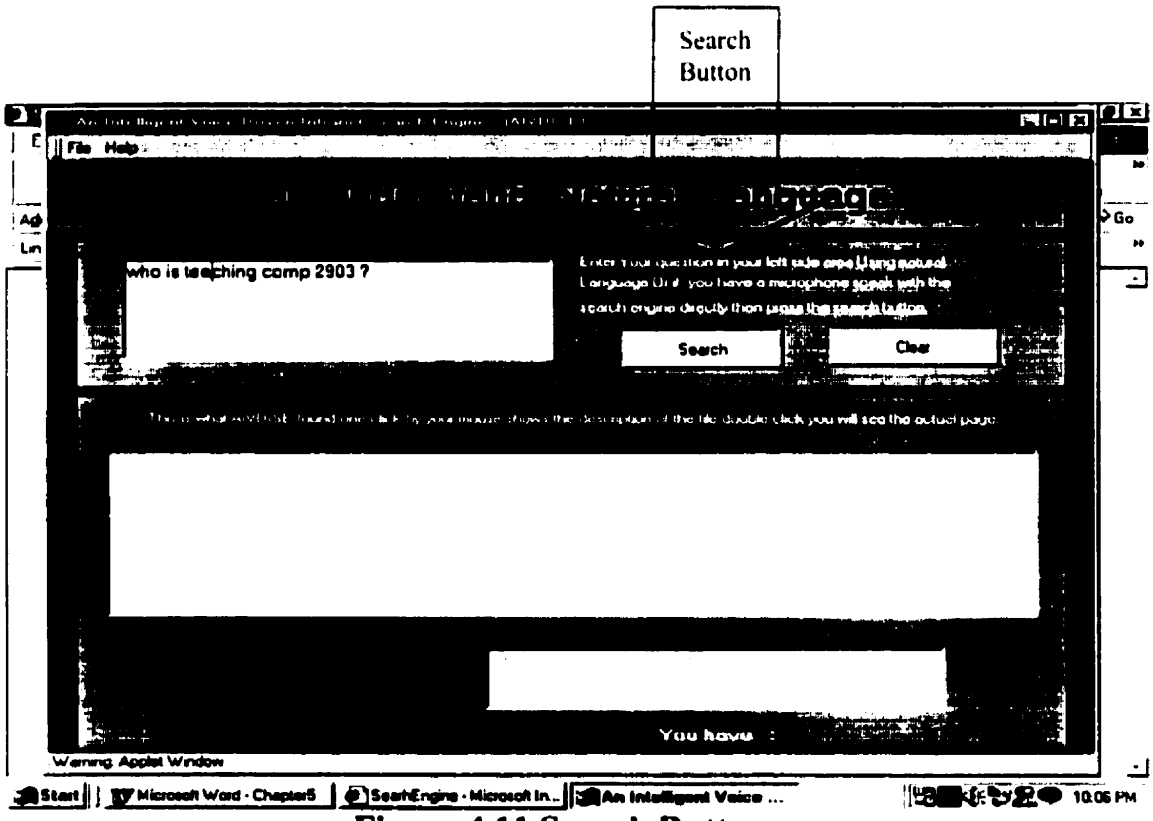

Figure 4.11 Search Button

4.5.1.1.1.7 Clear button

When the user presses the clear button, AIVDISE clears any information in the User requests, Results, and Description sections. Then the user can initiate a new search. Figure 4.12 shows the button and all the sections cleared.

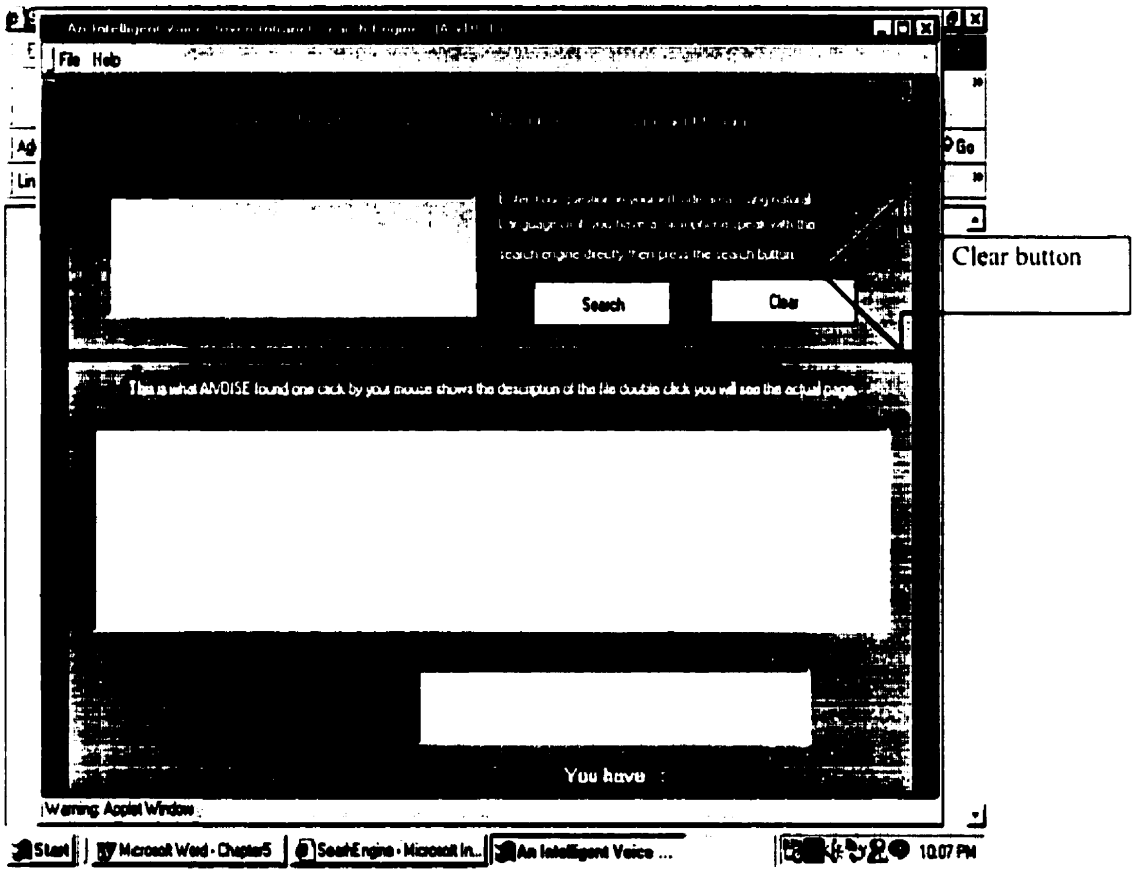

Figure 4.12 Clear Button

4.5.1.1.2 Results area

In this section, the user can see the results of the search. Only the addresses of the pages found by AIVDISE will be shown. Figure 4.13 shows the results of the question "who is teaching comp 2903?". Although we do not see the actual page, the user can access the page by double clicking on the address. Figure 4.14 shows the actual page.

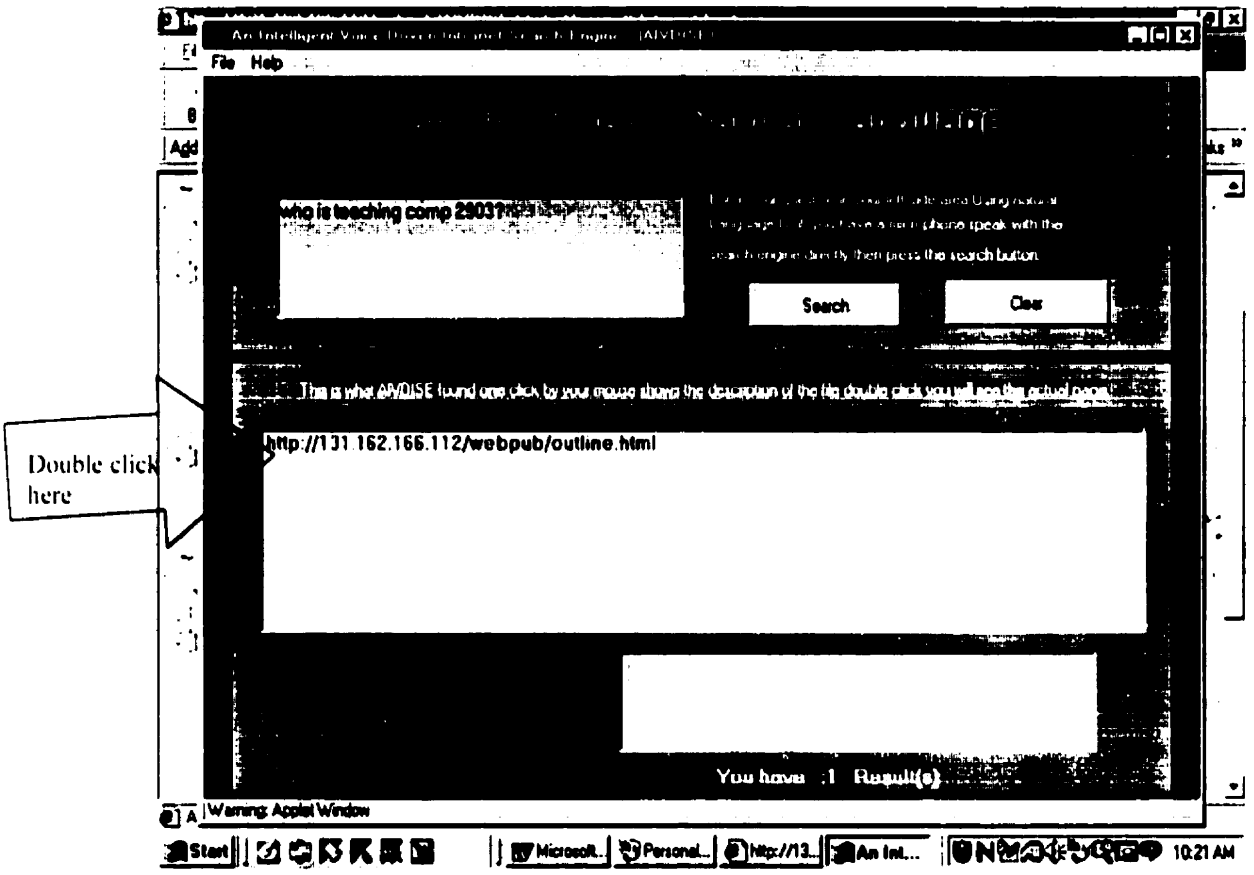

Figure 4.13 Results Area

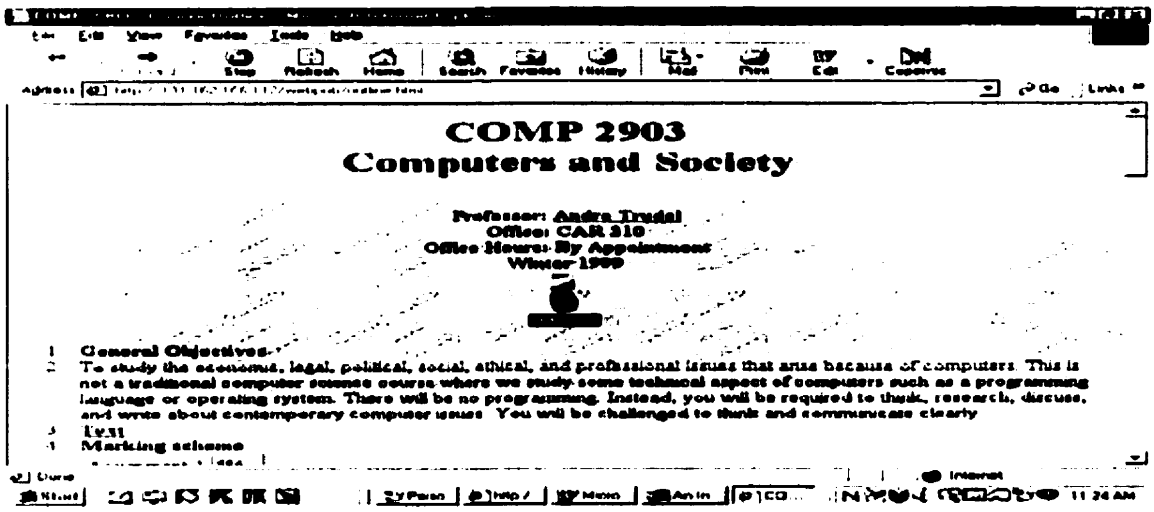

Figure 4.14 Actual Page

4.5.1.1.3 Description

This section endeavors to assist the user by showing a

page found by AIVDISE. The user can click on the address in the result area and AIVDISE will show the description in the description area (figure 4.15).

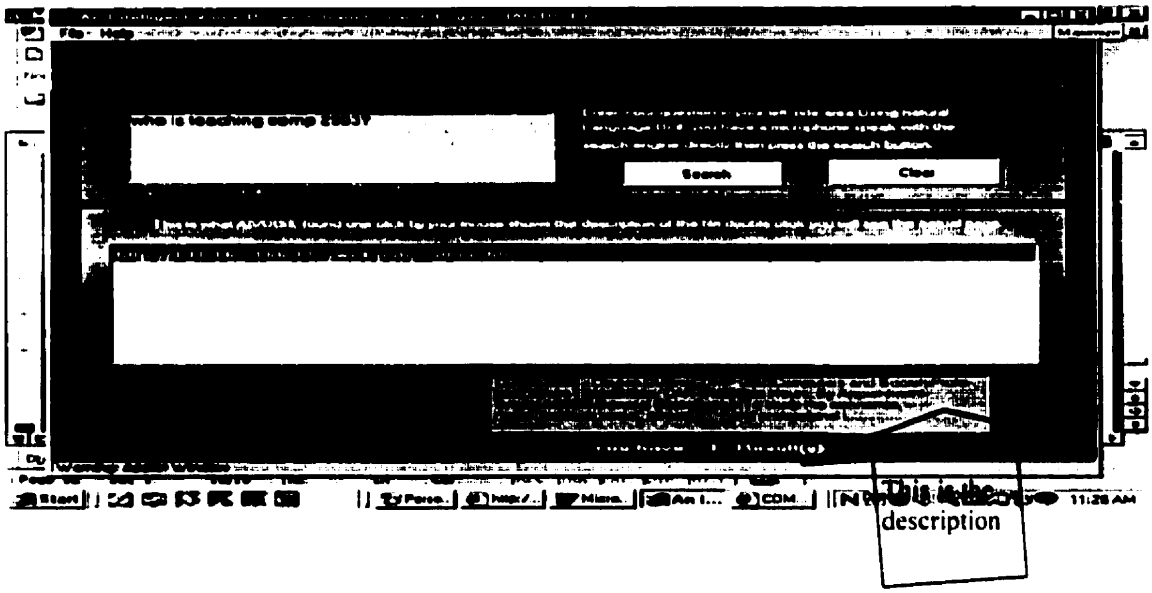

Figure 4.15 Description

Chapter 5 AIVDISE Implementation.

Introduction

In this chapter we describe the system implementation. We discuss the most important classes used by AIVDISE through code fragments.

5.1 SearchPanel Class

This class is responsible for all the interactions between the user and the system. The SearchPanel Class implements a series of methods to fulfill its functionalities.

5.1.1 getSearchText Method

This method is responsible for reading the request from the user e.g., "which courses were offered in fall 1999".

```
public String getSearchText() {
    return req_area.getText();
}
```

When actionPerformed() is called, the "clear" or "search" button was pressed. If the clear button is pressed, all the text area will be cleared.

```
public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == clearButton) {
Clear every thing in the main user interface, results list area, description area
    }
    if (e1.getSource() == searchButton) {
' If the button presses but no request entered
' AIVDISE will not make any search through this condition

  if (getSearchText().length() == 0) {
        return ;
      }
```

If the user presses the search button, we send the request and home page address to the Search class.

```
search = new Search(mainGUI, mainGUI.searchEngine.home_address,
getSearchText.toLowerCase());;
```

5.2 Result Panel Class

This class is responsible for holding all the results retrieved by the search class. Also, it shows the description of each page found.

5.2.1 clearList method

This method is invoked when a new search is initiated or when the 'Clear' button is clicked. It clears all the addresses from the result list.

```
public void clearList()
{
clear all the elements from the result list
}
```

5.2.2 addResult method

This method is responsible for adding the results to the result list. This method is invoked when a search is successful and it increases the number of results by one.

```
public void addResult(String match, String url)
{
   numberOfresults ++;   // number of results increased by one
   resultsList.addItem(match); // add  each results found to the results list
}
```

5.2.3 itemStateChanged method

This method is responsible for the description area. For example, it shows the description of each file when it is highlighted.

```
public void itemStateChanged(ItemEvent e){
String[] showMe = {"This is will be in ","You will find it in",".Are you looking for","Or
you are looking for" ,"You may looking for"};
  int index = resultsList.getSelectedIndex();
     String str = (String)description.get(new Integer(index));
    if (show1 > 4) show1 =0;
  description_Label.setText(showMe[show1] +" ");
  int len = str.length();
if(len > 400) {
   String line1 = str.substring(0,61);
   String line2 = str.substring(61,121);
      String line3 = str.substring(121,181 );
      String line4 = str.substring(181,241 );
  textArea1.setText(" "+ line1 + "-" +"\n");
  textArea1.append(" "+line2+ "-" +"\n");
    textArea1.append(" "+line3+ "-" +"\n");
    textArea1.append("  "+line4 +".....");
show1=show1+1;
  }
```

```
  else {
      String line5 = str.substring(0,61);
      String line6 = str.substring(61,121);
        textArea1.setText(" "+ line5 + "-" +"\n");
       textArea1.append(" "+line6 +".....");
show1=show1+1;}

}
```

5.2.4 actionPerformed method

The actionPerformed() method is called when the user double clicks on an address and the actual page is shown.

```
  public void actionPerformed(ActionEvent e1)
  {
        int listIndex = resultsList.getSelectedIndex();
          URL goal;
         try {
             goal = new URL((String) urls.elementAt(listIndex));
             frame1.searchEngine.getAppletContext().showDocument(goal, "Results");
          } catch (MalformedURLException badurl) {
          }
     return;
  }
```

5.3 Search Class

Search Class is the main class and it does the entire search.

5.3.1 GetDescription Method

This method is responsible for removing all the tags from each html file.

```
public String getDescription (String contents)
  {
' This method will delete all the tags in each file and return the file as text
    }
```

5.3.2 found method

This is the most important method since it searches the website for pages that satisfy the user's query.

5.3.2.1 Stop Words Strategy

This action is responsible for removing all the stop words from the user's query.

```
String[] wordIgnoreMe = {"who", "is" ,"where", "i", "show", "what", "the", "give" ,
"can",  "me", "all", "find" ,"list", "a", "of", "are" ,"information", "about", "were", "was",
"in", "dr.", "by",  "who's", "which" ,"offered", "doctor" ,"need" "some" ,  "want"
,"search" ,"could", "please", "tell" , "you", "does", "and"} ;
    Enumeration enumSubstrings = subStrs.elements();
```

```
        while(enumSubstrings.hasMoreElements()) {
    String subString = (String) enumSubstrings.nextElement();
      for( int i=0; i< 37; i++){
        if (subString.equals(wordIgnoreMe[i])){
          subString =  (String) enumSubstrings.nextElement();
            i=0;
          }
  }
```

5.3.2.2 Concept-based searching Strategy

This section of the code replaces words in the user's query. For example, the following code replaces every occurrence of "teaching", "teach", " taught", and "teaches" with "professor" or " professors" or "instructor" or "instructors ":

```
if((subString.equals("teaching"))
      ||(subString.equals("teach"))
      ||(subString.equals("tought"))
      ||(subString.equals("teaches")) ){
            subString = "professor";
            if (content.indexOf(subString) == -1) {
            subString="instructor";
             }
            if (content.indexOf(subString) == -1) {
            subString="professors";
```

```
            }
            if (content.indexOf(subString) == -1) {
            subString="instructors";
    }
```

In this instance, there is a high probability the user is interested in finding out who is teaching a particular course. Thus, "professor " is a better keyword to use.

5.4 How dragon naturally speaking interacts with AIVDISE

AIVDISE uses Dragon naturally speaking preferred (DNS) version 3.01 as a front end. DNS is commercial software that can work with any Windows application. After the installation of DNS (see Appendix A), users can use spoken English, simply by using the headphones.

5.5 Example

In this example, we describe all the steps performed by AIVDISE to answer the spoken query "which courses were offered in fall 1999". Instead of typing the request in the text area, the user can use spoken English by using dragon naturally speaking. AIVDISE accepts the request from the text area. Through the getSearchText method (section 5.1.1), AIVDISE starts searching after the search button is pressed. This action happens through the actionPerformed method in the class searchpanel. Then, AIVDISE changes the question to lower case by calling the method toLowercase from the Java library (figure 4.2 change the query to lower case). AIVDISE sends the query to search

class. In this stage, all the stop words must be removed by the method found in the search class by using the wordIgnoreMe array (figure 4.2). This will take out the stop words from the query. AIVDISE test each keyword from the request against each keyword in the array wordIgnoreMe and remove all keywords found from the request. The request is "Which courses were offered in fall 1999" and the stop words are "which", "were", "offered" and "in". Next, the word "courses" has to change because we want the name of the courses (figure 4.2 Concept-based searching strategy). In this case, we may not find this word through our search because we are looking for the name of the course. Since the course name is related to a professor, courses has to change to the word professor. AIVDISE test if the keyword "courses" appears in the request. If "courses" appears, the change has to be done by the method found through if condition. Also, the keyword "fall" may has to change to "September" because September is part of the fall term. Finally we may arrive at the allowing keywords professor, September and 1999. AIVDISE uses the strategy AND (figure 4.2 AND strategy). When searching each keyword against a webpage, if a keyword does not appear, the page is ignored. This action happens by calling the method indexOf from Java library.

Chapter 6 Comparison

Introduction

In this Chapter, we compare the performance of AIVDISE with the ht://dig search engine. We choose ht://dig because it used on the Acadia University WebPage .

6.1 AIVDISE and ht://dig

The questions that we passed to both search engines:

1. Which courses were offered in fall 1999?
2. Who is teaching comp 2903?
3. Where can I find comp 1113?

6.1.1 Which courses were offered in fall 1999?

6.1.1.1 AIVDISE

This question requested the names of all courses that were offered in the fall of 1999. We received five answers (see figure 6.1) from AIVDISE. Two of The pages are illustrated in figures 6.2 and 6.3. A double click on the chosen address shows a very specific answer to the question

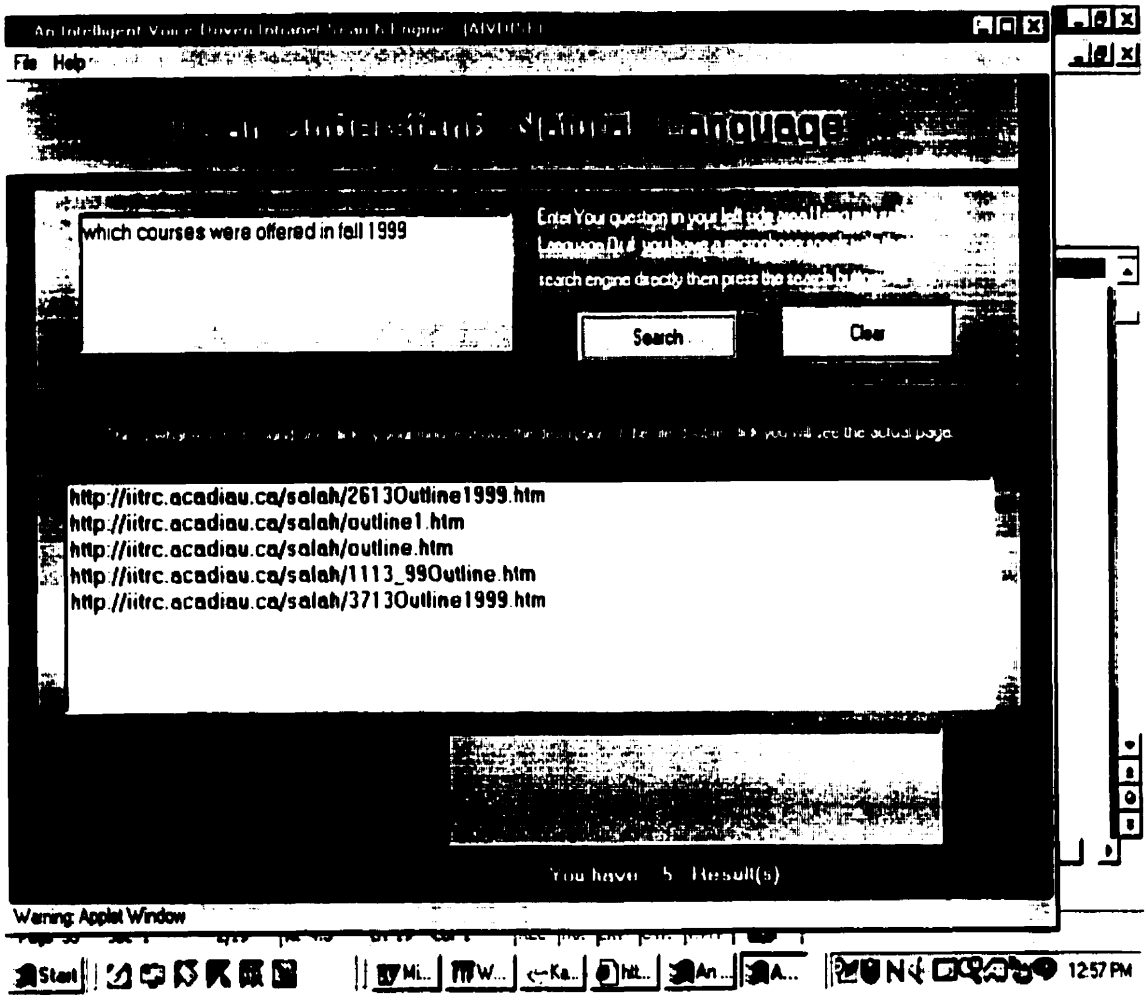

Figure 6.1 AIVDISE Results From Question 1

As you can see figure 6.2 shows one of the right answers retrieved by AIVDISE. It shows the name of the course and when it was offered. Note that "fall 1999" does not appear in the page. AIVDISE knows that "September" is part of the fall term.

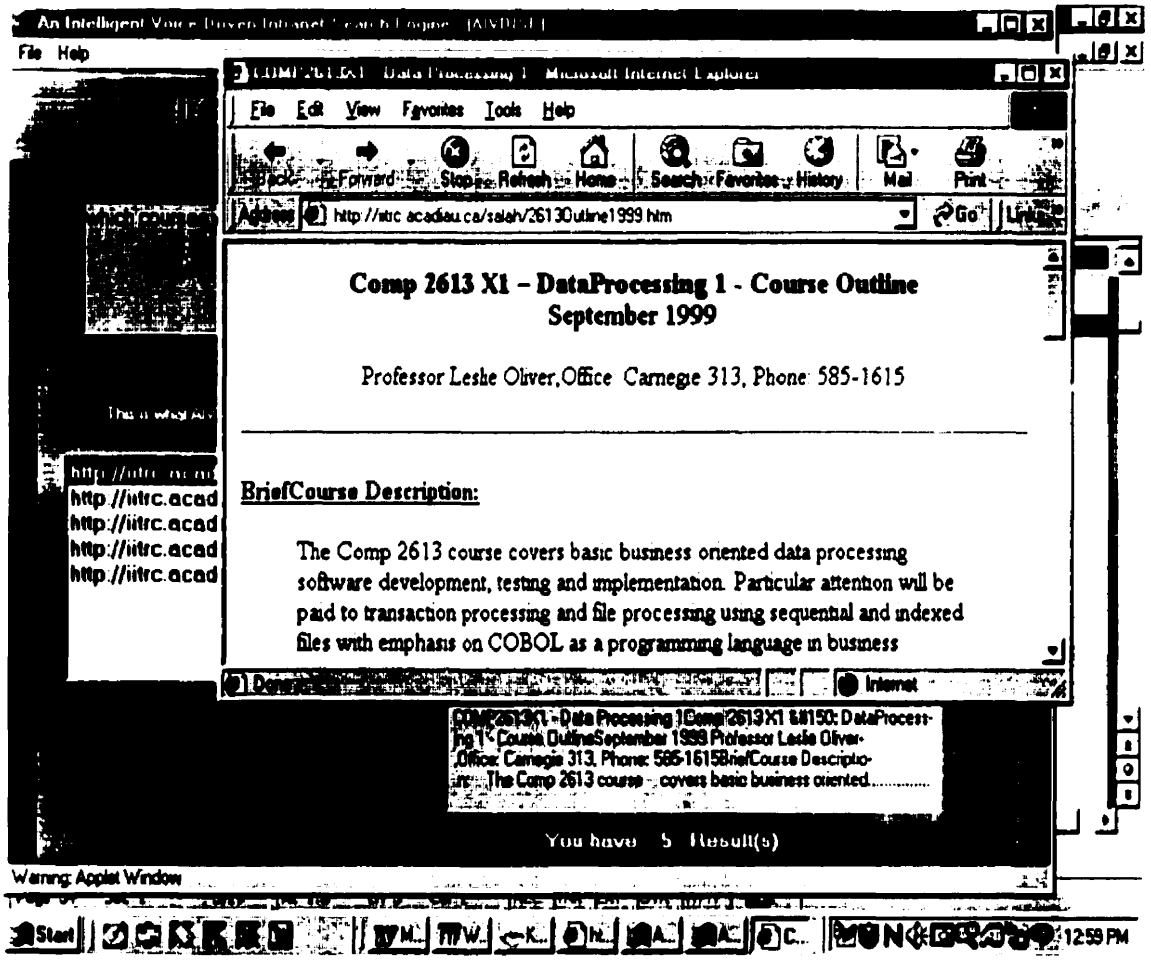

Figure 6.2 AIVDISE First Answer From Question 1

Another correct answer retrieved by AIVDISE shows the name of the course and when it was offered. In this case, the word "fall" is used (Figure 6.3).

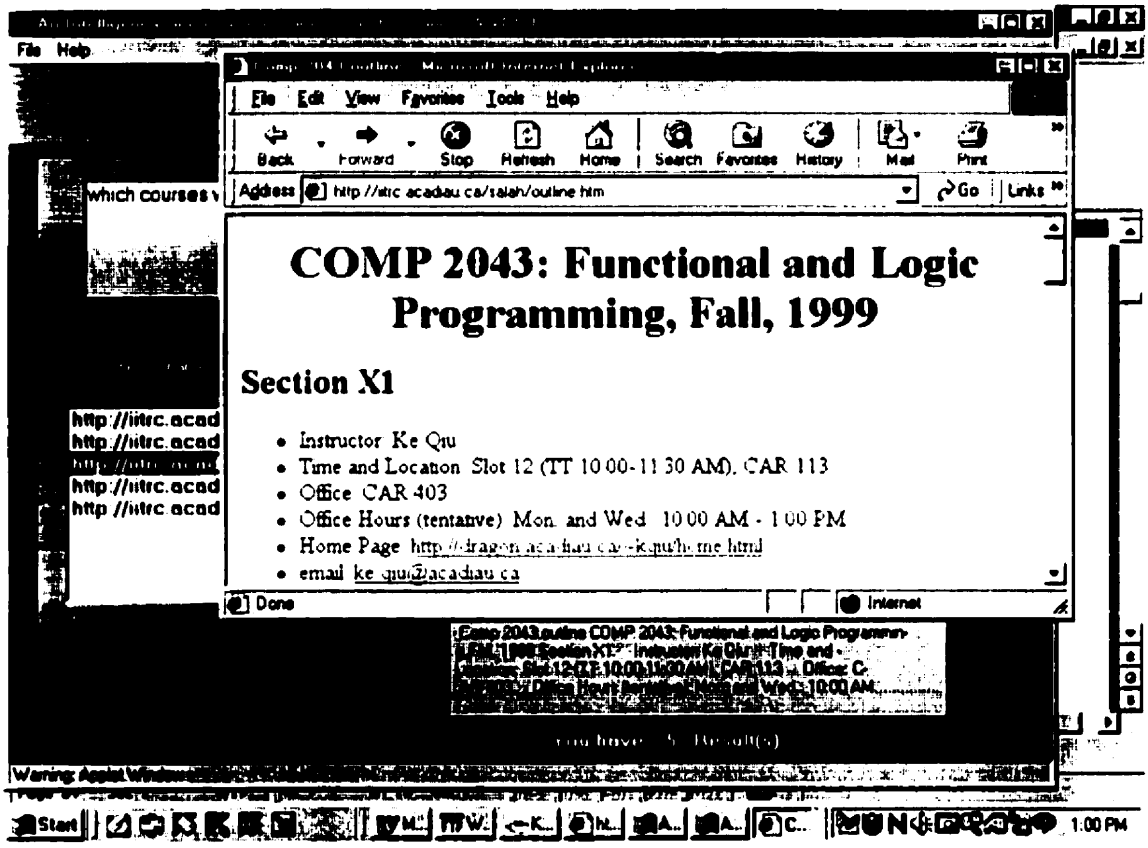

Figure 6.3 AIVDISE Second Answer From Question 1

6.1.1.2 ht://dig

The same question was asked of the ht://dig search engine. We received one hundred and sixteen answers from ht://dig (see figure 6.4).

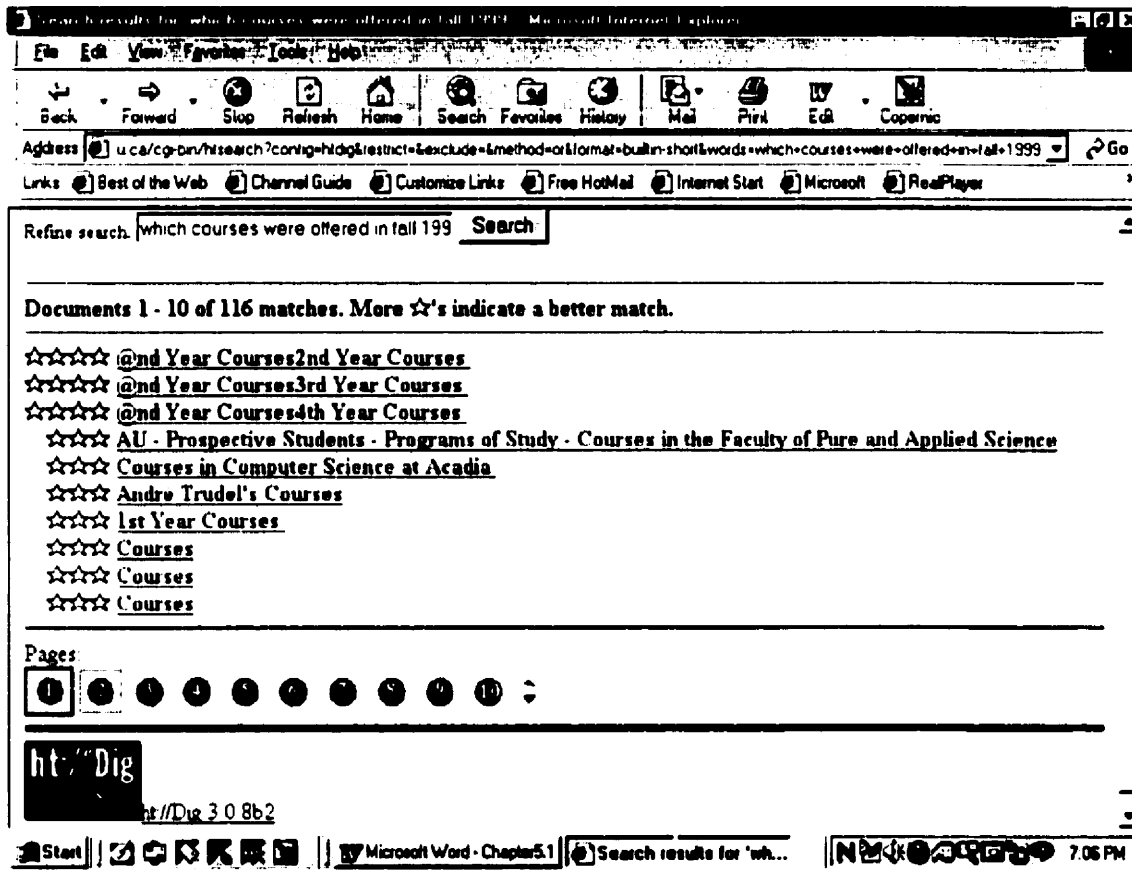

Figure 6.4 ht://dig Results From Question 1

Figure 6.5, shows one of the answers retrieved by ht://dig. It shows a page that contains the typical course load for students doing a BCS program in 2nd year. However, our question had nothing to do with the typical course load for students doing a BCS program in 2nd year. Note that ht://dig ranks this page with four stars which means a perfect answer but the answer was wrong.

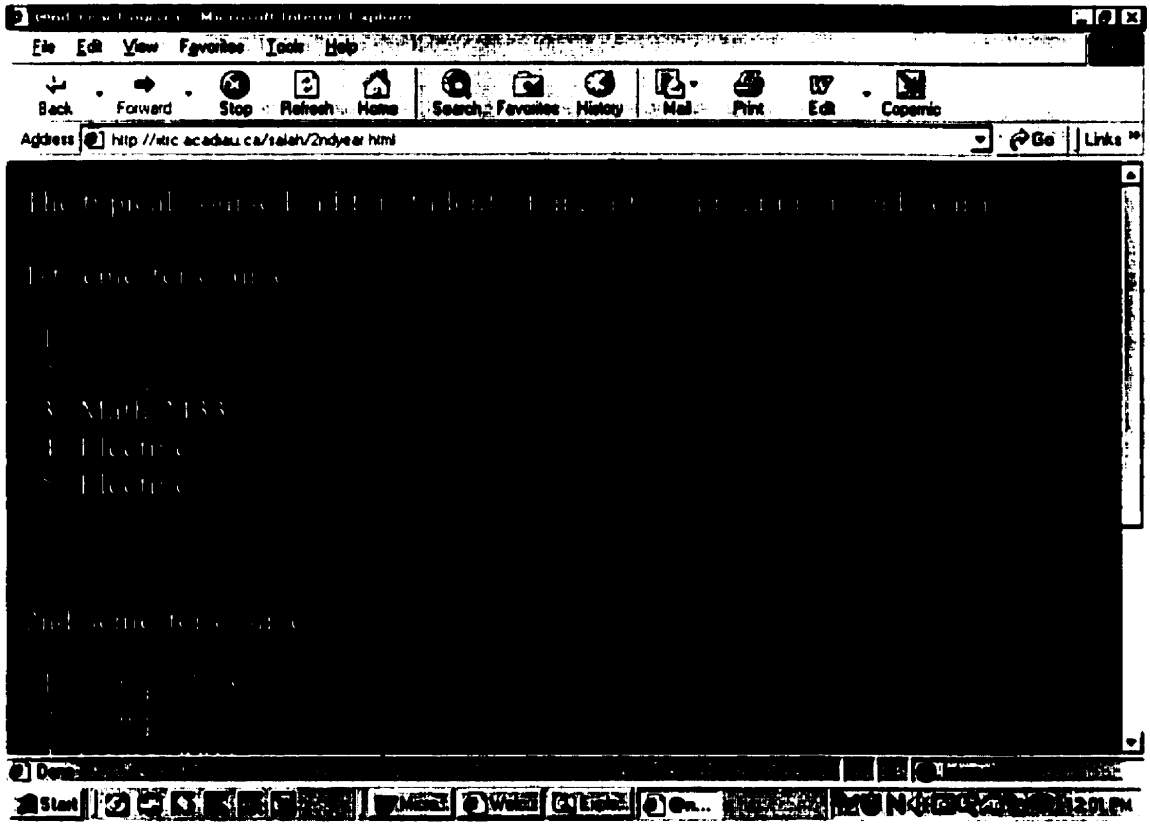

Figure 6.5 ht://dig First Answer From Question 1

Another answer is shown in Figure 6.6. The page concerned Academic Policy and Regulations. Our question was " which courses were offered in fall 1999?". Therefore, the answer was not related to our question.

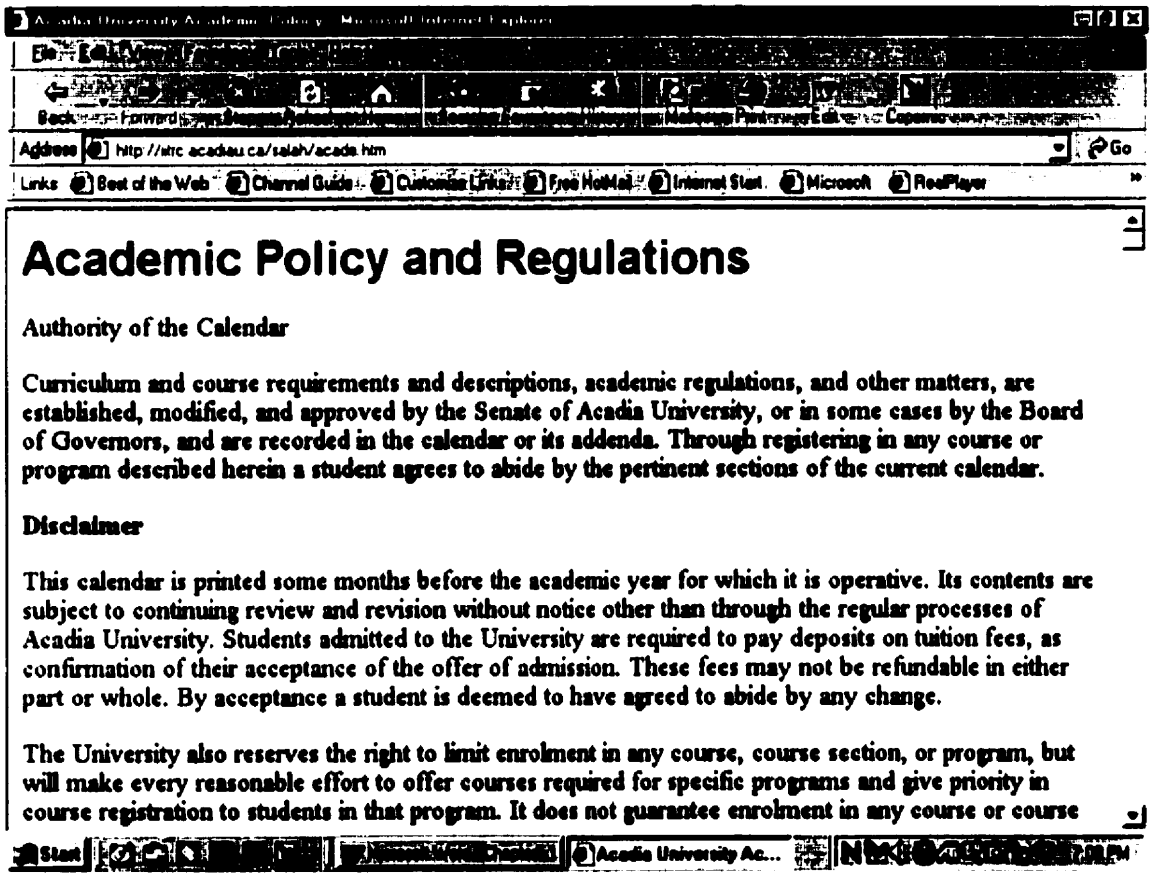

Figure 6.6 ht://dig Second Answer From Question 1

6.1.2 Who is teaching comp 2903?

6.1.2.1 AIVDISE

When we asked AIVDISE this question, we got one answer from the search engine (see figure 6.7) .The actual page is illustrated in figure 6.8.

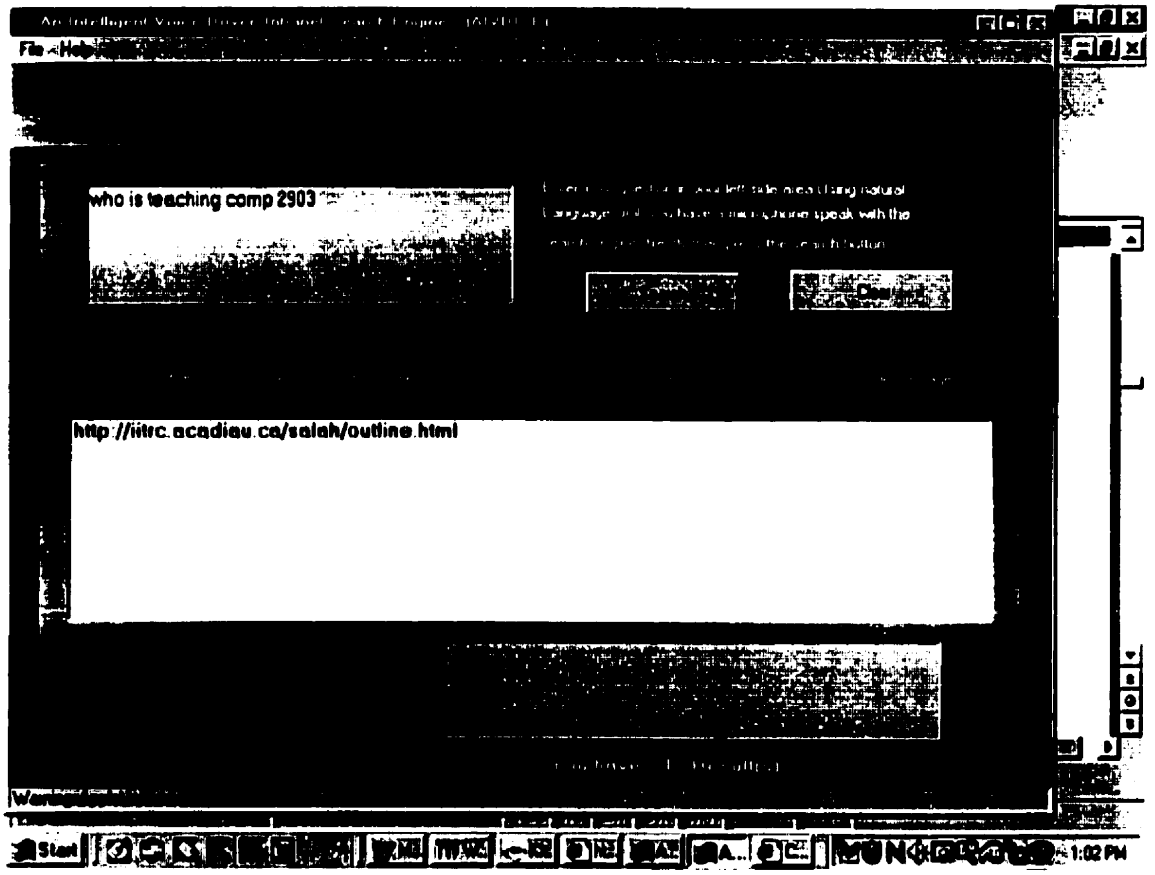

Figure 6.7 AIVDISE Result From Question 2

Figure 6.8 shows the answer of our question, " Who is teaching comp 2903." The page showed us the name of the Professor who teaches that specific course. It was clear Professor Trudel teaches COMP 2903 and that is the correct answer.

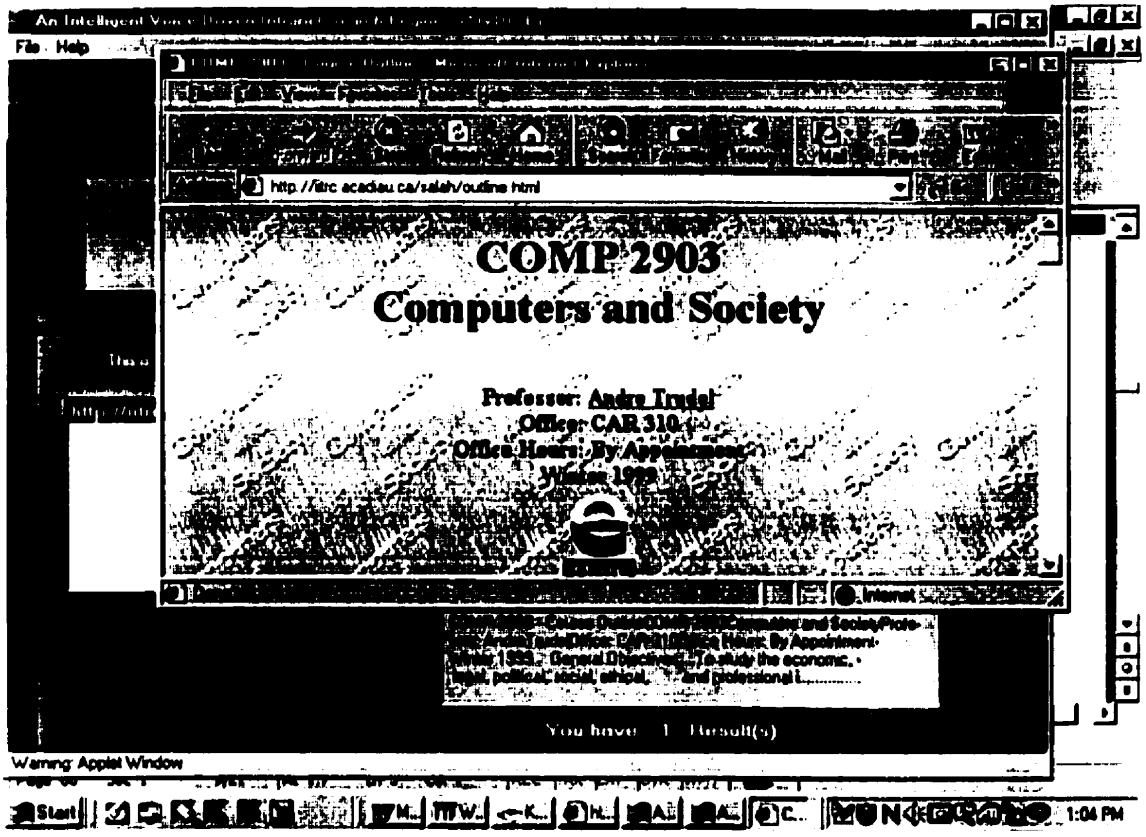

Figure 6.8 AIVDISE Answer From Question 2

6.1.2.2 ht://dig

We received one hundred and thirty six answers from ht://dig (see figure 6.9).The following are examples of the results returned from the search engine (figures 6.10 and 6.11).

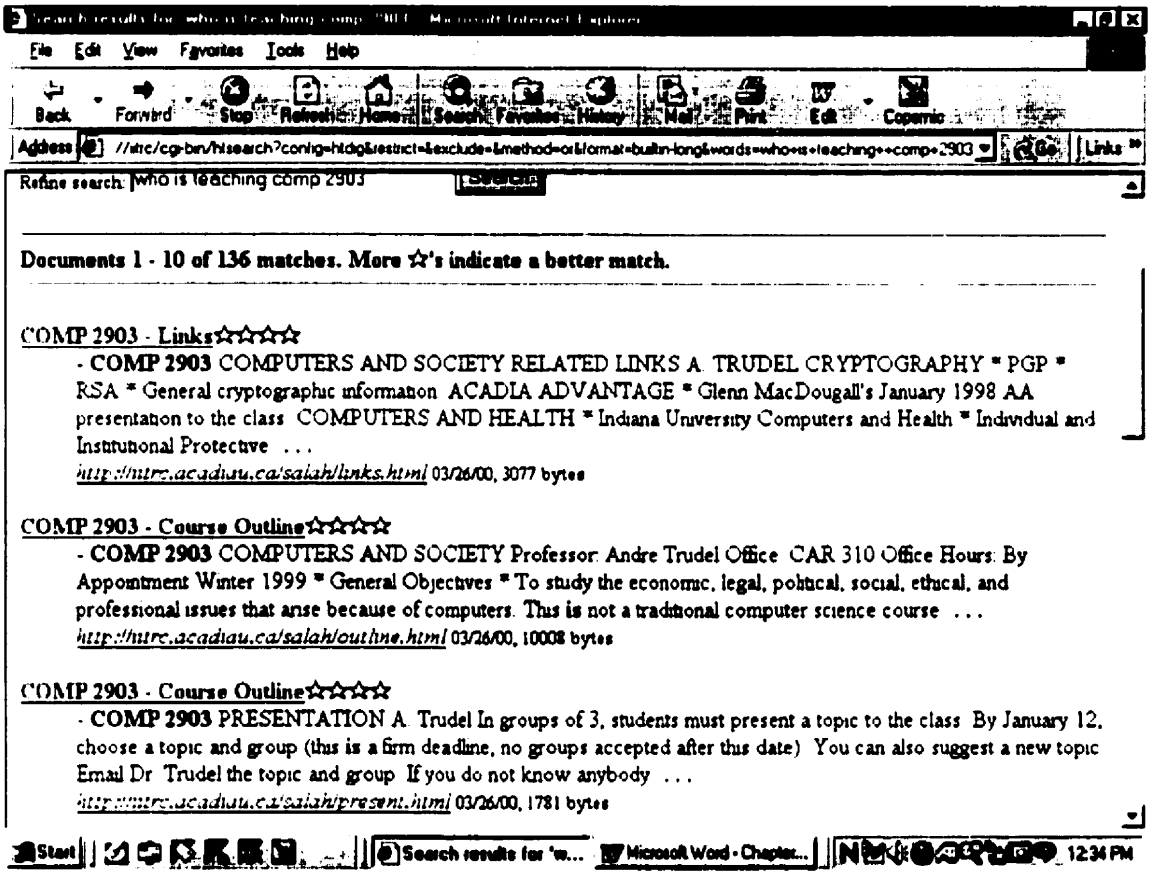

Figure 6.9 ht://dig Results From Question 2

Figure 6.10 illustrates one of the answers that we received from ht://dig. The answer was ranked with four stars but it did not answer our question. Instead, it gave a page containing "COMP 2903 Final Exam".

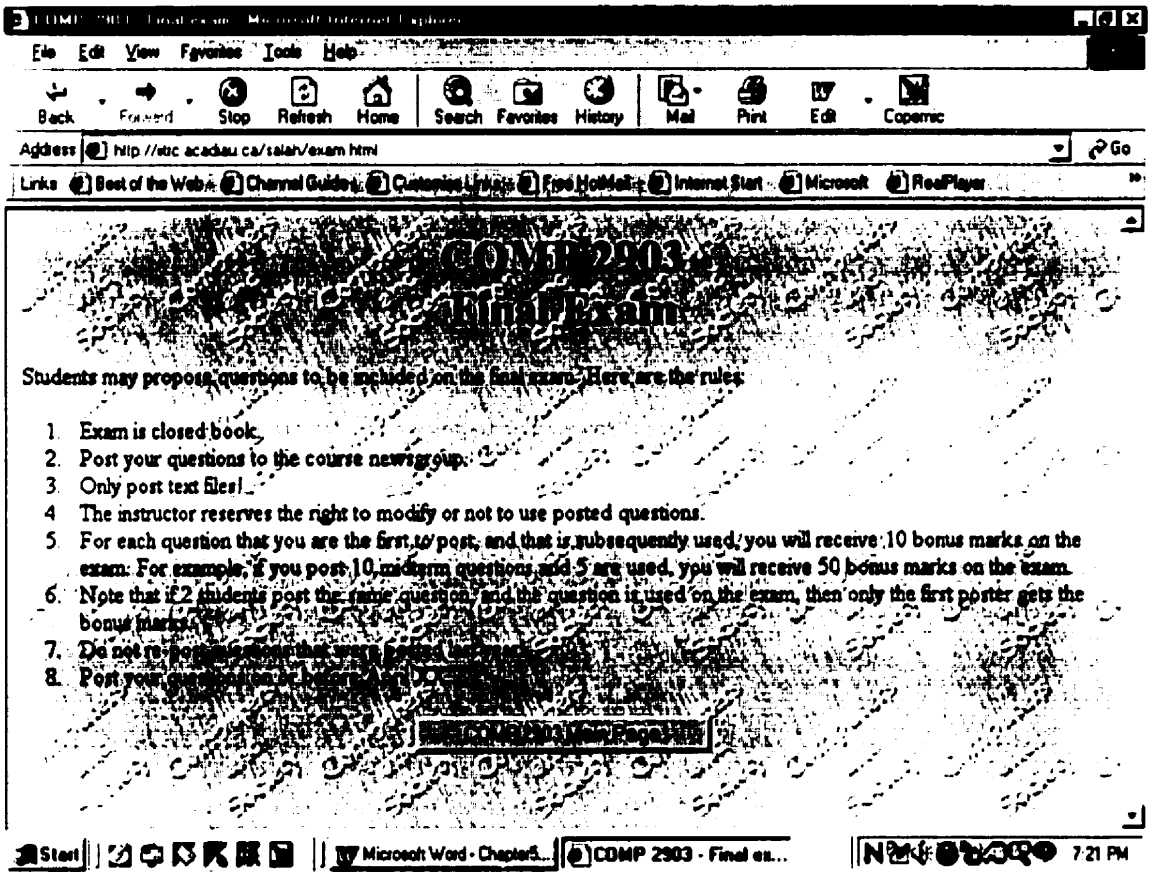

Figure 6.10 ht://dig First Answer From Question 2

Figure 6.11 was also answers about " COMP 4343 COMP NETWORKS/ DISTRIBUTED SYST." This answer by ht://dig was incorrect because our question was "who is teaching comp 2903?"

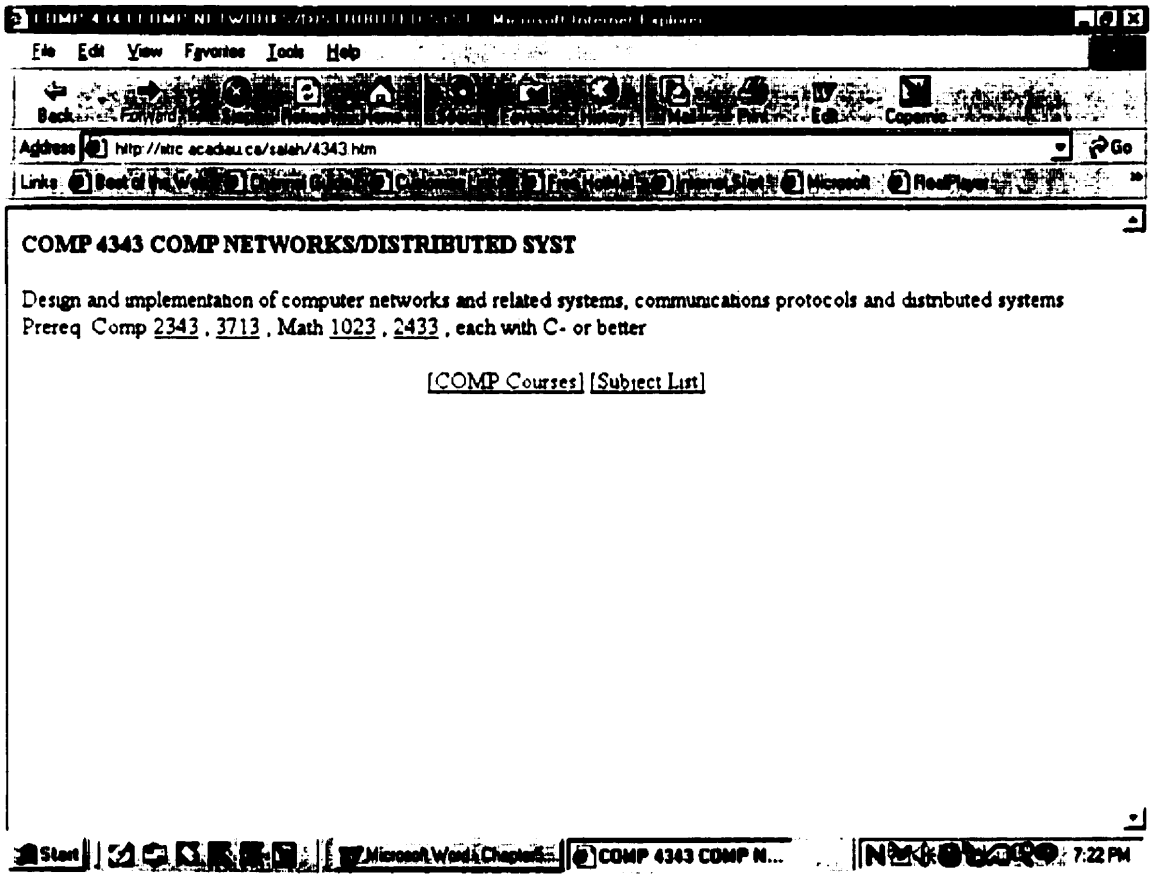

Figure 6.11 ht://dig Second Answer From Question 2

6.1.3 Where can I find comp 1113?

6.1.3.1 AIVDISE

We asked this question and received two answers from AIVDISE (see figure 6.12). The actual pages are illustrated in figures 6.13 and 6.14.

Figure 6.12 AIVDISE Results From Question 3

Figure 4.13 shows the page containing information about comp 1113. In figure 6.14, the answer was correct and gave the course outline of Comp 1113. The answers on both pages were related to our question and AIVDISE succeeded in finding the right answer.

Figure 6.13 AIVDISE First Answer From Question 3

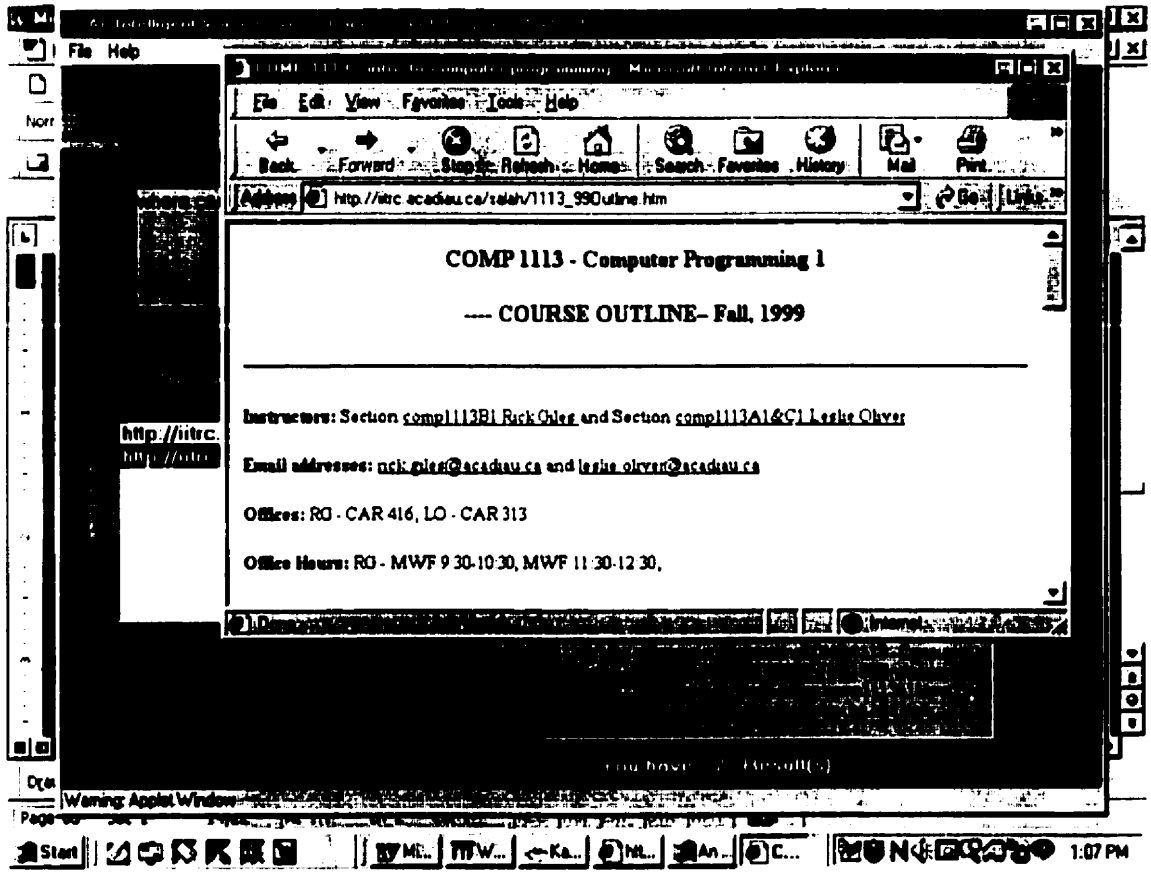

Figure 6.14 AIVDISE Second Answer From Question 3

6.1.3.2 ht://dig

To the question "Where can I find comp 1113?" we received eighty nine answers from ht://dig (see figure 6.15).The following figures are examples from the results received from the search engine (see figures 6.16 and 6.17).

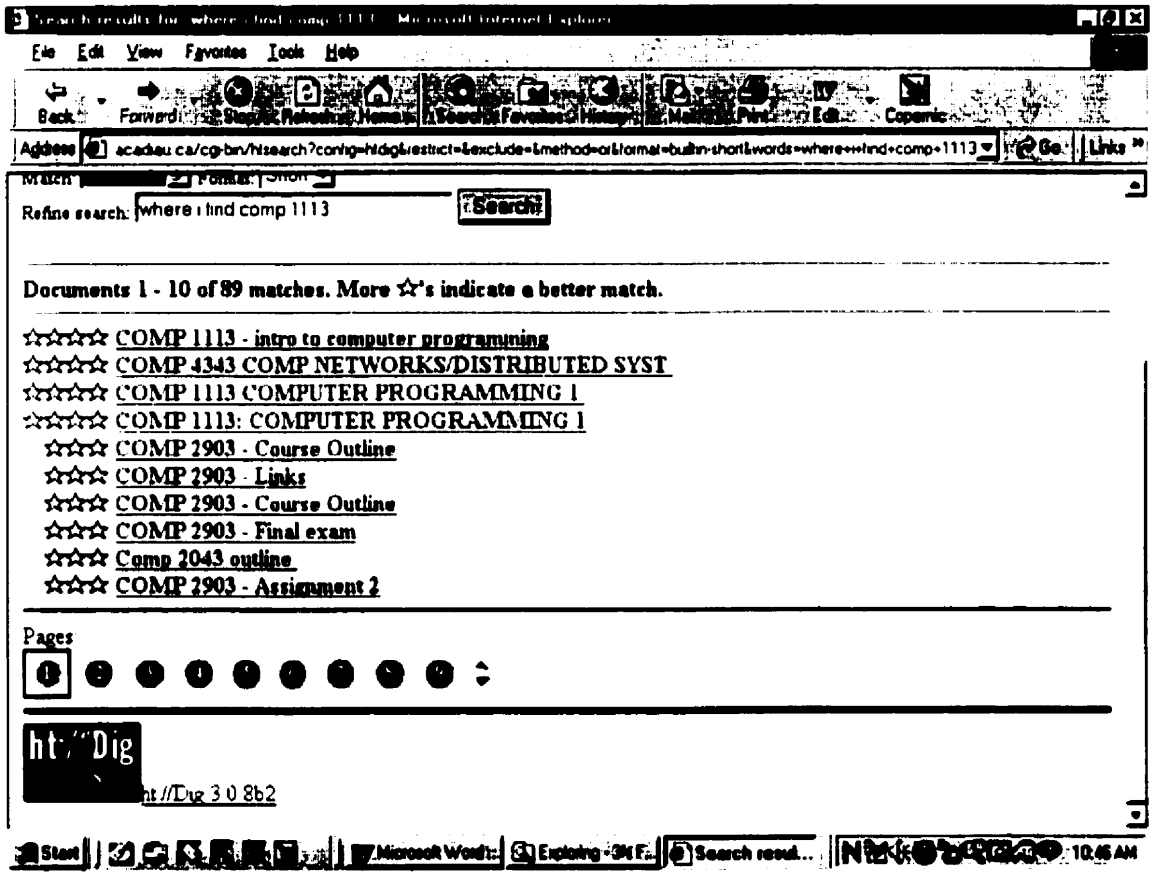

Figure 6.15 ht://dig Results From Question 3

In Figure 6.16, one of the answers retrieved by ht://dig contained information about COMP 4343 COMP NETWORKS/ DISTRIBUTED SYST. It is clear the answer was not correct because our question concerned comp 1113.

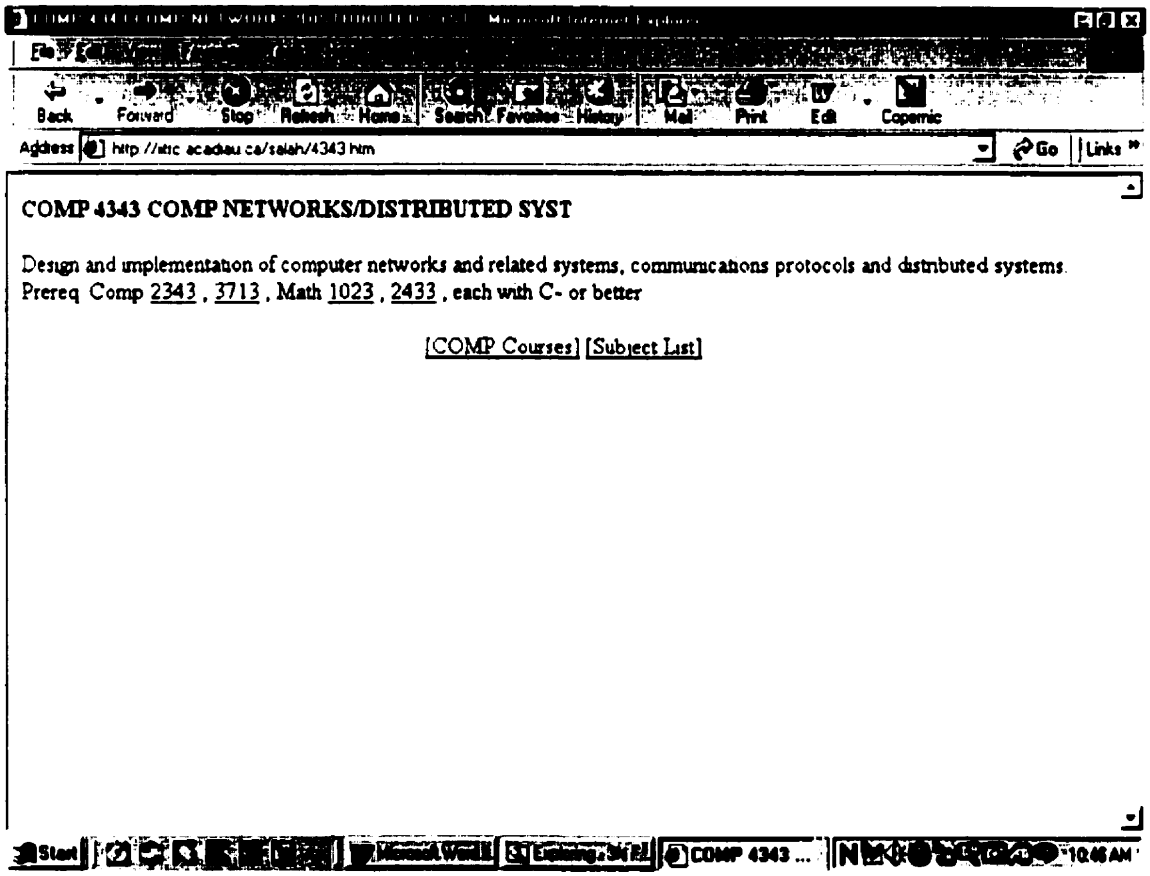

Figure 6.16 ht://dig First Answer From Question 3

Another answer from ht://dig is illustrated in figure 6.17. The page about "COMP 2903 Assignment 2" was wrong.

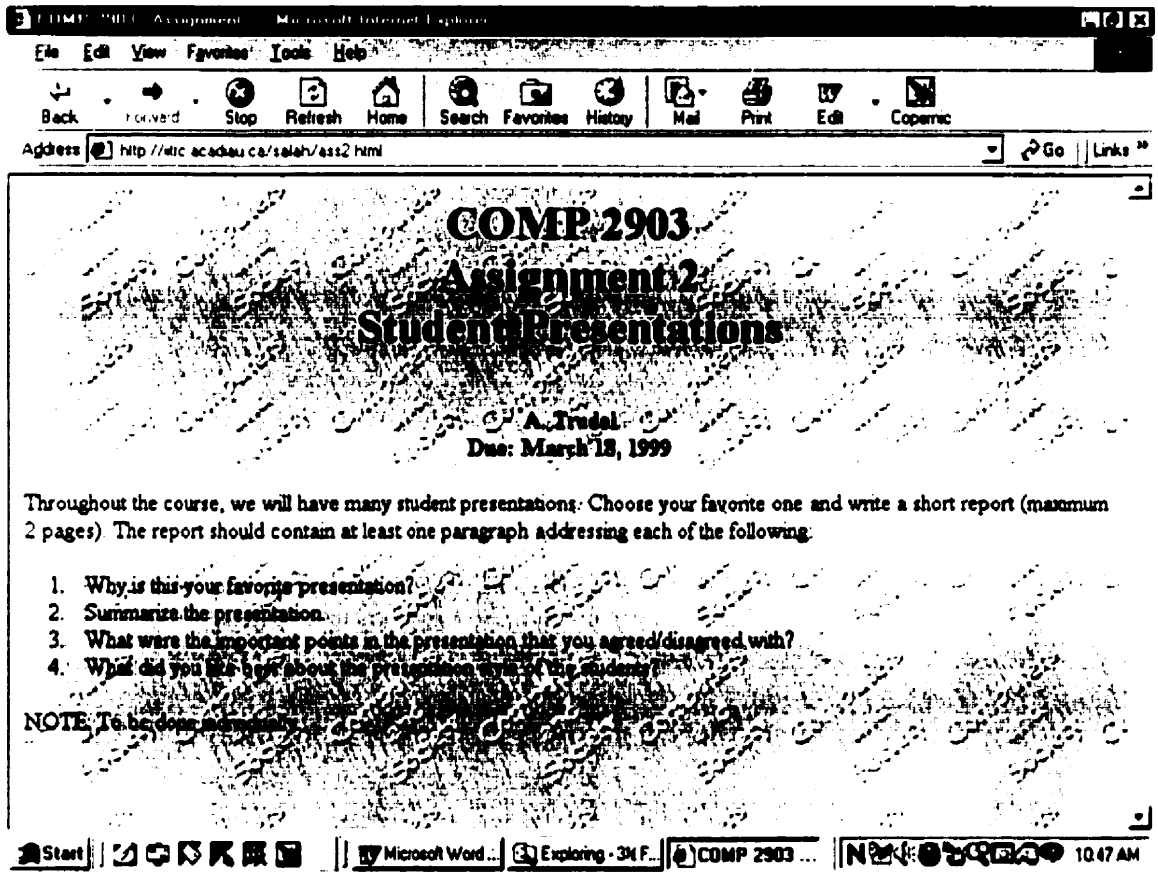

Figure 6.17 ht://dig Second Answer From Question 3

Conclusion

In the first three questions, the results from AIVDISE were excellent. In answer to question number one, AIVDISE showed only the results of the courses offered in fall 1999. On the other hand, ht://dig showed one hundred and sixteen results and some of them had nothing to do with the question, "Which courses were offered in fall 1999?". The next question, "Who is teaching comp 2903?" was supposed to yield the name of the professor who is teaching the course comp 2903.There was one result from AIVDISE and that was exactly what we requested. However, we got one hundred and thirty six answers from ht://dig and some were not remotely related to our question. The last natural language question that we asked both search engines was, "Where can I find comp 1113?". The result from AIVDISE was two answers. Both answers were related to our question. However, ht://dig did not return the right answer.

Chapter 7 Conclusions and directions for future work

7.1 Conclusion

In this thesis, we learned about internet and intranet search engines. We discussed the history of search engines, where we can use them, and why we need them. We discussed the importance of an intranet search engine. We explored the hidden mechanism of different search engines, including the process of accessing a page. We illustrated some popular intranet and internet search engines, such as ht://Dig , Phantom, Altavista, Yahoo, WebEnhancer and Searchlink. We also discussed why there are so many dead links (sites are no longer in existence) returned from popular search engines.

The aim of this thesis was to build An Intelligent Voice Driven Intranet Search Engine (AIVDISE) for a personal or a business website, for example the Acadia website. Our goal was to develop a search engine that could accept verbal English language sentences as input. We detailed our strategies and the methodologies used in our design. To increase the accuracy and speed of the search engine, we stored and used specialized knowledge about the Acadia website. Unlike keyword search systems, AIVDISE tries to determine what you mean, not what you say. In the best circumstances, AIVDISE returns hits on documents which pertain to the subject for which you were looking, even

if the words in the document do not precisely match the words you entered in the query. Currently, AIVDISE retrieves text information by searching for meaning and concepts in context.

AIVDISE is a user-friendly search engine. By typing plain English and/or using voice control, the user will be able to give a command to the search engine and it will execute the request.

The literature on natural language versus Boolean keyword searching makes it clear that NL input has certain overall advantages. For example, AIVDISE has an advantage over ht://dig since AIVDISE can retrieve the right information by using natural language as input. A single keyword query will retrieve many documents which will be irrelevant. Often, choosing the correct keyword means the difference between a successful search and a failure. Even when the right keyword is chosen, the relevant documents are often hidden within a myriad of irrelevant information. Sometimes users are required to combine several keywords. Many users do not understand boolean expressions or the correct keywords to combine. Since the search engine "knows" the contents of the intranet, it should be possible to engage that internal knowledge and put the query in its proper context.

7.2 Directions for future work.

We have demonstrated through our work that by utilizing natural language and voice control, AIVDISE provides a new and improved tool for searching the intranet. However, there are many ways in which this work can be extended. Currently, AIVDISE

searches a website by using simple sentences as input. AIVDISE could be improved to accept more complicated sentences such as "show me all types of search engines and how they work".

AIVDISE could also be improved by building a spider that can help AIVDISE. This spider would collect all the data from the website and build an index file automatically.

Bibliography:

[Barl 96] Monash Information Services. by Linda R. Barlow
Available at:
http://www.monash.com/barlow.html

[Corn 94] Cronin. Mary J. Doing Business on the Internet. New York: Van Nostrand Reinhold. 1994.

[Deltix 99] WebEnhancer founded in Deltix Software tools for web development
Available at:
http://www.deltix.com/search.html

[Garb 99] Searching for New Search Technologies By Ilan Greenberg and Lee Garber .Computer innovation Technology for computer Professionals. August 1999 available via:
http://computer.muni.cz/csinfo/stafforg/pubmags/lgarber.htm

[Hersh and Day 97] Hersh. W. and B. Day (Oregon Health Sciences University). 1997. A Comparison of Boolean and Natural Language Searching for the

TREC6 Interactive Task , in Voorhees and Harman 1997.

NIST Special Publication 500-240. Information Technology.--E. M. Voorhees and D. K.Harman, Editors. Co-sponsored by the National Institute of Standards and Technology (NIST) and the Defense Advanced Research Projects Agency (DARPA). Contains the proceedings of the sixth Text REtrieval Conference, held in Gaithersburg, Maryland, November 19-21, 1997. Evaluates new technologies in text retrieval.

Publisher:

Commerce Dept., Technology Administration, National Institute of Standards and Technology, Information Technology Laboratory

[leen 99] SearchLite by thomas leen available at http://members.xoom.com/holzp/searchlite.html

[Libr 99] Bartlesville Public Library(1999) : Learn how to search. Available via: http://www.bartlesville.lib.ok.us/boolean.htm

[Maci 98] Guide to search Engines by Wes Sonnernreich and Tim Macinta.

Publisher: John Wiley & Sons 1998.

[Note 99] Search Engine Statistics: Dead Links Report by Greg R. Notess Sept 10,1999 : Available Via: http://www.notess.com/search/stats/dead.shtml

[Phan 99] Maxum Development Corporation. Phantom ver 2.2. Available via: http://www.maxum.com/Phantom/

[Rich 98] Home Page Search Applet by Richard Everitt 1998 Available via http://www.babbage.demon.co.uk/java.html

[Rijs 79] Rijsbergen, C. J. v. (1979). Information Retrieval. London, Butterworths.

[Sche 99] Andrew Scherpbier: WWW Search Engine Software . Available via: http://www.htdig.org/

[Serv 99] 1999 Command Internet Services (1999). Search engine. Available via: http://www.commandnet.net/support/search.htm

[Silk 97] Searchlink developed by Silk webware 1997 Available at: http://silk.webware.co.nz/Products/Searchlink/

[Turt 94] Turtle, Howard R. 1994. Natural Language vs. Boolean Query Evaluation: A Comparison of Retrieval Performance. In Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR 94), Dublin, Ireland, July 1994.

Publisher:

Commerce Dept., Technology Administration. National Institute of Standards and Technology. Information Technology Laboratory

[whatis 97] whatis.com(1997). a knowledge exploration tool about information technology. Available via:

http://www.whatis.com/scooter.htm

[webp 98] WebPromote newsletter1998. The intelligent internet marketing Resource. Search Engine Update Available via:

http://www.webpromote.com/

Appendix A

AIVDISE Requirements and Installation

1. System Requirements.

1- JDK ver 1.1 or higher to compile and run AIVDISE. JDK software and documentation is free per the license agreement.

2- We used Personal web server because it is available free at microsoft.com.

3- Dragon NaturallySpeaking Preferred ver 3.01 or higher for speech recognition.

4- Microsoft windows 95 or higher.

Processer requirements

At least a 133 MHz Pentium® processor.

Memory Requirements

At least 32 MB.

Hard Disk Space

At least 75 MB free hard disk space.

Installation

1. AIVDISE

1- Download the personal web server from

http://www.microsoft.com/downloads/

2- Download jdk 1.1 from

http://java.sun.com/products/jdk/1.1/

3- In the file AIVDISE.html change the VALUE in <PARAM NAME=="server" VALUE= "homepageaddress" > to your home page address. If you are using personal web server, the home page address will be the name of the machine.

4- To run AIVDISE , in your browser write the address of your homepage followed by the filename INVDISE.html. For Example , http://131.162.166.112/INVDISE.html

5- By pressing the open button, the main user interface will appear.

II. Dragon naturally speaking preferred ver 3.01

1- Start Windows.

2- Put the disc in the CD-ROM drive

3- On the start menu , click RUN

4- Type D:\SETUP, then press Enter . The Setup wizard guides you through the installation.

User Guide

Main user interface

Once the user presses the open button, the system opens the main user interface.

User requests

The User request section is responsible for the requests from the user and sends the queries to the search engine.

File drop menu

File is a drop menu that lets the user exit from AIVDISE.

Help drop menu

The Help drop menu contains the About menu item and the Instructions menu item.

Instructions Menu Item

Clicking on Instructions opens the window. It will guide the user in the use of AIVDISE.

About Item Menu

This menu item shows information about the search engine implementation.

Text area

The text area is responsible for accepting the request from the user in either typed or spoken English.

Search button

By pressing the Search button, AIVDISE accepts the question from the text area and starts searching.

Clear button

When the user presses the clear button, AIVDISE clears any information in the User requests, Results, and Description sections. Then the user can make a new search.

Results area

In this section, the user can see the search results. Only addresses are shown and the user can access a page by double clicking on it.

Description

This section shows a simple description of each page found by AIVDISE. The user can click on the address in the result area and AIVDISE will show the description in the description area.

Appendix B

In this Appendix, we list the queries used to test AIVDISE.

What courses were offered in winter 1999

What courses were offered in fall 1999

Who is teaching comp 2043?

Who is teaching comp 3403?

Who is teaching comp 2903?

Where is comp 2903?

Comp 1113

Where can i find comp 1113?

Who is the director ?

Show me first year courses

Where is a first year course?

First year courses

Find first year courses

I need second year courses

Who is teaching comp 2903?

Show me second year courses?

What dr. trudel teaches?

Where can I find computer professors?

Show me computer professors

Give me second year courses

Who is teaching comp 1113?

Where can I find comp 1113?

What dr. Giles teaches

Show me third year courses

What dr. Oliver teaches

Show me fourth year courses

What dr. Qiu teaches

Winter courses

Fall courses

Appendix C

Source code

```
/**
 * This file "AboutDialog.java"  provides  the  necessary
 *  methods and attributes for  the the aboutDialog class.
 *
 * Author:       Salah Ali
 * Version:      2.4  April 13      1999
 *
 *
import java.awt.*;
/**
 *       the About  class        contains all the elements
 *       necessary to prsent an information about the search engine like the version , copy
right.
 *
 * @version  1.3  April 13 1999
 * @author   Salah Ali
 **/
```

```
/*
        A basic extension of the java.awt.Dialog class
*/
import java.awt.*;
public class AboutDialog extends Dialog {
        public AboutDialog(Frame parent, boolean modal)
        {
                super(parent, modal);
                setLayout(null);
                setVisible(false);
                setSize(512,232);
                panel1 = new java.awt.Panel();
                panel1.setLayout(null);
                panel1.setBounds(2,18,512,232);
                panel1.setForeground(new Color(0));
                panel1.setBackground(new Color(12632256));
                add(panel1);
                okButton = new java.awt.Button();
                okButton.setLabel("OK");
                okButton.setBounds(223,177,66,27);
                panel1.add(okButton);
                list1 = new java.awt.List(4);
```

```
        list1.addItem("       An Intelligent Voice Driven Intranet Search Engine
");
        list1.addItem("                    ");
        list1.addItem("                              (AIVDISE)");
        list1.addItem(" ");
        list1.addItem("                          Acadia University");
        list1.addItem("");
        list1.addItem("Copyright © 2000  Salah Aboulkhasam");
        panel1.add(list1);
        list1.setBounds(53,12,406,133);
        list1.setFont(new Font("Dialog", Font.PLAIN, 14));
        list1.setForeground(new Color(16711680));
        list1.setEnabled(false);
        setTitle("About AIVDISE ");
        }}

        //{{REGISTER_LISTENERS
        SymWindow aSymWindow = new SymWindow();
        this.addWindowListener(aSymWindow);
        SymAction lSymAction = new SymAction();
        okButton.addActionListener(lSymAction);
        //}}
```

```
    }

    public AboutDialog(Frame parent, String title, boolean modal)
    {
        this(parent, modal);
        setTitle(title);
    }
    public void addNotify()
    {
        // Record the size of the window prior to calling parents addNotify.
      Dimension d = getSize();
        super.addNotify();
        // Only do this once.
        if (fComponentsAdjusted)
            return;
        // Adjust components according to the insets
        setSize(insets().left + insets().right + d.width, insets().top + insets().bottom
+ d.height);
        Component components[] = getComponents();
        for (int i = 0; i < components.length; i++)
        {
            Point p = components[i].getLocation();
            p.translate(insets().left, insets().top);
```

```
                components[i].setLocation(p);
            }
            // Used for addNotify check.
            fComponentsAdjusted = true;
        }
        public void setVisible(boolean b)
        {
            if (b)
            {
                Rectangle bounds = getParent().bounds();
                Rectangle abounds = bounds();
                move(bounds.x + (bounds.width - abounds.width)/ 2,
                        bounds.y + (bounds.height - abounds.height)/2);
            }
            super.setVisible(b);
        }
        //{{DECLARE_CONTROLS
        java.awt.Panel panel1;
        java.awt.Button okButton;
        java.awt.List list1;
        //}}

    // Used for addNotify check.
```

```
boolean fComponentsAdjusted = false;

class SymWindow extends java.awt.event.WindowAdapter
{
    public void windowClosing(java.awt.event.WindowEvent event)
    {
        Object object = event.getSource();
        if (object == AboutDialog.this)
            AboutDialog_WindowClosing(event);
    }
}

void AboutDialog_WindowClosing(java.awt.event.WindowEvent event)
{
  dispose();
}

class SymAction implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == okButton)
            okButton_Clicked(event);
```

```
            }
        }

        void okButton_Clicked(java.awt.event.ActionEvent event)
        {
            //{{CONNECTION
            // Clicked from okButton Hide the Dialog
          dispose();
            //}}
        }
}

**
* This file "AboutDialog.java" provides the necessary
*  methods and attributes for the the aboutDialog class.
*
* Author:       Salah Ali
* Version:      2.4  April 13      1999
*
*/

import java.awt.*;
```

```
**
*       the About  class       contains all the elements
*       necessary to present an information about the search engine like the version ,
copy right.
*
* @version  1.3  April 13 1999
* @author   Salah Ali
**
import java.awt.*;
public class AboutDialogtext extends Dialog {

        public AboutDialogtext(Frame1 parent, boolean modal)
        {
                super(parent, modal);
                setLayout(null);
                setVisible(false);
                setSize(400,340);
                panel1 = new java.awt.Panel();
                panel1.setLayout(null);
                panel1.setBounds(20,20,365,305);
                panel1.setBackground(new Color(16776960));
                add(panel1);
                okButton = new java.awt.Button();
```

```
okButton.setLabel("OK");
okButton.setBounds(150,220,66,27);
okButton.setBackground(new Color(16711680));
panel1.add(okButton);
panel2 = new java.awt.Panel();
panel2.setLayout(null);
panel2.setBounds(120,10,106,80);
panel1.add(panel2);
scrollingText2 = new java.Component.ScrollingText();
try {
        java.lang.String[] tempString = new java.lang.String[1];
        tempString[0] = new java.lang.String("Salah Ali");
        scrollingText2.setMessageList(tempString);
}
catch(java.beans.PropertyVetoException e) { }
scrollingText2.setBounds(0,40,110,30);
scrollingText2.setBackground(new Color(16711680));
panel2.add(scrollingText2);
scrollingText3 = new symantec.itools.multimedia.ScrollingText();
try {
        java.lang.String[] tempString = new java.lang.String[1];
        tempString[0] = new java.lang.String("Acadia University");
        scrollingText3.setMessageList(tempString);
```

```
        }
        catch(java.beans.PropertyVetoException e) { }
        scrollingText3.setBounds(90,100,180,30);
        scrollingText3.setBackground(new Color(16711680));
        panel1.add(scrollingText3);
        scrollingText4 = new symantec.itools.multimedia.ScrollingText();
        try {
                java.lang.String[] tempString = new java.lang.String[1];
                tempString[0] = new java.lang.String("Copy right 1999");
                scrollingText4.setMessageList(tempString);
        }
        catch(java.beans.PropertyVetoException e) { }
        scrollingText4.setBounds(120,160,120,20);
        scrollingText4.setBackground(new Color(16711680));
        panel1.add(scrollingText4);
        scrollingText1 = new symantec.itools.multimedia.ScrollingText();
        try {
                java.lang.String[] tempString = new java.lang.String[1];
                tempString[0] = new java.lang.String("An Intelligebt Voice Driven
Interanet Search Engine");
                scrollingText1.setMessageList(tempString);
        }
        catch(java.beans.PropertyVetoException e) { }
```

```
        scrollingText1.setBounds(20,10,320,21);
        scrollingText1.setBackground(new Color(16711680));
        panel1.add(scrollingText1);
        setTitle("About");
        //}}
        //{{REGISTER_LISTENERS
        SymWindow aSymWindow = new SymWindow();
        this.addWindowListener(aSymWindow);
        SymAction lSymAction = new SymAction();
        okButton.addActionListener(lSymAction);
        //}}
    }
    public AboutDialogtext(Frame parent, String title, boolean modal)
    {
        this(parent, modal);
        setTitle(title);
    }
    public void addNotify()
    {
        // Record the size of the window prior to calling parents addNotify.
      Dimension d = getSize();
        super.addNotify();
```

```
            // Only do this once.
            if (fComponentsAdjusted)
                return;
            // Adjust components according to the insets
            setSize(insets().left + insets().right + d.width, insets().top + insets().bottom
+ d.height);
            Component components[] = getComponents();
            for (int i = 0; i < components.length; i++)
            {
                Point p = components[i].getLocation();
                p.translate(insets().left, insets().top);
                components[i].setLocation(p);
            }
            // Used for addNotify check.
            fComponentsAdjusted = true;
        }
        public void setVisible(boolean b)
        {
          if (b)
          {
            Rectangle bounds = getParent().bounds();
            Rectangle abounds = bounds();
```

```
            move(bounds.x + (bounds.width - abounds.width)/ 2,
                    bounds.y + (bounds.height - abounds.height)/2);
        }
        super.setVisible(b);
    }
    //{{DECLARE_CONTROLS
    java.awt.Panel panel1;
    java.awt.Button okButton;
    java.awt.Panel panel2;
//}}
    // Used for addNotify check.
    boolean fComponentsAdjusted = false;
    class SymWindow extends java.awt.event.WindowAdapter
    {
        public void windowClosing(java.awt.event.WindowEvent event)
        {
            Object object = event.getSource();
            if (object == AboutDialogtext.this)
                AboutDialog_WindowClosing(event);
        }
    }
    void AboutDialog_WindowClosing(java.awt.event.WindowEvent event)
    {
```

```
            dispose();
        }

        class SymAction implements java.awt.event.ActionListener
        {
                public void actionPerformed(java.awt.event.ActionEvent event)
                {
                        Object object = event.getSource();
                        if (object == okButton)
                                okButton_Clicked(event);
                }
        }
        void okButton_Clicked(java.awt.event.ActionEvent event)
        {
            dispose();
                }
        }
}

import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
```

```
import java.util.*;
import java.awt.event.*;
import MainGUI;

public class SearchEngine extends Applet implements ActionListener
{
        private Button openButton = new Button ("  OPEN");
        MainGUI mainGUI;
        String server;
        String main_page;
        int totalPages;

        int width1, high1;      // Width and high1ht of the main user interface
        public void init()
        {
                getParameters();
                main_page = new String(server + homepage);
  Panel buttonPanel = new Panel();
  buttonPanel.add(openButton);
  add(buttonPanel);
  openButton.addActionListener(this);
        }
```

```
        public void getParameters()
        {

                String num = getParameter("totalPages");
                totalPages = Integer.parseInt(num);
           String       homepage = getParameter("homepage")
                String docBase = this.getDocumentBase().toString();
                    int endIndex = docBase.lastIndexOf("/");
                    server = new String(docBase.substring(0,endIndex) + "/");
        String w = getParameter("width1");
                    width1 = Integer.parseInt(w);
        String h = getParameter("high1");
                    high1 = Integer.parseInt(h);
        }
        public void actionPerformed(ActionEvent e)
        {
              if (e.getSource() == openButton)
        mainGUI = new MainGUI(this, (Frame)f, "An Intelligent Voice Driven Intranet
Search Engine  (AIVDISE) ");
         mainGUI.resize(width1,high1);
        mainGUI.show();
        }
}
```

```
/**
* This file "mainGUI.java" provides the necessary
* methods and attributes for the the mainGUI class.
*
* Author:       Salah Ali
* Version:      2.4 may 27   1999
*
*/
import java.awt.*;
import Search;
/**
*       the mainGUI class     contains all the elements
*       necessary to act as the main window of an application.
*
* @version 1.3 may 27 1999
* @author  Salah Ali
*/
public class MainGUI extends Frame
{
        // Used for addNotify check.
        boolean fComponentsAdjusted = false;
        //{{DECLARE_CONTROLS
        java.awt.FileDialog openFileDialog1;
```

```
        java.awt.Label head_Label;
        //}}

        //{{DECLARE_MENUS
        java.awt.MenuBar mainMenuBar;
        java.awt.Menu menu1;
        java.awt.MenuItem miExit;
        java.awt.Menu menu3;
        java.awt.MenuItem miAbout;
        java.awt.MenuItem miHelp;
        //}}
        SearchEngine searchEngine;
        SearchPanel searchPanel;
        ResultPanel resultPanel;
         int numberOfPages;
MainGUI(SearchEngine searchengine, Frame frame, String title)
        {
                super(title);
                searchEngine = searchengine;

                //{{INIT_CONTROLS
                setLayout(null);
```

```
            setVisible(false);
            setSize(600,500);
            setBackground(new Color(0,128,192));
                searchPanel = new SearchPanel(this);
            searchPanel.setLayout(null);
                searchPanel.setBounds(30,60,530,91);
                 searchPanel.setBackground(new Color(234,219,112));
    add(searchPanel);
            resultPanel = new ResultPanel(this);
            resultPanel.setLayout(null);
            resultPanel.setBounds(30,170,530,200);
              resultPanel.setBackground(new Color(234,219,112));
            add(resultPanel);
        head_Label = new java.awt.Label("I Can Understand  Natural
Language",Label.CENTER);
            head_Label.setBounds(30,10,530,37);
            head_Label.setFont(new Font("Dialog", Font.PLAIN, 30));
            head_Label.setBackground(new Color(234,219,112));
            add(head_Label);
            //{{INIT_MENUS
            mainMenuBar = new java.awt.MenuBar();
            menu1 = new java.awt.Menu("File");
            miExit = new java.awt.MenuItem("Exit");
```

```
            menu1.add(miExit);
            mainMenuBar.add(menu1);
            menu3 = new java.awt.Menu("Help");
            mainMenuBar.setHelpMenu(menu3);
            miAbout = new java.awt.MenuItem("About..");
            miHelp = new java.awt.MenuItem("Instructions");
            menu3.add(miAbout);
            menu3.add(miHelp);
            mainMenuBar.add(menu3);
            setMenuBar(mainMenuBar);
            //{{REGISTER_LISTENERS
            SymWindow aSymWindow = new SymWindow();
            this.addWindowListener(aSymWindow);
            SymAction lSymAction = new SymAction();
            miAbout.addActionListener(lSymAction);
        miHelp.addActionListener(lSymAction);
            miExit.addActionListener(lSymAction);
            //}}
        }

    public void setVisible(boolean b)
        {
            if(b)
```

```
            {
                setLocation(50, 50);
            }
            super.setVisible(b);
        }


        public void addNotify()
        {
            // Record the size of the window prior to calling parents addNotify.
            Dimension d = getSize();

            super.addNotify();

            if (fComponentsAdjusted)
                return;

            // Adjust components according to the insets
            setSize(insets().left + insets().right + d.width, insets().top + insets().bottom
+ d.height);
            Component components[] = getComponents();
            for (int i = 0; i < components.length; i++)
            {
```

```
                Point p = components[i].getLocation();
                p.translate(insets().left, insets().top);
                components[i].setLocation(p);
            }
            fComponentsAdjusted = true;
    }

    class SymWindow extends java.awt.event.WindowAdapter
    {
        public void windowClosing(java.awt.event.WindowEvent event)
        {
            Object object = event.getSource();
            if (object == MainGUI.this)
                MainGUI_WindowClosing(event);
        }
    }

    void MainGUI_WindowClosing(java.awt.event.WindowEvent event)
    {
        setVisible(false);      // hide the Frame
        dispose();              // free the system resources
        System.exit(0);         // close the application
    }
```

```
class SymAction implements java.awt.event.ActionListener
{
        public void actionPerformed(java.awt.event.ActionEvent event)
        {
                Object object = event.getSource();
                 if (object == miAbout)
                        miAbout_Action(event);
                else if (object == miExit)
                        miExit_Action(event);
                else if (object == miHelp)
                        miHelp_Action(event);
        }
}

void miAbout_Action(java.awt.event.ActionEvent event)
{
        //{{CONNECTION
        // Action from About Create and show as modal
        (new AboutDialog(this, true)).setVisible(true);
        //}}
}

        void miHelp_Action(java.awt.event.ActionEvent event)
```

```
        {
                //{{CONNECTION
                // Action from Help Create and show as modal
                (new HelpDialog(this, true)).setVisible(true);
                //}}
        }



        void miExit_Action(java.awt.event.ActionEvent event)
        {
                //{{CONNECTION
                // Action from Exit Create and show as modal
                (new QuitDialog(this, true)).setVisible(true);
                //}}
        }


        void miOpen_Action(java.awt.event.ActionEvent event)
        {
                //{{CONNECTION
                // Action from Open... Show the OpenFileDialog
                int             defMode                 =
openFileDialog1.getMode();
                String  defTitle                = openFileDialog1.getTitle();
```

```
                String defDirectory    = openFileDialog1.getDirectory();
                String defFile                 = openFileDialog1.getFile();

                openFileDialog1 = new java.awt.FileDialog(this, defTitle, defMode);
                openFileDialog1.setDirectory(defDirectory);
                openFileDialog1.setFile(defFile);
                openFileDialog1.setVisible(true);
                }}
}       }
        /**
 * This file "ResultPanel.java"  provides  the  necessary
 *  methods and attributes for  the the resultpanel class.
 *
 * Author:       Salah Ali
 * Version:      2.4  Sep 13   1999
 *
 *
import java.awt.*;
import java.io.*;
import java.util.*;
import java.awt.event.*;
import java.awt.*;
import Search;
```

```
**
* The class take the request from the user and send it to the SearchForText class
*  also it clear  the results list and the number of results from the previouse search
*
* @version 4.3 sep 13 1999
* @author  Salah Ali
**

public class SearchPanel extends Panel implements  ActionListener
{

int results;
        MainGUI mainGUI;
        Search search;
        ResultPanel resultPanel;
        TextArea req_area ;
        Button searchButton, clearButton;
        Label search_label,lab_help1 ,lab_help2 ,lab_help3 ;
        SearchPanel(MainGUI myframe) {
        mainGUI = myframe;
        req_area = new TextArea();
        req_area.setBounds(32,15,301,85);
```

```
        req_area.setFont(new Font("Dialog", Font.PLAIN, 14));
        add(req_area);
        searchButton = new java.awt.Button();
        searchButton.setLabel("Search");
        searchButton.setBounds(382,76,112,32);
        searchButton.setBackground(new Color(12632256));
        searchButton.addActionListener(this);
                add(searchButton);
        clearButton = new java.awt.Button();
        clearButton.setLabel("Clear");
        clearButton.setBounds(526,74,119,32);
        clearButton.setBackground(new Color(12632256));
        clearButton.addActionListener(this);
                add(clearButton);
        lab_help1 = new java.awt.Label("  Enter Your question in your left side area
Using natural  " );
        lab_help1.setBounds(344,7,310,21);
        lab_help1.setForeground(new Color(255,255,128));
                add(lab_help1);
        lab_help2 = new java.awt.Label("  Language Or if  you have a microphone speak
with the");
        lab_help2.setBounds(344,25,310,20);
        lab_help2.setForeground(new Color(255,255,128));
```

```
                add(lab_help2);
        lab_help3 = new java.awt.Label("  search engine directly then press the search
button.  ");
        lab_help3.setBounds(344,44,300,23);
        lab_help3.setForeground(new Color(255,255,128));
        add(lab_help3);
        }
        // This method is responsible for reading
        // the request from the user e.g., "which
        // courses were offered in fall 1999".
        public String getSearchText() {
                return req_area.getText();
        }
// When actionPerformed()  is called,
// the  "clear" or "search" button is pressed
  // if clear button is pressed, all the text area
    will be cleared.
        public void actionPerformed(ActionEvent e)
        {
      if (e.getSource() == clearButton) {
        //  Clear every thing in the main user interface.
        // results list area, description area
      req_area.setText("");
```

```
        mainGUI.resultPanel.clearList();
        mainGUI.resultPanel.descrp.setText("");
        mainGUI.resultPanel.description_Label.setText("");
              mainGUI.resultPanel.textArea1.setText("");
        results=0;
        mainGUI.resultPanel.numOfResults.setText("You have   "+ (results) + "   Results"
);
          return ;
        }
                  if (e1.getSource() == searchButton) {
   if the user did not enter or say the request
   AIVDISE will not make any search
                        if (req_area.getText().length() == 0) {
                              return ;
                        }
                        If the user presses the search button, we send the
                  //          request and home page address to the  Search class
                    mainGUI.resultPanel.descrp.setText("");
                                                      mainGUI.resultPanel.index=0;

          mainGUI.resultPanel.clearList();
                                          mainGUI.resultPanel.descrption.clear();
                                             search.pageDatabase.clear();
```

```
mainGUI.resultPanel.description_Label.setText("");
                                mainGUI.resultPanel.textArea1.setText("");

        mainGUI.numberOfPages = 0;

        Search search;
                        search = new Search(mainGUI.
mainGUI.searchEngine.home_address, getSearchText().toLowerCase());
            }
              return ;
        }
} **
 * This file "Search.java" provides the all necessary methods
 * and attributes for  the Search class.
 *
 * Author:      Salah
 * Version:     2.4  Oct 20 1999
 *
 */
import java.awt.*;
import java.net.*;
import java.io.*;
```

```
import java.util.*;
 **
 * The class does the main search search for the urls and
 * remove the deplicate pages send the result to the resultPanel
 *
 * @version 9.2 Jan 15 2000
 * @author Salah
 **/
public class Search extends Thread
{
        // The maximum number of simultaneous threads
        static Hashtable pageDatabase = new Hashtable();
        String subString;
        URL pageToSearch;
        MainGUI mainGUI;
        String textField; // search for a string
        Vector extractString = new Vector();
            StringBuffer buf1 = new StringBuffer();

        // Constructor
        Search(MainGUI myFrame, String page, String string)
        {
                mainGUI = myFrame;
```

```
            try {
                    pageToSearch = new URL(page);
                    setName(page);
                    start();
            } catch (MalformedURLException e) {
                    this.stop();
            }
            textField = new String(string);

                    StringTokenizer st = new StringTokenizer(textField);
                    extractString.ensureCapacity(st.countTokens());
                    while (st.hasMoreTokens()) {
                            extractString.addElement((String) st.nextToken());
                    }
    }
    public void run()
    {
            mainGUI.setCursor(Frame.WAIT_CURSOR);
            mainGUI.searchPanel.searchButton.disable();
    mainGUI.numberOfPages++;
    if (mainGUI.numberOfPages > mainGUI.searchEngine.totalPages) {
            this.stop();
            }
```

```
String contentsOfPage;        // The entire page is stored in this string
Vector linksInPage;           // The parsed links in a page are stored here
boolean match;
contentsOfPage = getPage(pageToSearch);
// Grab the links in the page to crawl
linksInPage = extractLinks(contentsOfPage);
match = found(contentsOfPage, extractString);
// add the result to the list of results in resultPanel
if (match) {
        String deesription_page;
deesription_page = des_area(contentsOfPage))

    mainGUI.resultPanel.addResult(pageToSearch.toString(),pageToSearch.toString(
));
    mainGUI.resultPanel.putDescrp(deesription_page + "\n" +  " these words AIVDISE
Searches for:  " );
}
Enumeration enumLinks = linksInPage.elements();
while(enumLinks.hasMoreElements()) {
        String nextPage = (String) enumLinks.nextElement();
        if (! alreadyVisited(nextPage)) {
        markAsVisted(nextPage);
                new Search(mainGUI, nextPage, textField);
```

```
                }
            }

            try {
                Thread.sleep( (int) (Math.random()*200));
            } catch (Exception e) {
            }
        mainGUI.searchPanel.searchButton.enable();
            mainGUI.setCursor(Frame.DEFAULT_CURSOR);
        }
        protected String getPage (URL acadia)
        {
            InputStream conn = null;
    DataInputStream data = null;
    String line;
    StringBuffer buf = new StringBuffer();

            try {
                conn = acadia.openStream();
                data = new DataInputStream(conn);
                while ((line=data.readLine()) != null) {
                    buf.append(line);
                }
```

```
        } catch (IOException e) {
                return "";
        }
        return buf.toString();
}
// chick f a page has already been encountred
protected boolean alreadyVisited(String page)
{
                return pageDatabase.containsKey(page);
}

// Method to mark a page as visited
protected void markAsVisted(String page)
{
                mainGUI.searchEngine.pageDatabase.put(page, page);

// add page to the data base

pageDatabase.put(page, page);
}
// all the extractString have to be searched
// this is the main method that can understand the natural language.
protected boolean found(String content, Vector subStrs)
```

```
        {
        String[] wordIgnoreMe = {"who","is" ,"where","i","show","what","the",
                                 "give" , "can", "me", "all","find" ,"list","a",
                    "of", "are" ,"information","about","were","was",
                                              "in","dr.","by","are","who's","which"
,"offered",
                                            "doctor","need","some" ,  "want"
,"search","could","please"
                                                    ,"tell","you","does","and"} ;
                Enumeration enumSubstrings = subStrs.elements();
                                while(enumSubstrings.hasMoreElements()) {
                        String subString = (String) enumSubstrings.nextElement();
                for( int i=0; i< 38; i++){
                                        if (subString.equals(wordIgnoreMe[i])){
                        subString =  (String) enumSubstrings.nextElement();
        i=0;
                                                }
                }
  if (subString.equals("courses")){
        subString="instructors";

         if (content.indexOf(subString) == -1) {
                subString="professor";
```

```
            }
    if (content.indexOf(subString) == -1) {
            subString="instructor";
      }
      if (content.indexOf(subString) == -1) {
            subString="professor";
            }
      if (content.indexOf(subString) == -1) {
            subString="instructors";
            }
       if (content.indexOf(subString) == -1) {
            subString="professors";
            }
              if (content.indexOf(subString) == -1) {
            subString="professors";
                  }
  }
              if (subString.equals("professors")
                  ||subString.equals("instructors")
                        ||subString.equals("professor")        ){
     subString="instructors";
   if (content.indexOf(subString) == -1) {
```

```
                subString="instructor";}
            if (content.indexOf(subString) == -1) {
                subString="professor";}
             if (content.indexOf(subString) == -1) {
                subString="professors";}



if((subString.equals("fall"))){
   if (content.indexOf(subString) == -1) {
      subString="september";
      }
if((subString.equals("teaching"))
      ||(subString.equals("teach"))
      ||(subString.equals("tought"))
      ||(subString.equals("teaches")) ){
                                    subString = "professor";
                                if (content.indexOf(subString) == -1) {
      subString="instructor";
   }
                        if (content.indexOf(subString) == -1) {
      subString="professors";
   }
```

```
                    if (content.indexOf(subString) == -1) {
        subString="instructors";
    }

                        }
                    if((subString.equals("director"))){
                            subString = "Faculty at the Jodrey School of
Computer Science";
                    subString = (String) enumSubstrings.nextElement();
                        }
00                      }
        if(subString.equals("to")){
                                    subString = "2"; }
                    if((subString.equals("trudel"))){
                        subString = "Course Outline";
                subString = (String) enumSubstrings.nextElement();

if((subString.equals("teaches"))||(subString.equals("teaching"))
                                    ||(subString.equals("tought"))    ){
            subString= "CAR 310";
        }
```

```
        }

                if((subString.equals("qiu"))|| (subString.equals("q."))){

                        subString = "course outline";

        subString = (String) enumSubstrings.nextElement();


if((subString.equals("teaches"))||(subString.equals("teaching"))

                                                ||(subString.equals("tought"))){

                                subString = "CAR 403";


                                        }

                        }


                if((subString.equals("giles"))){

                                subString = "course outline";

        subString = (String) enumSubstrings.nextElement();


if((subString.equals("teaches"))||(subString.equals("teaching"))

                                                ||(subString.equals("tought"))){

                                                subString = "CAR 416";

                                        }

                        }

                                if((subString.equals("oliver"))){

                                subString = "Carnegie 313";
```

```
                                    if (content.indexOf(subString) == -1) {
                subString="CAR 313";
    }
        subString = (String) enumSubstrings.nextElement();

if((subString.equals("teaches"))||(subString.equals("teaching"))
                                                ||(subString.equals("tought"))){
                                subString = "Course Outline";
                                }
                                 if((subString.equals("first"))){
                                                subString = "1st";
                                                subString = (String)
enumSubstrings.nextElement();
                                                if (( subString.equals("year"))){

                                                }
                }
    if((subString.equals("second"))){
                                        subString = "2nd";
                                subString = (String) enumSubstrings.nextElement();
          if (( subString.equals("year"))){
                                 subString = (String) enumSubstrings.nextElement();
          }
```

```
if((subString.equals("third"))){
                    subString = "3rd";
                     subString = (String) enumSubstrings.nextElement();
          if (( subString.equals("year"))){

                }
       }
if((subString.equals("fourth"))){
                                    subString = "4th";
                                    subString = (String) enumSubstrings.nextElement();
          if (( subString.equals("year"))){

                                  subString = (String) enumSubstrings.nextElement();

                                }
       }



  // third year course
if((subString.equals("3rd"))){
                                    subString = "3rd year";
                                        subString3 = "3rd";
```

```
                    subString = (String) enumSubstrings.nextElement();

            if (( subString.equals("year"))){
                    subString="comp";
                        subString4="year";
                subString = (String) enumSubstrings.nextElement();
                            if ((subString.equals("courses"))){
                                subString="3703";

        subString5="courses";
                    }
                }
        }
            if (content.indexOf(subString) == -1) {
                    return false;
                }
            return true;
        }
public String des_area (String contents)
        {
            StringBuffer buf3 = new StringBuffer ("");
            boolean foundTag = false;
```

```
                for (int i=0; i < contents.length(); i++)
                {
                        if (contents.charAt (i) == '<')
                                foundTag = true;
                        else if (contents.charAt(i) == '>')
                                foundTag = false;
                        else if (foundTag == false)
                                buf3.append (contents.charAt(i));
                }
                return (buf3.toString());
        }
        protected Vector extractLinks (String content)
        {
                int marker = 0;
                int frameMarker = 0;
                int endOfLink;
                String buff = new String(content.toUpperCase());
                String link;
                Vector bufferOfLinks = new Vector();          // Storage for found links
                while (marker != -1) {
marker = buff.indexOf("HREF=", marker);
                        if (marker != -1) {
                                marker += 5;
```

```
                    endOfLink = content.indexOf(">",marker+1);
                link = content.substring(marker, endOfLink);
                    link = massageLink(link);
                        if (link.endsWith(".html") || link.endsWith(".htm")) {
                            String fullPath = new
String(mainGUI.searchEngine.server + link);
                            bufferOfLinks.addElement(fullPath);
                        }
                        marker ++;
                    }
                }
                return bufferOfLinks;
        }
        protected String massageLink(String link)
        {
            if (link.startsWith("\") ) {
                link = link.substring(1,link.length());
            }
            if (link.endsWith("\") ) {
                link = link.substring(0,link.length() -1);
            }
            if (link.indexOf("://") == -1) {
                return link;
```

```
                } else {
                        return "";
                }
        }
}
**
* This file "ResultPanel.java" provides the necessary
*  methods and attributes for the the resultpanel class.
*
* Author:       Salah Ali
* Version:      2.4 Oct 25 1999
*
*/
 import java.util.Date;
import java.awt.*;
import java.net.*;
import java.util.Vector;
import java.util.Hashtable;

import java.awt.event.*;

**
* The class represents the results of the search
```

```
* all the results will be displyed in a list
* also it show the number of results that presented in the list
*
* @version 2.4 Jan 13 2000
* @author Salah Ali
**/

public class ResultPanel extends Panel implements ItemListener, ActionListener
{

        int show1=0;
String num_OfResults;
        MainGUI mainGUI;
        List resultsList;
        Vector urls = new Vector();
        TextArea desc_area;
        /** table that maps users objects to a hashtable of Users **/
 public Hashtable descrption = new Hashtable();

 int index = 0;
        Label numOfResults, descrp;
        Label description_Label;
        int num_of_results = 0;
```

```
Label label_Description;

        ResultPanel(MainGUI myFrame)

        {

        Label help_1;

        Label help_2;

                                mainGUI = myFrame;

                urls.ensureCapacity(myFrame.searchEngine.totalPages);


        resultsList = new java.awt.List(0);

        //

        resultsList.addItemListener(this);

        resultsList.addActionListener(this);


                add(resultsList);

                resultsList.setBounds(20,46,650,142);

                resultsList.setBackground(new Color(255,255,255));


                resultsList.setFont(new Font("Dialog", Font.BOLD, 14));

                resultsList.setForeground(new Color(255));


                help_1 = new java.awt.Label("This is what AIVDISE found one click by
your mouse shows the description of the file double click you will see the actual page.
",Label.CENTER);
```

```
            help_1.setBounds(-7,10,711,17);
          help_1.setForeground(new Color(255,255,128));
        add(help_1);
            description_Label = new java.awt.Label("",Label.RIGHT);
            description_Label.setBounds(38,215,244,24);
            description_Label.setFont(new Font("Dialog", Font.BOLD|Font.ITALIC,
20));
                        description_Label.setForeground(new Color(213,43,59));//
red

            add(description_Label);
            numOfResults = new java.awt.Label("You have  :",Label.CENTER);
            numOfResults.setBounds(288,274,320,27);
            numOfResults.setFont(new Font("Dialog", Font.BOLD, 14));
            numOfResults.setForeground(new Color(255,255,0));
            add(numOfResults);
            desc_area = new java.awt.TextArea();
            desc_area.setBounds(285,200,350,70);
             desc_area.setFont(new Font("Dialog",Font.PLAIN, 12));
            desc_area.setForeground(new Color(0));
                              desc_area.setForeground(new Color(213,43,59));//
red

            add(desc_area);
```

```
        descrp = new java.awt.Label("");
                descrp.setBounds(10,230,690,28);
                descrp.setForeground(new Color(255,255,0));
                descrp.setFont(new Font("Dialog", Font.BOLD, 16));
                add(descrp);
        }
        // This method is invoked when a new search
         is initiated or when the 'Clear' buton is
         clicked. It clears all the addresses from the
        result list.      public void clearList()
        {
                urls.removeAllElements();
                resultsList.clear();
        }
  This method is responsible for adding the
        results to the result list.  This method
        is invoked when a search is found and it
                increases the number of results by one.
                public void addResult(String match, String url)
        {
                                                descrption.put(new Integer(-1), " ");


                num_of_results ++;
```

```
                urls.addElement(new String(url));
                resultsList.addItem(match);
        numOfResults.setText("You have  :"                + (num_of_results)
        + "  Result(s)  "  );
        mainGUI.searchPanel.clearButton.enable();
        }
  This method is responsible for the
  description area. For example, it shows the
  description of each file when it is highlighted.
 public void itemStateChanged(ItemEvent e){
  String[] showMe = {"This is will be in ","You will find it in","Are you looking for","Or
you are looking for"
                                                                    ,"You
may looking for"};
  int index = resultsList.getSelectedIndex();
          String str = (String)descrption.get(new Integer(index));
    if (show1 > 4) show1 =0;
  description_Label.setText(showMe[show1] +" ");
  int len = str.length()/2;
  if(len > 100) {
        String line1 = str.substring(0,61);
        String line2 = str.substring(61,121);
                String line3 = str.substring(121,181 );
```

```
                String line4 = str.substring(181,241 );
  desc_area.setText(" "+ line1 + "-" +"\n");
  desc_area.append(" "+line2+ "-" +"\n");
    desc_area.append(" "+line3+ "-" +"\n");
    desc_area.append("  "+line4 +".....");
        show1=show1+1;
  }
  else {
                String line5 = str.substring(0,len);
       desc_area.append(" "+line6 +".....");
    show1=show1+1;}
 }
 public void putDescrp(String str){
        descrption.put(new Integer(index), str);
        index++;
}
  The actionPerformed() method is called
  when the user double clicks on an address
// the and the actual page is shown.
        public void actionPerformed(ActionEvent e1)
        {
                int listIndex = resultsList.getSelectedIndex();
                        URL goal;
```

```
                    try {
                                        goal = new URL((String)
urls.elementAt(listIndex));

        mainGUI.searchEngine.getAppletContext().showDocument(goal, "Results");
                                } catch (MalformedURLException badurl) {
                                }
                return;
        }
} end of class

**
* This file "QuitDialog.java" provides the necessary
*  methods and attributes for the the QuitDialog class.
*
* Author:       Salah Ali
* Version:      2.4 june 23     1999
*
*/
import java.awt.*;
import java.awt.event.*;
```

```
**
*       the QuitDialog  class contains all the elements
*       necessary to quit or stay in the main window of the search engine .
*
* @version  1.1 june  21 1999
* @author   Salah Ali
**



public class QuitDialog extends Dialog
{
        public QuitDialog(Frame parent, boolean modal)
        {
                super(parent, modal);


                setLayout(null);
                setVisible(false);
                setSize(337,135);
                yesButton = new java.awt.Button();
                yesButton.setLabel(" Yes ");
                yesButton.setBounds(72,80,79,22);
```

```
            yesButton.setFont(new Font("Dialog", Font.BOLD, 12));
            add(yesButton);
            noButton = new java.awt.Button();
            noButton.setLabel("  No  ");
            noButton.setBounds(185,80,79,22);
            noButton.setFont(new Font("Dialog", Font.BOLD, 12));
            add(noButton);
            label1 = new java.awt.Label("Do you really want to
quit?",Label.CENTER);
            label1.setBounds(78,33,180,23);
            add(label1);
            setTitle("Search Engine - Quit");
            //}}

            //{{REGISTER_LISTENERS
            SymWindow aSymWindow = new SymWindow();
            this.addWindowListener(aSymWindow);
            SymAction lSymAction = new SymAction();
            noButton.addActionListener(lSymAction);
            yesButton.addActionListener(lSymAction);
            //}}
        }
```

```
    public void addNotify()
    {
        // Record the size of the window prior to calling parents addNotify.
        Dimension d = getSize();

        super.addNotify();

        if (fComponentsAdjusted)
            return;

        // Adjust components according to the insets
        setSize(insets().left + insets().right + d.width, insets().top + insets().bottom
+ d.height);
        Component components[] = getComponents();
        for (int i = 0; i < components.length; i++)
        {
            Point p = components[i].getLocation();
            p.translate(insets().left, insets().top);
            components[i].setLocation(p);
        }
        fComponentsAdjusted = true;
    }
```

```
    public QuitDialog(Frame parent, String title, boolean modal)
    {
        this(parent, modal);
        setTitle(title);
    }
 Shows or hides the component depending on the boolean flag b.
 param b  if true, show the component; otherwise, hide the component.
 see java.awt.Component#isVisible

public void setVisible(boolean b)
    {
        if(b)
        {
            Rectangle bounds = getParent().getBounds();
            Rectangle abounds = getBounds();

            setLocation(bounds.x + (bounds.width - abounds.width)/ 2,
                    bounds.y + (bounds.height - abounds.height)/2);
        }
        super.setVisible(b);
    }

 Used for addNotify check.
```

```
boolean fComponentsAdjusted = false;

//{{DECLARE_CONTROLS
java.awt.Button yesButton;
java.awt.Button noButton;
java.awt.Label label1;
//}}

class SymWindow extends java.awt.event.WindowAdapter
{
    public void windowClosing(java.awt.event.WindowEvent event)
    {
        Object object = event.getSource();
        if (object == QuitDialog.this)
            QuitDialog_WindowClosing(event);
    }
}

void QuitDialog_WindowClosing(java.awt.event.WindowEvent event)
{
dispose();
}
```

```
class SymAction implements java.awt.event.ActionListener
{
    public void actionPerformed(java.awt.event.ActionEvent event)
    {
        Object object = event.getSource();
        if (object == noButton)
            noButton_Clicked(event);
        else if (object == yesButton)
            yesButton_Clicked(event);
    }
}

void yesButton_Clicked(java.awt.event.ActionEvent event)
{
    Toolkit.getDefaultToolkit().getSystemEventQueue().postEvent(new
WindowEvent((java.awt.Window)getParent(), WindowEvent.WINDOW_CLOSING));
}

void noButton_Clicked(java.awt.event.ActionEvent event)
{
  dispose();
}
```

```
/
**
* This file "helpDialog.java"  provides  the  necessary
*  methods and attributes for  the the helpDialog class.
*
* Author:       Salah Ali
* Version:      2.4  April 17      1999
*
*
```

```
import java.awt.*;
```

```
**
*         the helpDialog  class  contains all the elements
*         necessary to present  an information for how to use the search engine.
*
* @version  2.3  April 17 1999
* @author   Salah Ali
**/
```

```
public class HelpDialog extends Dialog {

    public HelpDialog(Frame parent, boolean modal)
    {
        super(parent, modal);

        //{{INIT_CONTROLS
        setLayout(null);
        setVisible(false);
        setSize(500,300);
        panel1 = new java.awt.Panel();
        panel1.setLayout(null);
        panel1.setBounds(40,10,430,275);
        add(panel1);
        okButton = new java.awt.Button();
        okButton.setLabel("OK");
        okButton.setBounds(190,230,66,27);
        panel1.add(okButton);
        list1 = new java.awt.List(0);

list1.addItem("Using AIVDISE");
list1.addItem("");
```

```
list1.addItem("The first thing that we have to do to start working with AIVDISE ");
list1.addItem("is in the your browser write the address of your homepage followed");
list1.addItem("by the filename INVDISE.hmtl and press enter then press the Open ");
list1.addItem("and the system opens the main user interface. Then, the user can ");
list1.addItem("interact with the search engine.");
list1.addItem("");
list1.addItem("Main user interface Once the user presses the open button, the system ");
list1.addItem("opens the main user interface. ");
list1.addItem("");
list1.addItem("User requests");
list1.addItem("");
list1.addItem("The User request section is responsible for the requests from");
list1.addItem("the user and sends the commands to the search engine. ");
list1.addItem("");
list1.addItem("File drop menu");
list1.addItem("File is a drop menu that lets the user exit from AIVDISE.");
list1.addItem("The File drop menu contains one menu item which is exit menu item .");
list1.addItem("When the user finishes working with AIVDISE, the user can click on File.
AIVDISE will open the drop menu for the user. Then the user can click on exit and
AIVDISE will open an option window to exit or go back to the main user interface . ");
list1.addItem("");
list1.addItem("Help drop menu");
list1.addItem("");
```

```
list1.addItem("");
list1.addItem("The Help drop menu contains the About menu item and the Instructions");
list1.addItem("menu item . ");
list1.addItem("");
list1.addItem("Instructions Menu Item");
list1.addItem("");
list1.addItem("Help is a drop menu that lets the user get help from AIVDISE.");
list1.addItem("Clicking on Instructions opens  the window . It will guide the");
list1.addItem("user in the use of AIVDISE. ");
list1.addItem("");
list1.addItem("About Item Menu");
list1.addItem("");
list1.addItem("This menu item shows information about search engine implementation.
");
list1.addItem("");
list1.addItem("Text area");
list1.addItem("");
list1.addItem("The text area is responsible for accepting the request from the user ");
list1.addItem("in either typed or spoken English. ");
list1.addItem("");
list1.addItem("Search button");
list1.addItem("");
list1.addItem("By pressing Search button, AIVDISE accepts the question from the ");
```

```
list1.addItem("text area and starts searching. ");
list1.addItem("");
list1.addItem("");
list1.addItem("Clear button");
list1.addItem("");
list1.addItem("When the user presses the clear button, AIVDISE clears any information
");
list1.addItem("in the User requests, Results, and Description sections. Then the user can
");
list1.addItem("make a new search.  ");
list1.addItem("");
list1.addItem("Results area");
list1.addItem("");
list1.addItem("In this section, the user can see the results of the search. In this area,");
list1.addItem("only the address of the page found by AIVDISE .We do not see the actual
page. ");
list1.addItem("the user can access the page by double clicking on that address in the
result area. ");
list1.addItem(" ");
list1.addItem("Description");
list1.addItem("");
list1.addItem("This section endeavors to assist the user by showing a simple ");
```

```
list1.addItem("description of each page found by AIVDISE. The user can click on the
address in the");
list1.addItem("result area and AIVDISE will show the description in the description
area.");
list1.addItem("");
panel1.add(list1);
        list1.setBounds(70,20,300,200);
        setTitle("Help Menu");
        //}}

        //{{REGISTER_LISTENERS
        SymWindow aSymWindow = new SymWindow();
        this.addWindowListener(aSymWindow);
        SymAction lSymAction = new SymAction();
        okButton.addActionListener(lSymAction);
        //}}

    }

    public HelpDialog(Frame parent, String title, boolean modal)
    {
        this(parent, modal);
        setTitle(title);
```

```
    }

    public void addNotify()
    {
        // Record the size of the window prior to calling parents addNotify.
      Dimension d = getSize();

        super.addNotify();

        // Only do this once.
        if (fComponentsAdjusted)
            return;

        // Adjust components according to the insets
        setSize(insets().left + insets().right + d.width, insets().top + insets().bottom
- d.height);
        Component components[] = getComponents();
        for (int i = 0; i < components.length; i++)
        {
            Point p = components[i].getLocation();
            p.translate(insets().left, insets().top);
            components[i].setLocation(p);
        }
```

```
        // Used for addNotify check.
        fComponentsAdjusted = true;
    }

    public void setVisible(boolean b)
    {
      if(b)
      {
        Rectangle bounds = getParent().bounds();
        Rectangle abounds = bounds();

        move(bounds.x + (bounds.width - abounds.width)/ 2,
             bounds.y + (bounds.height - abounds.height)/2);
      }

        super.setVisible(b);
    }

    //{{DECLARE_CONTROLS
    java.awt.Panel panel1;
    java.awt.Button okButton;
    java.awt.List list1;
```

```
    //}}

// Used for addNotify check.
    boolean fComponentsAdjusted = false;

    class SymWindow extends java.awt.event.WindowAdapter
    {
        public void windowClosing(java.awt.event.WindowEvent event)
        {
            Object object = event.getSource();
            if (object == HelpDialog.this)
                HelpDialog_WindowClosing(event);
        }
    }

    void HelpDialog_WindowClosing(java.awt.event.WindowEvent event)
    {
      dispose();
    }

    class SymAction implements java.awt.event.ActionListener
    {
        public void actionPerformed(java.awt.event.ActionEvent event)
```

```
            {
                Object object = event.getSource();
                if (object == okButton)
                    okButton_Clicked(event);
            }
        }

        void okButton_Clicked(java.awt.event.ActionEvent event)
        {
            //{{CONNECTION
            // Clicked from okButton Hide the Dialog
            dispose();
            //}}
        }
}
```

AIVDISE.html

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-
1252">
<META NAME="Generator" CONTENT="Microsoft Word 97">
<TITLE> </TITLE>
```

```
<META NAME="Template" CONTENT="C:\PROGRAM FILES\MICROSOFT
OFFICE\OFFICE\html.dot">
</HEAD>
<BODY TEXT="#000000" LINK="#0000ff" VLINK="#ff0000"
BACKGROUND="Image1.jpg" alink="#000088">

<P ALIGN="CENTER"><IMG SRC="Image4.gif" WIDTH=45 HEIGHT=52><BR>
  <BR>
  </P>
<P ALIGN="CENTER"><IMG SRC="rainbow_thinline.gif" WIDTH=540
HEIGHT=4></P>
<P ALIGN="CENTER"><blink><B><I><FONT SIZE=6
COLOR="#808080">WelCome To Salah's Homepage</blink></B></I></FONT> </P>
<P ALIGN="CENTER"><IMG SRC="rainbow_thinline.gif" WIDTH=540
HEIGHT=4></P>
<P ALIGN="CENTER"><IMG SRC="fish1.gif" WIDTH=365 HEIGHT=100> 
</P>
<I><FONT SIZE=4><P ALIGN="CENTER">Hi My Name is </I></FONT><A
HREF="images/salah.htm"><I><FONT SIZE=4>Salah</I></FONT></A><I><FONT
SIZE=4> Ali I was born in Libya.</P>
</I></FONT><P ALIGN="CENTER">This is a test of AIVDISE</P>
<P> </P>
<P ALIGN="CENTER"> </P>
```

<P ALIGN="CENTER">Search My Homepage </P>

<P ALIGN="CENTER"><applet code="SearchEngine.class" align="baseline"

width="150" height="50"><param name="width1" value="730"><param name="high1"

value="500"><param name="totalPages" value="200"><param name="server"

value="http://131.162.166.112/"><param name="homepage"

value="index20.html"><param name="searchingStyle" value="true"></applet></P>

<P ALIGN="CENTER"> </P>

<FONT SIZE=4><P ALIGN="CENTER">Under Construction</FONT>

  </P>

<P ALIGN="CENTER"><A HREF="salah34@hotmail.com"><IMG SRC="Mail.gif"

BORDER=0 WIDTH=45 HEIGHT=52></A></P>

<P ALIGN="CENTER"> </P></BODY>

</HTML>