HOW SEARCH ENGINES WORK

AND A WEB CRAWLER APPLICATION

Monica Peshave
Department of Computer Science
University of Illinois at Springfield
Springfield, IL 62703
mpesh01s@uis.edu

Advisor: Kamyar Dezhgosha
University of Illinois at Springfield
One University Plaza, MS HSB137
Springfield, IL 62703-5407
kdezh1@uis.edu

Abstract

The main purpose of this project is to present the anatomy of a large scale Hypertext Transfer Protocol (HTTP) based Web search engine by using the system architecture of large search engines such as Google, Yahoo as a prototype. Additionally, a web crawler is developed and implemented in Java v1.4.2, which demonstrates the operation of a typical Web crawler.

The paper describes in detail the basic tasks a search engine performs. An overview of how the whole system of a search engine works is provided. A WebCrawler application is implemented using Java programming language. The GUI of the developed application helps the user to identify various actions that can take place like specifying the start URL, maximum URLs to be crawled, the way crawling has to be done – breadth first or depth first. This paper also lists proposed functionalities as well as features not supported by the web crawler application.

Key Words – Spider, Indexer, Repository, Lexicon, Robot Protocol, Document indexer, Hit list.

1 Introduction

Engineering a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been conducted on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. There are differences in the ways various search engines work, but they all perform three basic tasks:

1. They search the Internet or select pieces of the Internet based on important words.
2. They keep an index of the words they find, and where they find them.
3. They allow users to look for words or combinations of words found in that index.

A search engine finds information for its database by accepting listings sent in by authors who want exposure, or by getting the information from their "web crawlers," "spiders," or "robots," programs that roam the Internet storing links to and information about each page they visit. A web crawler is a program that downloads and stores Web pages, often for a Web search engine. Roughly, a crawler starts off by placing an initial set of URLs, $S0$, in a queue, where all URLs to be retrieved are kept and prioritized. From this queue, the crawler gets a URL (in some order), downloads the page, extracts any URLs in the downloaded page, and puts the new URLs in the queue. This process is repeated until the crawler decides to stop. Collected pages are later used for other applications, such as a Web search engine or a Web cache.

The most important measure for a search engine is the search performance, quality of the results and ability to crawl, and index the web efficiently. The primary goal is to provide high quality search results over a rapidly growing World Wide Web. Some of the efficient and recommended search engines are Google, Yahoo and Teoma, which share some common features and are standardized to some extent.

1.1 Why the Web is so popular now?

Commercial developers noticed the potential of the web as a communications and marketing tool when graphical Web browsers broke onto the Internet scene (Mosaic, the precursor to Netscape Navigator, was the first popular web browser) making the Internet, and specifically the Web, "user friendly." As more sites were developed, the more popular the browser became as an interface for the Web, which spurred more Web use, more Web development etc. Now graphical web browsers are powerful, easy and fun to use and incorporate many "extra" features such as news and mail readers. The nature of the Web itself invites user interaction; web sites are composed of hypertext documents, which mean they are linked to one another. The user can choose his/her own path by selecting predefined "links". Since hypertext documents are not organized in an arrangement which requires the user to access the pages sequentially, users really like the ability to choose what they will see next and the chance to interact with the site contents.

2 Search Engine System Architecture

This section provides an overview of how the whole system of a search engine works. The major functions of the search engine crawling, indexing and searching are also covered in detail in the later sections.

Before a search engine can tell you where a file or document is, it must be found. To find information on the hundreds of millions of Web pages that exist, a typical search engine employs special software robots, called *spiders*, to build lists of the words found on Web sites. When a spider is building its lists, the process is called *Web crawling*. A *Web crawler* is a program, which automatically traverses the web by downloading documents and following links from page to page. They are mainly used by web search engines to gather data for indexing. Other possible applications include page validation, structural analysis and visualization, update notification, mirroring and personal web assistants/agents etc. Web crawlers are also known as spiders, robots, worms etc. Crawlers are automated programs that follow the links found on the web pages.

There is a URL Server that sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the store server. The store server then compresses and stores the web pages into a repository. Every web page has an associated ID number called a doc ID, which is assigned whenever a new URL is parsed out of a web page. The indexer and the sorter perform the indexing function. The indexer performs a number of functions. It reads the repository, uncompresses the documents, and parses them. Each document is converted into a set of word occurrences called hits. The hits record the word, position in document, an approximation of font size, and capitalization. The indexer distributes these hits into a set of "barrels", creating a partially sorted forward index. The indexer performs another important function. It parses out all the links in every web page and stores important information about them in an anchors file. This file contains enough information to determine where each link points from and to, and the text of the link.

The URL Resolver reads the anchors file and converts relative URLs into absolute URLs and in turn into doc IDs. It puts the anchor text into the forward index, associated with the doc ID that the anchor points to. It also generates a database of links, which are pairs of doc IDs. The links database is used to compute Page Ranks for all the documents.

The sorter takes the barrels, which are sorted by doc ID and resorts them by word ID to generate the inverted index. This is done in place so that little temporary space is needed for this operation. The sorter also produces a list of word IDs and offsets into the inverted index. A program called Dump Lexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. A lexicon lists all the terms occurring in the index along with some term-level statistics (e.g., total number of documents in which a term occurs) that are used by the ranking algorithms
The searcher is run by a web server and uses the lexicon built by Dump Lexicon together with the inverted index and the Page Ranks to answer queries. (Brin and Page 1998)

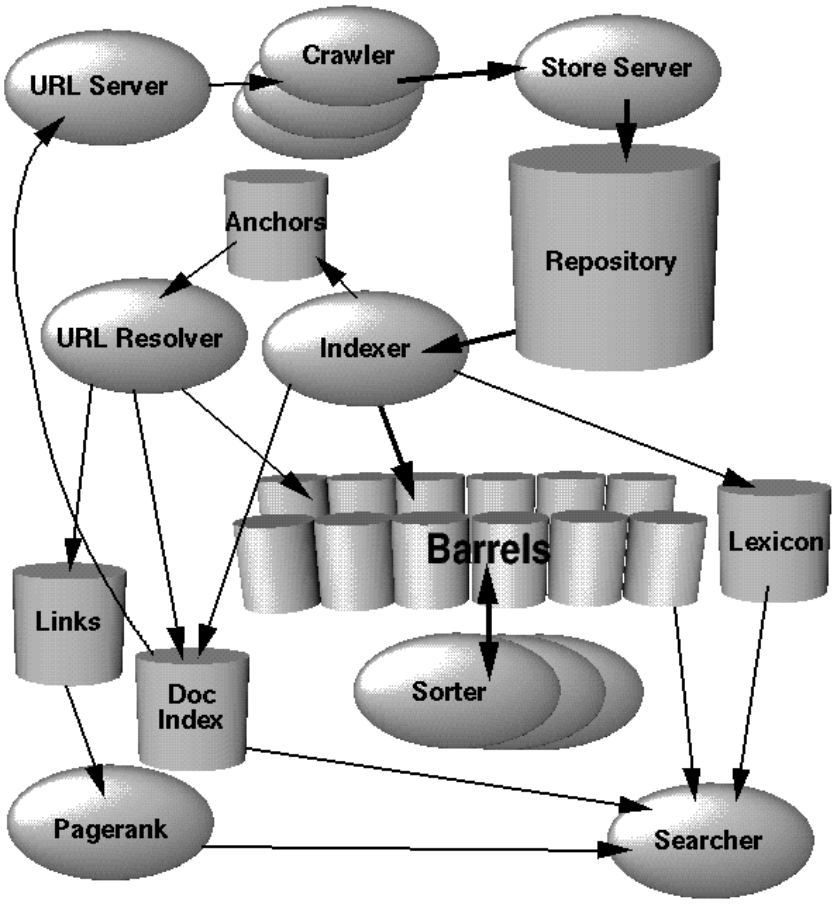

Figure 1: High Level Search Engine Architecture (Brin and Page, 1998)

3 How a Web Crawler Works

Web crawlers are an essential component to search engines; running a web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application since it involves interacting with hundreds of thousands of web servers and various name servers, which are all beyond the control of the system. Web crawling speed is governed not only by the speed of one's own Internet connection, but also by the speed of the sites that are to be crawled. Especially if one is a crawling site from multiple servers, the total crawling time can be significantly reduced, if many downloads are done in parallel.

Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same. Following is the process by which Web crawlers work:

1. Download the Web page.
2. Parse through the downloaded page and retrieve all the links.
3. For each link retrieved, repeat the process.

The Web crawler can be used for crawling through a whole site on the Inter-/Intranet. You specify a start-URL and the Crawler follows all links found in that HTML page. This usually leads to more links, which will be followed again, and so on. A site can be seen as a tree-structure, the root is the start-URL; all links in that root-HTML-page are direct sons of the root. Subsequent links are then sons of the previous sons.

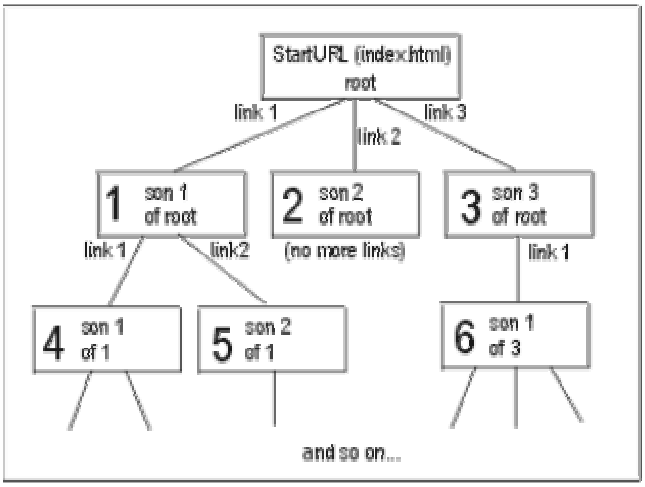

Figure 2: Working of a web crawler

A single URL Server serves lists of URLs to a number of crawlers. Web crawler starts by parsing a specified web page, noting any hypertext links on that page that point to other web pages. They then parse those pages for new links, and so on, recursively. Web-crawler software doesn't actually move around to different computers on the Internet, as viruses or intelligent agents do. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve web pages at a fast enough pace. A crawler resides on a single machine. The crawler simply sends HTTP requests for documents to other machines on the Internet, just as a web browser does when the user clicks on links. All the crawler really does is to automate the process of following links.

Web crawling can be regarded as processing items in a queue. When the crawler visits a web page, it extracts links to other web pages. So the crawler puts these URLs at the end of a queue, and continues crawling to a URL that it removes from the front of the queue. (Garcia-Molina 2001)

3.1 Resource Constraints

Crawlers consume resources: network bandwidth to download pages, memory to maintain private data structures in support of their algorithms, CPU to evaluate and select URLs, and disk storage to store the text and links of fetched pages as well as other persistent data.

4 Meta-Search Engine

A meta-search engine is the kind of search engine that does not have its own database of Web pages. It sends search terms to the databases maintained by other search engines and gives users the results that come from all the search engines queried. Fewer meta-searchers allow you to delve into the largest, most useful search engine databases. They tend to return results from smaller and/or free search engines and miscellaneous free directories, often small and highly commercial.

The mechanism and algorithms that meta-search engines employ are quite different. The simplest meta-search engines just pass the queries to other direct search engines. The results are then simply displayed in different newly opened browser windows as if several different queries were posed. Some improved meta-search engines organize the query results in one screen in different frames, or in one frame but in a sequential order. Some more sophisticated meta-search engines permit users to choose their favorite direct search engines in the query input process, while using filters and other algorithms to process the returned query results before displaying them to the users.

Problems often arise in the query-input process though. Meta-Search engines are useful if the user is looking for a unique term or phrase; or if he (she) simply wants to run a couple of keywords. Some meta-search engines simply pass search terms along to the underlying direct search engine, and if a search contains more than one or two words or very complex logic, most of them will be lost. It will only make sense to the few search engines that supports such logic. Following are some of the powerful meta-search engines with some direct search engines like AltaVista and Yahoo. (Liu, 1999)

4.1 Dogpile

It uses About.com, Alta Vista, Dogpile Open Directory, GoTo.com, Infoseek, Lycos, Lycos' Top 5%, Thunderstone and “Yahoo!” as its underlying search tools for Web search queries. It is customizable and supports OR, NOT, and "()". The default logic between different query words is "AND". Dogpile does not sort the query results received from the search engines. The output is a concatenation of the different results with the ranking by each search tool; therefore, there may be duplicates in Dogpile's output. The results are displayed sequentially. (Liu, 1999)

4.2 MetaCrawler

It uses Lycos, About.com, Alta Vista, Excite, Infoseek, Looksmart, Thunderstone, WebCrawler, and “Yahoo!” as its tools for Web search queries. It is customizable, and also provides "Power Search". It consolidates results in one large list, which are ranked by a "vote" score. It doesn't support Boolean logic in its queries. It supports “ALL”, “ANY” or exact PHRASE to, and allows the use of "+/-", "" around phrases. (Liu, 1999)

5 Improvements of Web Search Engines

There are different ways to improve the performance of web search engines. Generally speaking, there are three main directions:

1. Improving user interface on query input
2. Using Filtering towards the query results
3. Solving algorithms in web page spying and collecting, indexing, and output

Method 3 is the fundamental solution for any direct search engines to deal with the problems of unequal accessing, out-of-date information, and low metadata using, as well as information coverage. It is also very important to look at the user interface issues that methods 1 and 2 are dealing with — How to handle user queries effectively and present the results efficiently. (Baldi, 2003)

6 Basic Types of Search Tools

6.1 Crawler Based Search Engines – Crawler based search engines create their listings automatically. Computer programs 'spiders' build them not by human selection. They are not organized by subject categories; a computer algorithm ranks all pages. Such kinds of search engines are huge and often retrieve a lot of information -- for complex searches it allows to search within the results of a previous search and enables you to refine search results. These types of search engines contain full text of the web pages they link to. So one can find pages by matching words in the pages one wants. (Grossan, 1997)

6.2 Human Powered Directories – These are built by human selection i.e. they depend on humans to create listings. They are organized into subject categories and subjects do classification of pages. Human powered directories never contain full text of the web page they link to. They are smaller than most search engines. (Grossan, 1997)

7 Crawling Techniques

7.1 Focused Crawling

A general purpose Web crawler gathers as many pages as it can from a particular set of URL's. Where as a focused crawler is designed to only gather documents on a specific topic, thus reducing the amount of network traffic and downloads. The goal of the focused crawler is to selectively seek out pages that are relevant to a pre-defined set of *topics*. The topics are specified not using keywords, but using exemplary documents. Rather than collecting and indexing all accessible web documents to be able to answer all possible ad-hoc queries, a focused crawler analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the web. This leads to significant savings in hardware and network resources, and helps keep the

crawl more up-to-date. The focused crawler has three main components: a classifier, which makes relevance judgments on pages crawled to decide on link expansion, a distiller which determines a measure of centrality of crawled pages to determine visit priorities, and a crawler with dynamically reconfigurable priority controls which is governed by the classifier and distiller.

The most crucial evaluation of focused crawling is to measure the harvest ratio, which is rate at which relevant pages are acquired and irrelevant pages are effectively filtered off from the crawl. This harvest ratio must be high, otherwise the focused crawler would spend a lot of time merely eliminating irrelevant pages, and it may be better to use an ordinary crawler instead (Baldi, 2003).

Distributed Crawling

Indexing the web is a challenge due to its growing and dynamic nature. As the size of the Web is growing it has become imperative to parallelize the crawling process in order to finish downloading the pages in a reasonable amount of time. A single crawling process even if multithreading is used will be insufficient for large – scale engines that need to fetch large amounts of data rapidly. When a single centralized crawler is used all the fetched data passes through a single physical link. Distributing the crawling activity via multiple processes can help build a scalable, easily configurable system, which is fault tolerant system. Splitting the load decreases hardware requirements and at the same time increases the overall download speed and reliability. Each task is performed in a fully distributed fashion, that is, no central coordinator exists (Baldi, 2003).

7 Robot Protocol

Web sites also often have restricted areas that crawlers should not crawl. To address these concerns, many Web sites adopted the Robot protocol, which establishes guidelines that crawlers should follow. Over time, the protocol has become the unwritten law of the Internet for Web crawlers. The Robot protocol specifies that Web sites wishing to restrict certain areas or pages from crawling have a file called robots.txt placed at the root of the Web site. The ethical crawlers will then skip the disallowed areas. Following is an example robots.txt file and an explanation of its format:

 # robots.txt for http://somehost.com/

*User-agent: **

Disallow: /cgi-bin/

Disallow: /registration # Disallow robots on registration page

Disallow: /login

The first line of the sample file has a comment on it, as denoted by the use of a hash (#) character. Crawlers reading robots.txt files should ignore any comments.

The third line of the sample file specifies the User-agent to which the Disallow rules following it apply. User-agent is a term used for the programs that access a Web site. Each browser has a unique User-agent value that it sends along with each request to a Web server. However, typically Web sites want to disallow all robots (or User-agents) access to certain areas, so they use a value of asterisk (*) for the User-agent. This specifies that all User-agents be disallowed for the rules that follow it. The lines following the User-agent lines are called disallow statements. The disallow statements define the Web site paths that crawlers are not allowed to access. For example, the first disallow statement in the sample file tells crawlers not to crawl any links that begin with "/cgi-bin/". Thus, the following URLs are both off limits to crawlers according to that line.
http://somehost.com/cgi-bin/
http://somehost.com/cgi-bin/register (Searching Indexing Robots and Robots.txt 2002)

8 Major Data Structures Involved

8.1 Repository

A Web repository stores and manages a large collection of 'data objects' in this case web page. It contains the full HTML of every web page. The web pages are compressed in the repository. Various methods of data compression can be used e.g. gzip, zlib. The choice of compression technique is a tradeoff between speed and compression ratio. In the repository, the documents are stored one after the other and are prefixed by docID, length, and URL as can be seen in Figure. The repository requires no other data structures to be used in order to access it. This helps with data consistency and makes development much easier. (Brin and Page 1998)

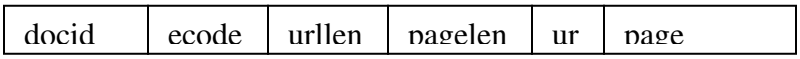

Figure 3: Repository Data Structure - Packet (stored compressed in repository)

8.2 Document Index

The document index keeps information about each document. It is a fixed width ISAM (Index sequential access mode) index, ordered by doc ID. The information stored in each entry includes the current document status, a pointer into the repository, a document checksum, and various statistics. If the document has been crawled, it also contains a pointer into a variable width file called doc info, which contains its URL and title. Otherwise the pointer points into the URL list, which contains just the URL (Brin and Page 1998).

8.3 Indexer

An Indexer is a program that “reads” the pages, which are downloaded by spiders. The indexer does most of the work deciding what the web site is about. Web indexing includes back-of-book-style indexes to individual websites or web documents and the creation of metadata (subject keywords and description tags) to provide a more useful vocabulary for Internet search engines. The indexer also examines the HTML code, which makes up the webpage and looks for words that are considered important. Words in bold, italics or header tags are given more priority. Each web database has a different indexing method (Brin and Page 1998).

8.4 Hit List

A hit list corresponds to a list of occurrences of a particular word in a particular document including position, font, and capitalization information. Hit lists account for most of the space used in both the forward and the inverted indices. Because of this, it is important to represent them as efficiently as possible. There are two types of hits: fancy hits and plain hits. Fancy hits include hits occurring in a URL, title, anchor text, or Meta tag. Plain hits include everything else. A plain hit consists of a capitalization bit, font size, and 12 bits of word position in a document. A fancy hit consists of a capitalization bit, the font size set to 7 to indicate it is a fancy hit, 4 bits to encode the type of fancy hit, and 8 bits of position (Chakrabarti, 2003).

9 Methods of Indexing

There are two important methods of indexing used in web database creation - full-text and human. (Nicholson, 2002)

9.1 Full-Text Indexing

As its name implies, full-text indexing is where every word on the page is put into a database for searching. Alta Vista, Google, Infoseek, Excite are examples of full-text databases. Full-text indexing will help you find every example of a reference to a specific name or terminology. A general topic search will not be very useful in this database, and one has to dig through a lot of "false drops" (or returned pages that have nothing to do with the search). In this case the websites are indexed my computer software. This software called “spiders” or “robots” automatically seeks out Web sites on the Internet and retrieves information from those sites (which matches the search criteria) using set instructions written into the software. This information is then automatically entered into a database.

9.2 Key word Indexing

In key word indexing only “important” words or phrases are put into the database. Lycos is a good example of key word indexing.

9.3 Human Indexing

Yahoo and some of Magellan are two of the few examples of human indexing. In the Key word indexing, all of the work was done by a computer program called a "spider" or a "robot". In human indexing, a person examines the page and determines a very few key phrases that describe it. This allows the user to find a good start of works on a topic - assuming that the topic was picked by the human as something that describes the page. This is how the directory-based web databases are developed.

10 Indexing the web content

Similar to an index of a book, a search engine also extracts and builds a catalog of all the words that appear on each web page and the number of times it appears on that page etc. Indexing of web content is a challenging task assuming an average of 1000 words per web page and billions of such pages. Indexes are used for searching by keywords; therefore, it has to be stored in the memory of computers to provide quick access to the search results.

Indexing starts with ***parsing*** the website content using a parser. Any parser, which is designed to run on the entire Web, must handle a huge array of possible errors. The parser can extract the relevant information from a web page by excluding certain common words (such as a, an, the - also known as stop words), HTML tags, Java Scripting and other bad characters. A good parser can also eliminate commonly occurring content in the website pages such as navigation links, so that they are not counted as a part of the page's content. Once the indexing is completed, the results are stored in memory, in a sorted order. This helps in retrieving the information quickly. Indexes are updated periodically as new content is crawled. Some indexes help create a dictionary (lexicon) of all words that are available for searching. Also a lexicon helps in correcting mistyped words by showing the corrected versions in a search result. A part of the success of the search engine lies in how the indexes are built and used. Various algorithms are used to optimize these indexes so that relevant results are found easily without much computing resource usage (Pant, 2003) (Chakrabarti, 2003).

11 Storing the Web Content

In addition to indexing the web content, the individual pages are also stored in the search engine's database. Due to cheaper disk storage, the storage capacity of search engines is very huge, and often runs into terabytes of data. However, retrieving this data quickly and efficiently requires special distributed and scalable data storage functionality. The amount of data, that a search engine can store, is limited by the amount of data it can retrieve for search results. Google can index and store about 3 billion web documents. This capacity is far more than any other search engine during this time. "Spiders" take a Web page's content and create key search words that enable online users to find pages they're looking for. (Franklin, 2002)

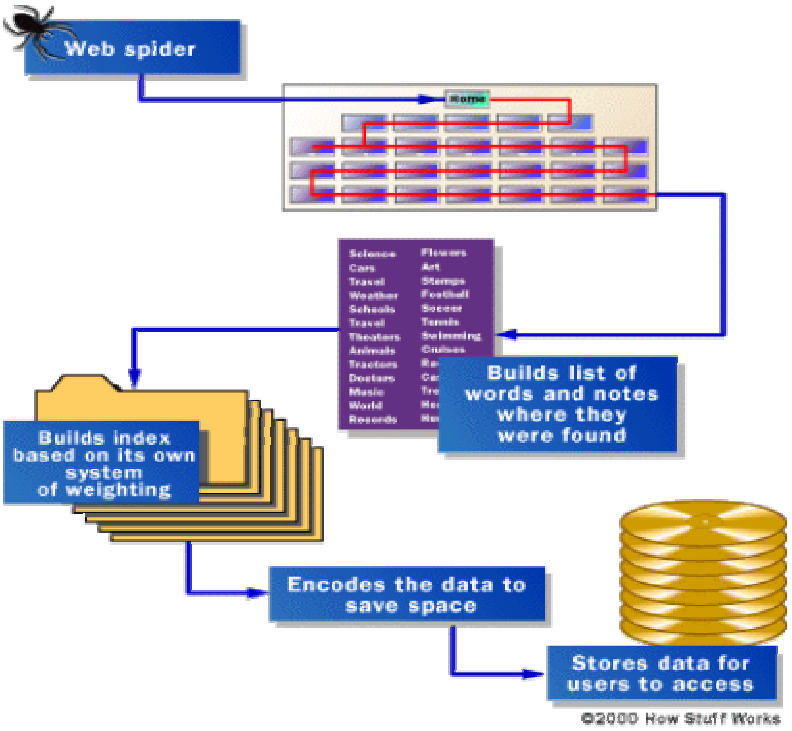

Figure 4: How a Web Spider Works (Franklin, 2002)

12 Web Crawler Application Development Methodology

12.1 Analysis

The web crawler application will be developed in Java 1.4.2 for many reasons. First Java's support for networking makes downloading Web pages simple. Second, Java's support for regular expression processing simplifies the finding of links. Third, Java's Collection Framework supplies the mechanisms needed to store a list of links.

The overall tool will be developed in an incremental fashion, giving highest priority to The most important features. The web crawler is a basic search crawler application for searching the web and it illustrates the fundamental structure of crawler-based applications. With this application you can search the web, in real time, URL by URL, looking for matches to the criteria. The GUI will be developed first, which should help in identifying all of the various actions that can be taken (e.g. specifying the start URL, maximum number of URL's to crawl). This application is developed using java1.4.2 because of the various features that support web crawler application. The java.net package provides a powerful and flexible infrastructure for networking. This package provides classes for implementing networking applications. Using the socket classes, one can communicate with any server on the Internet or implement his own Internet server. A number of classes are provided to make it convenient to use Universal Resource Locators (URLs) to retrieve data on the Internet.

Some of the key points about the Web crawler are:

1. Only HTTP links are supported not HTTPS or FTP
2. URL's that redirect to other URL's are not supported
3. Links such as 'http://gmail.com and http://gmail.com/ are treated as separate unique links. This is because the Web Crawler cannot categorically know that both are the same in all instances.

Feature	Support
Searching a particular search string	Yes
Programming language support	Java
Operating system support	Windows XP
Online help manual	No
Integration with other applications	No
Specifying case sensitivity for a search string	Yes
Specifying maximum number of URL's to search	Yes
Specifying the start URL	Yes
Supports breath - first crawling	Yes
Supports depth - first crawling	No
Supports broken link crawling	No
Supports comparison crawling	No
Supports archive crawling	No
Will check if the given URL is valid	Yes
Checks for 'disallow' statements in robot.txt file	Yes

Table 1. Proposed Functionalities for the Web crawler application

12.2 Design

The Web Crawler Application is divided into three main modules.

1. Controller
2. Fetcher
3. Parser

Controller Module - This module focuses on the Graphical User Interface (GUI) designed for the web crawler and is responsible for controlling the operations of the crawler. The GUI enables the user to enter the start URL, enter the maximum number of URL's to crawl, view the URL's that are being fetched. It controls the Fetcher and Parser.

Fetcher Module - This module starts by fetching the page according to the start URL specified by the user. The fetcher module also retrieves all the links in a particular page and continues doing that until the maximum number of URL's is reached.

Parser Module - This module parses the URL's fetched by the Fetcher module and saves the contents of those pages to the disk.

12.3 Implementation

This Web crawler application builds on this knowledge and uses ideas from previous crawlers. This is purely a Java application. The choice of Java v1.4.2 as an implementation language is motivated by the need to achieve platform independence.

It uses various classes and interfaces in java, which illustrate java's networking capabilities. Moreover, Java makes it possible to adopt *Remote Method Invocation*, a technology that enables one to create distributed applications in which the methods of remote Java objects can be invoked from other Java virtual machines. The currently proposed web crawler application uses the breath-first crawling to search the links. Breadth-first crawling checks each link on a page before proceeding to the next page. Thus, it crawls each link on the first page and then crawls each link on the first page's first link, and so on, until each level of links has been exhausted

13 Conclusion

This paper explains the anatomy of typical search engine and demonstrates the working of a web crawler developed in Java. It discusses the functionalities of all the components involved in finding information on the Web.

14 References

1. Baldi, Pierre. Modeling the Internet and the Web: Probabilistic Methods and Algorithms, 2003.
2. Brin, Sergey and Page Lawrence. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, April 1998
3. Chakrabarti, Soumen. Mining the Web: Analysis of Hypertext and Semi Structured Data, 2003
4. Franklin, Curt. How Internet Search Engines Work, 2002.

   www.howstuffworks.com
5. Grossan, B. "Search Engines: What they are, how they work, and practical suggestions for getting the most out of them," February1997. http://www.webreference.com

6. Garcia-Molina, Hector. Searching the Web, August 2001
   http://oak.cs.ucla.edu/~cho/papers/cho-toit01.pdf
7. Liu, Jian. Guide to Meta-Search Engines. Reference Department, Indiana University Libraries. June 1999
   http://www.indiana.edu/~librcsd/search/meta.html
8. Nicholson, Scott. Introduction to Web Databases, 2002
   http://www.askscott.com/sec1.html#methods
9. Pant, Gautam, Padmini Srinivasan and Filippo Menczer: Crawling the Web, 2003.
   http://dollar.biz.uiowa.edu/~pant/Papers/crawling.pdf
10. Search Indexing Robots and Robots.txt, 2002
    http://www.searchtools.com/robots/robots-txt.html

15 Acknowledgements

I would like to thank my advisor, Dr. Kaymar Dezhgosha for providing me with valuable resources and insights needed for my project. He has been a guideline for me throughout the project. I really appreciate his valuable time spent for the betterment of this project.

I would like to thank all my professors from the Computer Science Department who have helped enrich my knowledge and for all their support and encouragement. Finally I would like to thank my husband and my family for all their support and belief in me and for being a knack boosting morale during rough times.