VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

SYNOPSIS FOR FINAL YEAR PROJECT

This is to certify that the following students have satisfactorily carried out the work for the synopsis of Project entitled:

GENERIC SEARCH ENGINE

In partial fulfillment of BE in Information Technology of the University of Mumbai during the academic year 2006-2007.

Project Group Members :

Basesh Gala GH04955 (60)
Avani Vadera GH031094 (53)
Tejal Bhatt GH04952 (57)
Swapnil Mahtre GH031065 (30)
Sneha Rajan GH031091(50)

_____ _____

PROJECT GUIDE HEAD OF DEPARTMENT

_____ _____

INTERNAL EXAMINER EXTERNAL EXAMINER

INTRODUCTION

We, the students of B.E.I.T.,V.J.T.I. are making a project with I.I.T., Mumbai on '***Search Engine***' as part of the final year curriculum.

We are presenting a synopsis which is phase 1 of the project in which the design(s) of a basic search engine is studied. The report also includes a comparative study of various search engines. A survey has been conducted to find out which search engine is the most popular and widely used. The report includes a detailed study of GOOGLE which has been the most popular search engine.

We have identified the need for a specialized animation based search engine and present the analysis and the design for the same.

INDEX

1. INTRODUCTION

1.1. PROBLEM DEFINITION:

To design a Search Engine for searching and indexing specific web pages on the Internet(First phase for OSCAR project). OSCAR project is an open source project which is responsible for creating and promoting the educational animations on the Internet. Hence this search engine searches only educational animations to support the motivation behind OSCAR project. It then proceeds to be applicable for the search of any kind of web pages.

The general purpose interactive animations are distributed using either Macromedia Flash (.swf) or Java Applet (.class) format; while non-interactive animations are created using any of multimedia format e.g. GIF, AVI etc. Since most of the educational animations need to be interactive, this search engine is intended to search only those animations which are stored in either Macromedia Flash or Java Applet format.

1.2. PITFALLS OF GENERIC SEARCH ENGINES:

The weaknesses of current system can be discussed using the following features

1) Irrelevant results
2) Indexing of non-text data.

When we submit a query to a generic search engine (e.g. Google, Ask etc) to find animations it gives very irrelevant results. This is because such types of search engine finds available words of query within the pages and if given page contains all of such words then it is included in result set. It does not check, whether given page contains animations or not.

Secondly, generic search engines are not capable of indexing files which are used for storing animations. Consider a page which contains only Flash or Applet and no data to display on page. Then such pages are not properly indexed because the generic search engines are not capable of indexing text data used for storing animations.

1.3. NEED FOR ANIMATION SPECIFIC SEARCH ENGINE:

From the above points it is clear that currently there is no system which could efficiently find animations (specifically, educational animations) on the Internet. There is need to create such a search engine which could do both, index non- text data of animation files as well as be able to use special techniques such as subject specific *PageRank* to give quality results to user queries.

Animation specific search engine:

Given search engine has the following parts:

1) **Crawler:** Crawler downloads the web pages from the web sites distributing animations on the Internet.
2) **Metadata Extractor:** Metadata Extractor extracts the metadata present in the non-text animation files e.g. SWF files or CLASS files
3) **Indexer:** Indexes the downloaded pages according to there HTML content as well as metadata extracted from the animation files
4) **Ranking Criteria:** It decides which pages given higher rank compared to other depending upon some predefined algorithms.
5) **Searcher:** Accepts user query and searches pages satisfying given query and gives results based on the ranking criteria.

1.4. FEATURES PROVIDED:

The project provides following features.

- The important features & characteristic is search for animations that are created using JAVA applets or Flash. The animation search differs from image search & needs special techniques to recognize animation files from web pages.

- Dead link checker which would search for the non-functional web sites and thus inform the web developer.

- The permission to be granted in order to access the web site will be done using robot.txt file.

- The search process will be optimized for efficient search & to reduce response time.

- The search is provided in two modes.
    - Normal search: The basic search for animations can include multi word search.
    - Advanced search: The advanced search provides Boolean operators & other options.

2. SCOPE:

With a four month time constraint we students have looked into the analysis of the search engine. Animation based search engine is a specific area on which we will be dealing in the next 6 months prior to the implementation details of the anatomy of the search engine.

For gaining an insight into how the existing search engine works, a comparative study of various features the several engines offer have been made. A survey of the existing search engines has also been conducted in order to understand the in-addition expectations from current search engine.

Analysis & Design aspect of a search engine has been the focus of our project this semester.

3. GOALS:

- Study of the evolution of search engines.
- The need for search engine
- Specific tasks and features offered by search engines
- Survey based study
- Deign issues with generic based search engines

4. PROCESS MODEL SELECTION:

To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process, methods and tools layers and the generic phases, this strategy is called as a Process Model or a software engineering paradigm

A process model is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.
Various process models are:

1. The linear sequential model.
2. Prototyping model.
3. The spiral model.
4. RAD (Rapid Application Development) model.

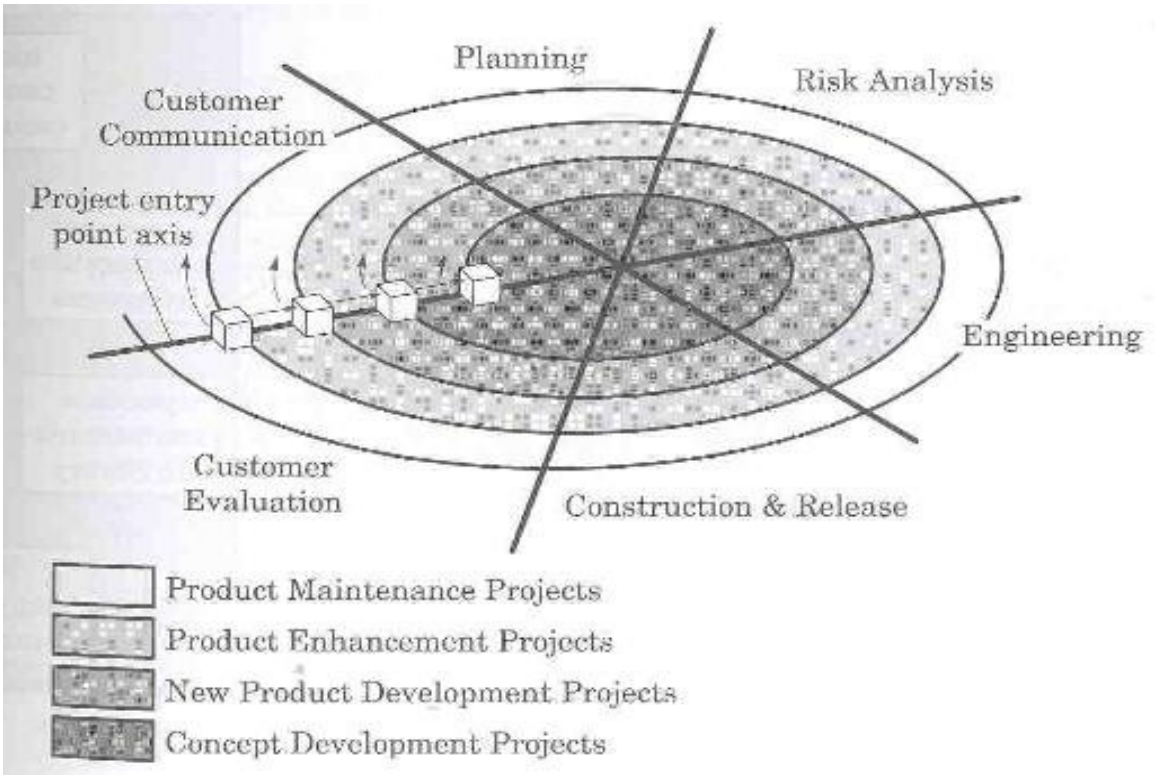
The model adopted for the development of "**GENERIC SEARCH ENGINE**" was "Spiral Model" which is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. The reason for choosing this model as the basis for the development of this project is that it provides the potential for rapid development of incremental versions of the software. Using the concepts of spiral model, SEARCH ENGINE will be developed as a series of incremental releases.

Each release will be an enhancement of the previous one. Initially the search engine will behave like a search tool in a desktop computer (standalone PC). The next release will be for an intranet and the last release for internet.

Spiral model

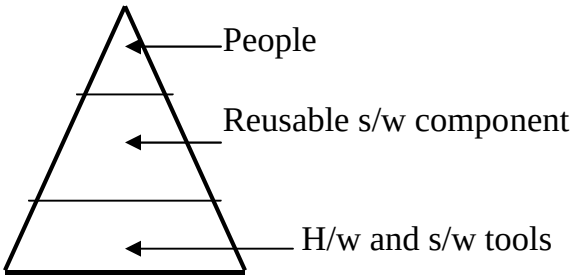
The spiral model for software engineering has been developed to encompass the best features of both the classic lifestyle and prototyping, while at the same time adding a new element: risk analysis that is missing in these paradigms. The model, represented by the spiral in figure given below defines four major activities represented by the four quadrants of the figure:

1. Planning: determination of objectives, alternatives and constraints.
2. Risk analysis: analysis of alternatives and identification or resolution of risks.
3. Engineering: development of the "next level "product.
4. Customer evaluation: assessment of the results of engineering.



5. RESOURCES REQUIRED:

After defining the scope, estimation of resources required to accomplish the software development effort is done. The figure below illustrates the development resources as a pyramid



HUMAN RESOURCES:

Among the many problems posed by the software crisis, none is more ominous than the relative scarcity of capable human resources for software development. People are the primary software development resource. A team of 5 people will be required for the construction and the development of the software, as regard to its size and complexity. Hence the human resource will be available easily for the planning.

TEAM SIZE:

The team BATSS comprises of 5 group members as follows.

1. Basesh Gala baseshg@yahoo.co.in
2. Tejal Bhatt tejalpbhatt@yahoo.co.in
3. Avani Vadera avani_b_vadera@yahoo.co.in
4. Swapnil Mahtre swapnil_me65@yahoo.co.in
5. Sneha Rajan sne252003@yahoo.com

HARDWARE RESOURCES:

1. We need to run crawler 24 * 7 (preferably) so that all requested web pages are crawled & the reverse index is created. Hence the search query is executed immediately.
2. The server needs enough main memory to be efficient enough(approximately 1 GB)
3. To keep all animation web pages that are downloaded in storage server to maintain forward & reverse index, we need at least two hard disk of 80 GB.

SOFTWARE REQUIRED:

1. JDK development kit for implementation of indexer & other functions.
2. The python for crawler implementation.
3. The database server for storage of crawled web pages & indexes.
4. The JDK decompiler to extract .class files.
5. The Flash decompiler to extract .swf file

6. DURATIONS:

The project duration is June 2006 to May 2007. A year is devoted for analysis, implementation & testing of project .

7. REPORTING MILESTONES:

- Project chart—By the end of October.
- Design---- By the end of December
- Coding---By the end of March
- Testing— The testing will go simultaneously with coding & we will have other test cases as needed . So till the end of May .

8. LITERARY SURVEY

8.1. INTRODUCTION OF A SEARCH ENGINE

A **search engine** or **search service** is a program designed to help find information stored on a computer system such as the World Wide Web, inside a corporate or proprietary network or a personal computer. The search engine allows one to ask for content meeting specific criteria (typically those containing a given word or phrase) and retrieves a list of references that match those criteria.

Search engines use regularly updated indexes to operate quickly and efficiently. *Search Engine* usually refers to a ***Web* search engine**, which searches for information on the public Web. However there are other kinds of search engines like ***enterprise search engines***, which search on intranets, ***personal search engines***, which search individual personal computers, and ***mobile search engines***.Some search engines also mine data available in newsgroups, large databases, or open directories like DMOZ.org. Unlike **Web directories**, which are maintained by human editors, search engines operate algorithmically. Most web sites which call themselves search engines are actually front ends to search engines owned by other companies.

8.2. TYPES OF SEARCH ENGINES

Although the term "search engine" is often used indiscriminately to describe crawler-based search engines, human-powered directories, and everything in between, they are not all the same. Each type of "search engine" gathers and ranks listings in radically different ways.

1. Crawler-Based

Crawler-based search engines such as Google, compile their listings automatically. They "crawl" or "spider" the web, and people search through their listings. These listings are what make up the search engine's index or catalog. You can think of the index as a massive electronic filing cabinet containing a copy of every web page the

spider finds. Because spiders scour the web on a regular basis, any changes you make to a web site may affect your search engine ranking.

It is also important to remember that it may take a while for a spidered page to be added to the index. Until that happens, it is not available to those searching with the search engine.

2.Directories

Directories such as Open Directory depend on human editors to compile their listings. Webmasters submit an address, title, and a brief description of their site, and then editors review the submission. Unless you sign up for a paid inclusion program, it may take months for your web site to be reviewed. Even then, there's no guarantee that your web site will be accepted.

After a web site makes it into a directory however, it is generally very difficult to change its search engine ranking. So before you submit to a directory, spend some time working on your titles and descriptions. Moreover, make sure your pages have solid well-written content.

3.Mixed Results /hybrid search engine

Some search engines offer both crawler-based results and human compiled listings. These hybrid search engines will typically favor one type of listing over the other however. Yahoo for example, usually displays human-powered listings. However, since it draws secondary results from Google, it may also display crawler-based results for more obscure queries. Many search engines today combine a spider engine with a directory service. The directory normally contains pages that have already been reviewed and accessed.

4.Pay Per Click

More recently, search engines have been offering very cost effective programs to ensure that your ads appear when a visitor enters in one of your keywords. This new trend is to charge you on a Cost per Click model (CPC).The listings are comprised entirely of advertisers who have paid to be there. With services such as Yahoo SM, Google AdWords, and FindWhat, bids determine search engine ranking. To get top ranking, an advertiser just has to outbid the competition.

5.Metacrawlers Or Meta Search Engines

Metasearch Engines search, accumulate and screen the results of multiple Primary Search Engines (i.e. they are search engines that search search engines) Unlike search engines, metacrawlers don't crawl the web themselves to build listings. Instead, they allow searches to be sent to several search engines all at once. The results are then blended together onto one page.

The search engine can be categorized as follows based on the application for which they are used:

1.Primary Search Engines:

They scan entire sections of the World Wide Web and produce their results from databases of Web page content, automatically created by computers.

2.Subject Guides :

They are like indexes in the back of a book. They involve human intervention in selecting and organizing resources, so they cover fewer resources and topics but provide more focus and guidance

3.People Search Engines :

They search for names, addresses, telephone numbers and e mail address.

4.Business and Services Search Engines:

They essentially national yellow page directories .

5.Employment and Job Search Engines

They either (a) provide potential employers access to resumes of people interested in working for them or (b) provides prospective employees with information on job availability .

6.Finance-Oriented Search Engines:

They facilitate searches for specific information about companies(officers, annual reports, SEC filings, etc.)

7.News Search Engines :

They search newspaper and news web site archives for the selected information

8.Image Search Engines:

They which help you search the WWW for images of all kinds.

9.Specialized Search Engines:

They that will search specialized databases, allow you to enter your search terms in a particularly easy way, look for low prices on items you are interested in purchasing, and even give you access to real, live human beings to answer your questions.

8.3.WORKING OF SEARCH ENGINE

Search engines use automated software programs known as **spiders** or **bots** to survey the Web and build their databases. Web documents are retreived by these programs and analyzed. Data collected from each web page are then added to the search engine index. When you enter a query at a search engine site, your input is checked against the search engine's index of all the web pages it has analyzed. The best URLs are then returned to you as hits, ranked in order with the best results at the top.

The working of a search engine can be thus divided into three steps

1. Web Crawling
2. Indexing
3. Ranking

8.3.1 WEB CRAWLING

A **web crawler** (also known as a **web spider** or **web robot**) is a program or automated script which browses the World Wide Web in a methodical, automated manner. Other less frequently used names for web crawlers are **ants**, **automatic indexers**, **bots**, and **worms**

This process is called ***web crawling* or *spidering***. Many legitimate sites, in particular search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine, that will index the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a web site, such as checking links or validating HTML code. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for spam).

A web crawler is one type of bot, or software agent. In general, it starts with a list of URLs to visit, called the **seeds**. As the crawler visits these URLs, it identifies all the hyperlinks in

the page and adds them to the list of URLs to visit, called the **crawl frontier**. URLs from the frontier are recursively visited according to a set of policies.

Crawling policies

There are two important characteristics of the Web that generate a scenario in which web crawling is very difficult: its large volume and its rate of change, as there are a huge number of pages being added, changed and removed every day. Also, network speed has improved less than current processing speeds and storage capacities.
The large volume implies that the crawler can only download a fraction of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change implies that by the time the crawler is downloading the last pages from a site, it is very likely that new pages have been added to the site, or that pages have already been updated or even deleted.

A crawler must carefully choose at each step which pages to visit next.

The behavior of a web crawler is the outcome of a combination of policies:

- A selection policy that states which pages to download.
- A re-visit policy that states when to check for changes to the pages.
- A politeness policy that states how to avoid overloading websites.
- A parallelization policy that states how to coordinate distributed web crawlers.

- **Selection policy**

Given the current size of the Web, even large search engines cover only a portion of the publicly available content; no search engine indexes more than 16% of the Web. As a crawler always downloads just a fraction of the Web pages, it is highly desirable that the downloaded fraction contains the most relevant pages, and not just a random sample of the Web.

This requires a metric of importance for prioritizing Web pages. The importance of a page is a function of its intrinsic quality, its popularity in terms of links or visits, and even of its URL (the latter is the case of vertical search engines restricted to a single top-level domain, or search engines restricted to a fixed Website). Designing a good selection policy has an

added difficulty: it must work with partial information, as the complete set of Web pages is not known during crawling.

Various selection policies are:

A crawler may only want to seek out HTML pages and avoid all other MIME types. In order to request only HTML resources, a crawler may make an HTTP HEAD request to determine a web resource's MIME type before requesting the entire resource with a GET request. To avoid making numerous HEAD requests, a crawler may alternatively examine the URL and only request the resource if the URL ends with .html, .htm or a slash. This strategy may cause numerous HTML web resources to be unintentionally skipped.

Some crawlers may also avoid requesting any resources that have a "?" in them (are dynamically produced) in order to avoid spider traps which may cause the crawler to download an infinite number of URLs from a website.

1. **Path-ascending crawling**

Some crawlers intend to download as many resources as possible from a particular web site. Cothey (Cothey, 2004) introduced a *path-ascending crawler* that would ascend to every path in each URL that it intends to crawl. For example, when given a seed URL of http://foo.org/a/b/page.html, it will attempt to crawl /a/b/, /a/, and /. Cothey found that a path-ascending crawler was very effective in finding isolated resources, or resources for which no inbound link would have been found in regular crawling.

2. **Focused crawling**

The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query. Web crawlers that attempt to download pages that are similar to each other are called **focused crawlers** or **topical crawlers**. Focused crawling was first introduced by Chakrabarti et al.

The main problem in focused crawling is that in the context of a web crawler, we would like to be able to predict the similarity of the text of a given page to the query before actually downloading the page. The performance of a focused crawling depends mostly on the richness of links in the specific topic being searched, and a focused crawling usually relies on a general Web search engine for providing starting points.

3. Crawling the Deep Web

A vast amount of web pages lie in the deep or invisible web. These pages are typically only accessible by submitting queries to a database, and regular crawlers are unable to find these pages if there are no links that point to them. Google's Sitemap Protocol and mod_oai (Nelson et al., 2005) are intended to allow discovery of these deep-web resources.

- **Re-visit policy**

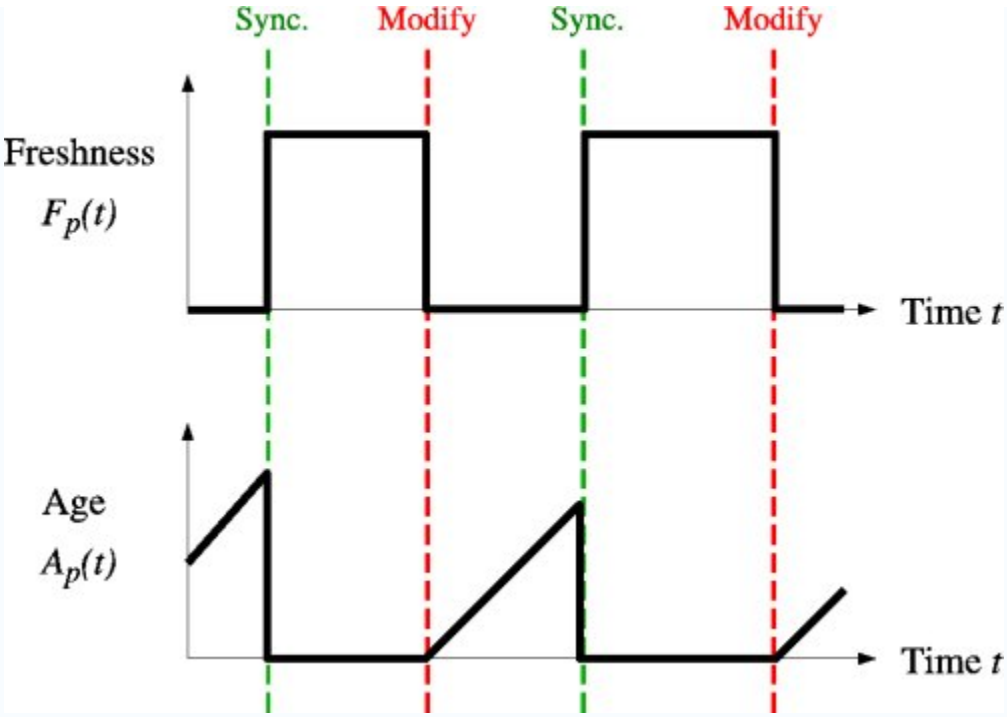
The Web has a very dynamic nature, and crawling a fraction of the Web can take a long time, usually measured in weeks or months. By the time a web crawler has finished its crawl, many events could have happened. These events can include creations, updates and deletions.
From the search engine's point of view, there is a cost associated with not detecting an event, and thus having an outdated copy of a resource. The most used cost functions, introduced in (Cho and Garcia-Molina, 2000), are freshness and age.

Freshness: This is a binary measure that indicates whether the local copy is accurate or not. The freshness of a page *p* in the repository at time *t* is defined as:

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases}$$

Age This is a measure that indicates how outdated the local copy is. The age of a page *p* in the repository, at time *t* is defined as:

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t - \text{modification time of } p & \text{otherwise} \end{cases}$$



Evolution of freshness and age in Web crawling

The objective of the crawler is to keep the average freshness of pages in its collection as high as possible, or to keep the average age of pages as low as possible. These objectives are not equivalent: in the first case, the crawler is just concerned with how many pages are out-dated, while in the second case, the crawler is concerned with how old the local copies of pages are.

Two simple re-visiting policies were studied by Cho and Garcia-Molina

1. **Uniform policy**:

This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.

2. **Proportional policy**:

This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.

in terms of average freshness, the uniform policy outperforms the proportional policy in both a simulated Web and a real Web crawl. The explanation for this result comes from the fact that, when a page changes too often, the crawler will waste time by trying to re-crawl it too fast and still will not be able to keep its copy of the page fresh.

- **Politeness policy**

Crawlers can retrieve data much quicker and in greater depth than human searchers, so they can have a crippling impact on the performance of a site. Needless to say if a single crawler is performing multiple requests per second and/or downloading large files, a server would have a hard time keeping up with requests from multiple crawlers.

As noted by Koster (Koster, 1995), the use of Web crawlers is useful for a number of tasks, but comes with a price for the general community. The costs of using Web crawlers include:

- Network resources, as crawlers require considerable bandwidth and operate with a high degree of parallelism during a long period of time.
- Server overload, especially if the frequency of accesses to a given server is too high.
- Poorly written crawlers, which can crash servers or routers, or which download pages they cannot handle.
- Personal crawlers that, if deployed by too many users, can disrupt networks and Web servers.

A partial solution to these problems is the robots exclusion protocol, also known as the robots.txt protocol (Koster, 1996) that is a standard for administrators to indicate which parts of their Web servers should not be accessed by crawlers. This standard does not include a suggestion for the interval of visits to the same server, even though this interval is the most effective way of avoiding server overload. A non-standard robots.txt file may use a "Crawl-delay:" parameter to indicate the number of seconds to delay between requests, and some commercial search engines like MSN and Yahoo will adhere to this interval Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes. It is worth noticing that even when being very polite, and taking all the safeguards to avoid overloading Web servers, some complaints from Web server administrators are received.

- **Parallelization policy**

Main article: Distributed web crawling

A parallel crawler is a crawler that runs multiple processes in parallel. The goal is to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page. To avoid downloading the same page more than once, the crawling system requires a policy for assigning the new URLs discovered during the crawling process, as the same URL can be found by two different crawling processes. Cho and Garcia-Molina (Cho and Garcia-Molina, 2002) studied two types of policies:

1. **Dynamic assignment**:

With this type of policy, a central server assigns new URLs to different crawlers dynamically. This allows the central server to, for instance, dynamically balance the load of each crawler.
With dynamic assignment, typically the systems can also add or remove downloader processes. The central server may become the bottleneck, so most of the workload must be transferred to the distributed crawling processes for large crawls.
There are two configurations of crawling architectures with dynamic assignments that have been described by Shkapenyuk and Suel (Shkapenyuk and Suel, 2002):

- A small crawler configuration, in which there is a central DNS resolver and central queues per website, and distributed downloaders.
- A large crawler configuration, in which the DNS resolver and the queues are also distributed.

2. **Static assignment**:

With this type of policy, there is a fixed rule stated from the beginning of the crawl that defines how to assign new URLs to the crawlers.
For static assignment, a hashing function can be used to transform URLs (or, even better, complete website names) into a number that corresponds to the index of the corresponding crawling process. As there are external links that will go from a website assigned to one crawling process to a website assigned to a different crawling process, some exchange of URLs must occur.
To reduce the overhead due to the exchange of URLs between crawling processes, the exchange should be done in batch, several URLs at a time, and the most cited URLs in the collection should be known by all crawling processes before the crawl (e.g.: using data from a previous crawl) (Cho and Garcia-Molina, 2002).

An effective assignment function must have three main properties: each crawling process should get approximately the same number of hosts (balancing property), if the number of crawling processes grows, the number of hosts assigned to each process must shrink (contra-variance property), and the assignment must be able to add and remove crawling processes dynamically. Boldi et al. (Boldi et al., 2004) propose to use consistent hashing, which replicates the buckets, so adding or removing a bucket does not requires re-hashing of the whole table to achieve all of the desired properties. crawling is an effective process synchronisation tool between the users and the search engine.

URL normalization

Crawlers usually perform some type of URL normalization in order to avoid crawling the same resource more than once. The term **URL normalization**, also called **URL canonicalization**, refers to the process of modifying and standardizing a URL in a consistent manner. There are several types of normalization that may be performed including conversion of URLs to lowercase, removal of "." and ".." segments, and adding trailing slashes to the non-empty path component

Crawler identification

Web crawlers typically identify themselves to a web server by using the User-agent field of an HTTP request. Website administrators typically examine their web servers' log and use the user agent field to determine which crawlers have visited the web server and how often. The user agent field may include a URL where the website administrator may find out more information about the crawler. Spambots and other malicious web crawlers are unlikely to place identifying information in the user agent field, or they may mask their identity as a browser or other well-known crawler.
It is important for web crawlers to identify themselves so website administrators can contact the owner if needed. In some cases, crawlers may be accidentally trapped in a crawler trap or they may be overloading a web server with requests, and the owner needs to stop the crawler. Identification is also useful for administrators that are interested in knowing when they may expect their web pages to be indexed by a particular search engine.

8.3.2 INDEXING

The web pages searched by the Web Crawlers have to be indexed for future use so that whenever search for related articles is asked the result can be provided by looking up the index.The index stores the URL along with the keywords relevant to the page.It may also store information like the dateon which the page was last visited and also the date on which the page should be revisited under the revisiting policy along with the age and freshness factors.

8.3.3 RANKING

Most of the search engines return results with confidence or relevancy rankings. In other words, they list the hits according to how closely they think the results match the query. However, these lists often leave users shaking their heads on confusion, since, to the user, the results may seem completely irrelevant.

Basically this happens because search engine technology has not yet reached the point where humans and computers understand each other well enough to communicate clearly.

Most search engines use search term frequency as a primary way of determining whether a document is relevant. If you're researching diabetes and the word "diabetes" appears multiple times in a Web document, it's reasonable to assume that the document will contain useful information. Therefore, a document that repeats the word "diabetes" over and over is likely to turn up near the top of your list.

If your keyword is a common one, or if it has multiple other meanings, you could end up with a lot of irrelevant hits. And if your keyword is a subject about which you desire information, you don't need to see it repeated over and over--it's the information *about* that word that you're interested in, not the word itself.

Some search engines consider both the frequency and the positioning of keywords to determine relevancy, reasoning that if the keywords appear early in the document, or in the headers, this increases the likelihood that the document is on target. For example, one method is to rank hits according to how many times your keywords appear and in which fields they appear (i.e., in headers, titles or plain text). Another method is to determine which documents are most frequently linked to other documents on the Web. The reasoning here is that if other folks consider certain pages important, you should, too.

If you use the advanced query form on AltaVista, you can assign relevance weights to your query terms before conducting a search. Although this takes some practice, it essentially allows you to have a stronger say in what results you will get back.

As far as the user is concerned, relevancy ranking is critical, and becomes more so as the sheer volume of information on the Web grows. Most of us don't have the time to sift through scores of hits to determine which hyperlinks we should actually explore. The more clearly relevant the results are, the more we're likely to value the search engine.

8.3.4 STORAGE COSTS AND CRAWLING TIME

Storage costs are not the limiting resource in search engine implementation. Simply storing 10 billion pages of 10kbytes each (compressed) requires 100TB and another 100TB or so for indexes, giving a total hardware cost of under $200k: 400 500GB disk drives on 100 cheap PCs.

However, a public search engine requires considerably more resources than this to calculate query results and to provide high availability. And the costs of operating a large server farm are not trivial.

Crawling 10B pages with 100 machines crawling at 100 pages/second would take 1M seconds, or 11.6 days on a very high capacity Internet connection. Most search engines crawl a small fraction of the web (10-20% pages) at around this frequency or better, but also crawl dynamic web sites (e.g. news sites and blogs) at a much higher frequency

8.4.SURVEY

A survey of sample size 50 was conducted on technical and non-technical degree college students to analyse the following points:

- Use of search engines on net to gather information
- The best search engines among the popular five search engines in context to following features:
    1. Amount of information received
    2. Access time
    3. Accuracy of information
    4. Reliability
    5. User-friendly
    6. Visual appeal

- Use of effective search techniques while searching information on net.
- Awareness among the mass regarding the techniques use by search engines to list the required websites.

The survey was in the form of Questionnaire containing six questions and a suggestion for adding additional features in the search engines.

The analysis of the survey reveals the following:

- The best way to gather information is using search engines on net and 99% of the surveyed students often use them.
- The average ranking given by surveyed students to the five popular search engines(i.e.ALTAVISTA,GOOGLE,LYCOS,YAHOO,HOTMAIL) in the following features are as given below:

	Google	Yahoo	Altavista	Lycos	Hotmail
Amount of information	1	3	2	4	5
Access time	1	2	5	3	4
Acuracy of information	1	2	3	4	5
Reliability	1	2	4	3	5
User friendly	1	2	3	4	5
Visual appeal	1	3	2	4	5

- Approximately only 50% of the surveyed students are aware of the effective searching techniques.

The suggestions given by the target group for adding additional features in the search engines are:

a. Feedback mechanism
b. Help index
c. Techniques to gather reliable and accurate information
d. Techniques to obtain simultaneous additional information
e. Only required contents should be listed

8.5.Features of different Search Engines :

AltaVista

Alta Vista is a fast, powerful search engine with enough bells and whistles to do a extremely complex search, but first you have to master all its options. If you're serious about Web searching, however, mastering Alta Vista is a wise policy.

Type of search: Keyword

Search options: Simple or Advanced search, search refining.

Domains searched: Web, Usenet

Search refining: Boolean "AND," "OR" and "NOT," plus the proximal locator "NEAR." Allows wildcards and "backwards" searching (i.e., you can find all the other web sites that link to another page). You can decide how search terms should be weighed, and where in the document to look for them. Powerful search refining tools, and the more refining you do, the better your results are.

Relevance ranking: Ranks according to how many of your search terms a page contains, where in the document, and how close to one another the search terms are.

Results presented as: First several lines of document. "Detailed" summaries don't appear any more detailed than "standard" ones.

User interface: Reasonably good, but not very friendly to the casual user. Advanced query now allows you to further refine your search at the end of each results page. You can also visit specialized zones or channels in areas like finance, travel, news.

Help files: Complete, but confusing. Too much thrown at you at once. More clarity and more explanation of options would be appreciated!

Good points: Fast searches, capitalization and proper nouns recognized, largest database; finds things others don't. Alta Vista searches both the Web and Usenet. It will search on both words and on phrases, including names and titles. You can even search to discover how many people have linked their site to yours. You can also have the resulting pages of your searches translated into several other languages.

Bad points: Multiple pages from the same site show up too frequently; some curious relevancy rankings, especially on Simple search.

Overall Rating: A-

Lycos

Type of search: Keyword, but Lycos is gradually becoming less of a search engine, it seems, and more of a Yahoo-like subject index. Has recently had a cool graphical facelift. Proud of its ability to search on image and sound files.

Search options: Basic or Advanced

Domains searched: Web, Usenet, News, Stocks, Weather, Mult-media.

Search refining : Lycos now has full Boolean capabilities (using choices on drop-down forms).

Relevance ranking: Lycos no longer provides a relevancy ranking.

Results presented as: First 100 or so words in simple search, you choose in advanced search--summary, full results or short version.

User interface: Clean, clear, focuses more on directory now than on simple search.

Help files: Good, informative, graphical help screens are easy to understand.

Good points: Large database. Comprehensive results given--i.e., the date of the document, its size, etc. Lycos indexes the frequency with which documents are linked to by other documents to make sure the most popular web sites are found and indexed before the less popular ones.

Overall Rating: B+

Webcrawler

AOL owns **Webcrawler**, but AOL's new deal with Excite means that the Webcrawler search engine and directory will be incorporated into Excite.

Type of search: Keyword

Search options: Simple, refined

Search options: Domains searched: Web, Usenet

Search refining : Uses either "and" or "any." Webcrawler has added full Boolean search term capability, including AND, OR, AND NOT, ADJ, (adjacent) and NEAR.

Relevance ranking: Yes--frequency calculated--computes the total number of times your keywords appear in the document and divides it by the total number of words in the document. Webcrawler returns surprisingly relevant results.

Results presented as: lists of hyperlinks or summaries, as the user chooses.

User interface: Good--easy and fun to use

Help files: Useful tips and FAQ.

Good points: Easy to use. Popular on the Web because it belongs to AOL and there are a lot of websurfers who sign on from AOL. Publishes usage statistics on their site. Also provides a service by which you can check to see whether a particular URL is in their index, and, if so, when it was last visited by their "spider." There is also some fascinating information about how Webcrawler's search strategy works.

Bad points: Speed seems to be slowing down a little recently. Its previous weakness--no way to refine search--has been eliminated with the addition of Boolean operators.

Overall Rating: B-

HotBot

Type of search: Keyword

Search options: Simple, Modified, Expert

Domains searched: Web

Search refining: Multiple types, including by phrase, person and Boolean-like choices in pull-down boxes. No proximal operators at present. In Expert searches you can search by date and even by different media types (Java, Javascript, Shockwave, VRML, etc.).

Relevance ranking: Yes. Methods used--search terms in the title will be ranked higher search terms in the text. Frequency also counts, and will result in higher rankings when search terms appears more frequently in short documents than when they appear frequently in very long documents. (This sounds sensible and useful).

Results presented as: Relevancy score and URL

User interface: Very cool and lively. Some users have complained about the bright green background, but we kinda like it.

Help files: A FAQ that answers users' questions, but not a lot of serious help files.

Good points: Claims to be fast because of the use of parallel processing, which distributes the load of queries as well as the database over several work stations.

Bad points: Some limitations still on Boolean operators, and the help files still aren't very good.

Overall Rating: B

Yahoo

Although not precisely a search engine site, Yahoo is an important Web resource. It works as an hierarchical subject index, allowing you to drill

down from the general to the specific. Yahoo is an attempt to organize and catalogue the Web.

Yahoo also has search capabilities. You can search the Yahoo index (note: when you do this you are *not* searching the entire Web). If your query gets no hits in this manner, Yahoo offers you the option of searching the Alta Vista, which *does* search the entire Web.

Yahoo will also automatically feed your query into the other major search engine sites if you so desire. Thus, Yahoo has the capacity to act as a kind of meta-search engine.

Type of search: Keyword

Search options: Simple, Advanced

Domains searched: Yahoo's index, Usenet, E-mail addresses. Yahoo searches titles, URLs and the brief comments or descriptions of the Web sites Yahoo indexes.

Search refining: Boolean AND and OR. Yahoo is case insensitive.

Relevance ranking: Since Yahoo returns relatively few hits (it will never return more than 100), it's not clear how results are ranked.

Results presented as: Yahoo tells you the category where a hit is found, then gives you a two-line description of the site.

User interface: Excellent, easy-to-use

Help files: Not very complete, but since there aren't a lot of search options, detailed help files are not necessary.

Good points: Easy-to-navigate subject catalogue. If you know what you want to find, Yahoo should be your first stop on the Web.

Bad points: Only a small portion of the Web has actually been catalogued by Yahoo.

Overall rating: A (This rating refers simply to Yahoo's quality as a directory--searches of the entire Web are not possible).

8.6 GOOGLE SEARCH ENGINE

GOOGLE *is one of the popular and most widely used and efficient search engines.Hence here we are discussing the various policies used by Google in web crawling,indexing and ranking.*

GOOGLE ARCHITECTURE

Most of Google is implemented in C or C++ for efficiency and can run in either Solaris or Linux.

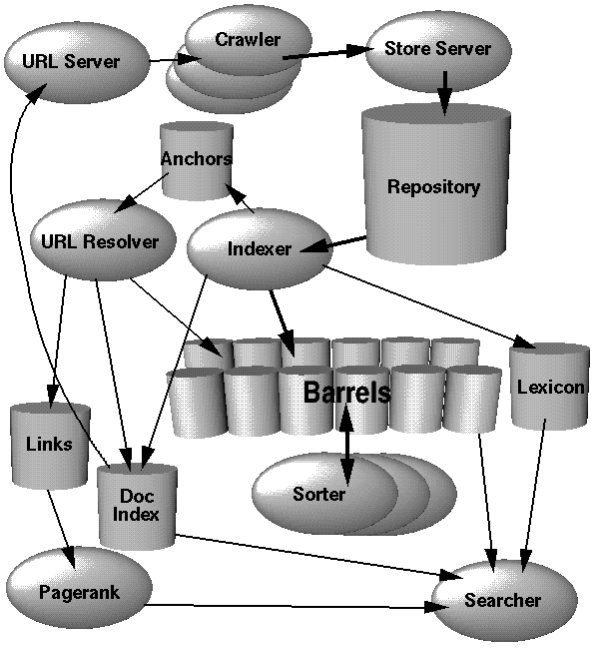


Figure 1. High Level Google Architecture

A. **Crawlers:** In Google, the web crawling (downloading of web pages) is done by several distributed crawlers.
B. URL Servers: The URLserver sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the **storeserver.**
C. **Store Server:** The storeserver compresses and stores the web pages into a repository. Every web page has an associated ID number called a **docID** which is assigned whenever a new URL is parsed out of a web page.

D. **Indexer:** The indexing function is performed by the indexer and the sorter. 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.

E. **URL Resolver:** The URLResolver reads the Anchor files and in turn converts relative URL into absolute URL and then into docIDs. It puts the anchor text into the forward index, associated with the docID that the anchor points to. It also generates a database of links which are pairs of docIDs. The links database is used to compute PageRanks for all the documents.

F. **Sorter:** The sorter takes the barrels, which are sorted by docID and resorts them by wordID 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 wordIDs and offsets into the inverted index.

G. **Lexicon:** A program called **DumpLexicon** takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. The searcher is run by a web server and uses the lexicon built by DumpLexicon together with the inverted index and the PageRanks to answer queries.

GOOGLE CRAWLER

In order to scale to hundreds of millions of web pages, Google has a fast distributed crawling system. A single URLserver serves lists of URLs to a number of crawlers (we typically ran about 3). Both the URLserver and the crawlers are implemented in Python. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve web pages at a fast enough pace. At peak speeds, the system can crawl over 100 web pages per second using four crawlers. This amounts to roughly 600K per second of data. A major performance stress is DNS lookup. Each crawler maintains a its own DNS cache so it does not need to do a DNS lookup before crawling each document. Each of the hundreds of connections can be in a number of different states: looking up DNS, connecting to host, sending request, and receiving response. These factors make the crawler a complex component of the system. It uses asynchronous IO to manage events, and a number of queues to move page fetches from state to state.

GOOGLE DATA STRUCTURES

Google's data structures are optimized so that a large document collection can be crawled, indexed, and searched with little cost. Although, CPUs and bulk input output rates have improved dramatically over the years, a disk seek still requires about 10 ms to complete. Google is designed to avoid disk seeks whenever possible, and this has had a considerable influence on the design of the data structures.

The data structures maintained by Google are as follows:

1. **BigFiles:**

   BigFiles are virtual files spanning multiple file systems and are addressable by 64 bit integers. The allocation among multiple file systems is handled automatically. The BigFiles package also handles allocation and deallocation of file descriptors, since the operating systems do not provide enough for our needs. BigFiles also support rudimentary compression options.

2. **Repository:** The repository contains the full HTML of every web page. Each page is compressed using zlib (see RFC1950). 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 URLThe repository requires no other data structures to be used in order to access it. This helps with data consistency and makes development much easier; we can rebuild all the other data structures from only the repository and a file which lists crawler errors.

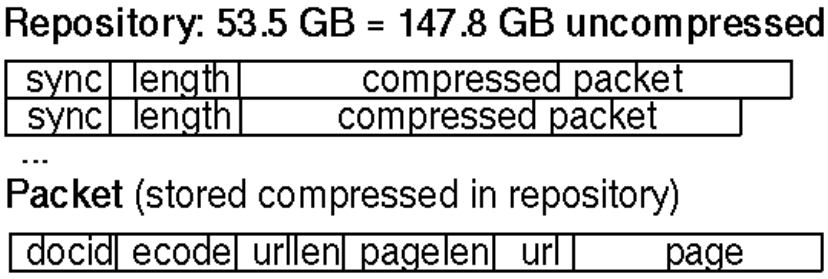


Figure 1Repository

3. **Document Index:**

The document index keeps information about each document. It is a fixed width ISAM (Index sequential access mode) index, ordered by docID. 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 docinfo which contains its URL and title. Otherwise the pointer points into the URLlist which contains just the URL. This design decision was driven by the desire to have a reasonably compact data structure, and the ability to fetch a record in one disk seek during a search

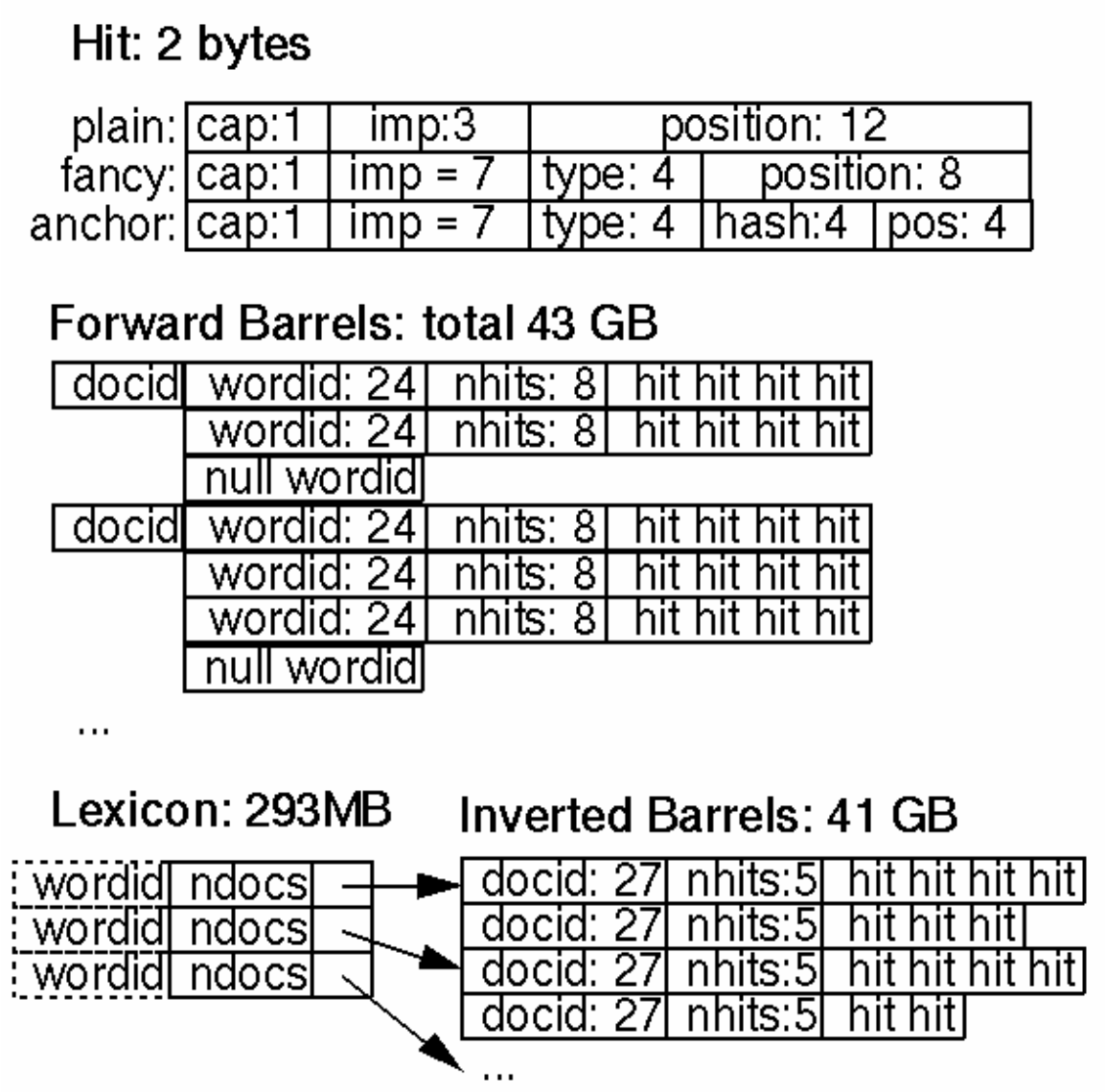
Additionally, there is a file which is used to convert URLs into docIDs. It is a list of URL checksums with their corresponding docIDs and is sorted by checksum. In order to find the docID of a particular URL, the URL's checksum is computed and a binary search is performed on the checksums file to find its docID. URLs may be converted into docIDs in batch by doing a merge with this file. This is the technique the URLresolver uses to turn URLs into docIDs. This batch mode of update is crucial because otherwise we must perform one seek for every link which assuming one disk would take more than a month for our 322 million link dataset.

4. Lexicon:

The lexicon has several different forms. One important change from earlier systems is that the lexicon can fit in memory for a reasonable price. It is implemented in two parts -- a list of the words (concatenated together but separated by nulls) and a hash table of pointers. For various functions, the list of words has some auxiliary information

5. 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. We considered several alternatives for encoding position, font, and capitalization -- simple encoding (a triple of integers), a compact encoding (a hand optimized allocation of bits), and Huffman coding. In the end we chose a hand optimized compact encoding since it required far less space than the simple encoding and far less bit manipulation than Huffman coding.



compact encoding uses two bytes for every hit. 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 (all positions higher than 4095 are labeled 4096). Font size is represented relative to the rest of the document using three bits (only 7 values are actually used because 111 is the flag that signals a fancy hit). 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. For anchor hits, the 8 bits of position are split into 4 bits for position in anchor and 4 bits for a hash of the docID the anchor occurs in. This gives us some limited phrase searching as long as there are not that many anchors for a particular word. We expect to update the way that anchor hits are stored to allow for greater resolution in the position and docIDhash fields. We use font size relative to the rest of the document because when searching, you do not want to rank otherwise identical documents differently just because one of the documents is in a larger font. The length of a hit list is stored before

the hits themselves. To save space, the length of the hit list is combined with the wordID in the forward index and the docID in the inverted index. This limits it to 8 and 5 bits respectively (there are some tricks which allow 8 bits to be borrowed from the wordID). If the length is longer than would fit in that many bits, an escape code is used in those bits, and the next two bytes contain the actual length.

6. Forward Index:

The forward index is actually already partially sorted. It is stored in a number of barrels (we used 64). Each barrel holds a range of wordID's. If a document contains words that fall into a particular barrel, the docID is recorded into the barrel, followed by a list of wordID's with hitlists which correspond to those words. This scheme requires slightly more storage because of duplicated docIDs but the difference is very small for a reasonable number of buckets and saves considerable time and coding complexity in the final indexing phase done by the sorter. Furthermore, instead of storing actual wordID's, we store each wordID as a relative difference from the minimum wordID that falls into the barrel the wordID is in. This way, we can use just 24 bits for the wordID's in the unsorted barrels, leaving 8 bits for the hit list length.

7. Inverted Index:

The inverted index consists of the same barrels as the forward index, except that they have been processed by the sorter. For every valid wordID, the lexicon contains a pointer into the barrel that wordID falls into. It points to a doclist of docID's together with their corresponding hit lists. This doclist represents all the occurrences of that word in all documents.

An important issue is in what order the docID's should appear in the doclist. One simple solution is to store them sorted by docID. This allows for quick merging of different doclists for multiple word queries. Another option is to store them sorted by a ranking of the occurrence of the word in each document. This makes answering one word queries trivial and makes it likely that the answers to multiple word queries are near the start. However, merging is much more difficult. Also, this makes development much more difficult in that a change to the ranking function requires a rebuild of the index. We chose a compromise between these options, keeping two sets of inverted barrels -- one set for hit lists which include title or anchor hits and another set for all hit lists. This way, we check the first set of barrels first and if there are not enough matches within those barrels we check the larger ones.

GOOGLE INDEXING

- **Parsing --** Any parser which is designed to run on the entire Web must handle a huge array of possible errors. These range from typos in HTML tags to kilobytes of zeros in the middle of a tag, non-ASCII characters, HTML tags nested hundreds deep, and a great variety of other errors that challenge anyone's imagination to come up with equally creative ones. For maximum speed, instead of using YACC to generate a CFG parser, we use flex to generate a lexical analyzer which we outfit with its own stack. Developing this parser which runs at a reasonable speed and is very robust involved a fair amount of work.
- **Indexing Documents into Barrels --** After each document is parsed, it is encoded into a number of barrels. Every word is converted into a wordID by using an in-memory hash table -- the lexicon. New additions to the lexicon hash table are logged to a file. Once the words are converted into wordID's, their occurrences in the current document are translated into hit lists and are written into the forward barrels. The main difficulty with parallelization of the indexing phase is that the lexicon needs to be shared. Instead of sharing the lexicon, we took the approach of writing a log of all the extra words that were not in a base lexicon, which we fixed at 14 million words. That way multiple indexers can run in parallel and then the small log file of extra words can be processed by one final indexer.
- **Sorting** -- In order to generate the inverted index, the sorter takes each of the forward barrels and sorts it by wordID to produce an inverted barrel for title and anchor hits and a full text inverted barrel. This process happens one barrel at a time, thus requiring little temporary storage. Also, we parallelize the sorting phase to use as many machines as we have simply by running multiple sorters, which can process different buckets at the same time. Since the barrels don't fit into main memory, the sorter further subdivides them into baskets which do fit into memory based on wordID and docID. Then the sorter, loads each basket into memory, sorts it and writes its contents into the short inverted barrel and the full inverted barrel

GOOGLE RANKING SYSTEM

Google maintains much more information about web documents than typical search engines. Every hitlist includes position, font, and capitalization information Additionally, Google factors in hits from anchor text and the PageRank of the document. Combining all of this information into a rank is difficult. Google’s ranking function does not allow a particular factor to have too much influence. First, consider the simplest case -- a single word query. In order to rank a document with a single word query, Google looks at that document's hit list for that word. Google considers each hit to be one of several different types (title, anchor, URL, plain text large font, plain text small font, ...), each of which has

its own type-weight. The type-weights make up a vector indexed by type. Google counts the number of hits of each type in the hit list. Then every count is converted into a count-weight. Count-weights increase linearly with counts at first but quickly taper off so that more than a certain count will not help. Then Google takes the dot product of the vector of count-weights with the vector of type-weights to compute an IR score for the document. Finally, the IR score is combined with PageRank to give a final rank to the document.

For a multi-word search, the situation is more complicated. Now multiple hit lists must be scanned through at once so that hits occurring close together in a document are weighted higher than hits occurring far apart. The hits from the multiple hit lists are matched up so that nearby hits are matched together. For every matched set of hits, a proximity is computed. The proximity is based on how far apart the hits are in the document (or anchor) but is classified into 10 different value "bins" ranging from a phrase match to "not even close". Counts are computed not only for every type of hit but for every type and proximity. Every type and proximity pair has a type-prox-weight. The counts are converted into count-weights and the dot product of the count-weights and the type-prox-weights is used to compute an IR score. All of these numbers and matrices can all be displayed with the search results using a special debug mode. These displays have been very helpful in developing the ranking system.

GOOGLE QUERY EVALUATION

1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4.
   Sort the documents that have matched by rank and return the top k.

STORAGE REQUIREMENTS

Google is designed to scale cost effectively to the size of the Web as it grows. One aspect of this is to use storage efficiently. Table 1 has a breakdown of some statistics and storage requirements of Google. Due to compression the total size of the repository is about 53 GB, just over one third of the total data it stores. At current disk prices this makes the repository a relatively cheap source of useful data. More importantly, the total of all the data used by the search engine requires a comparable amount of storage, about 55 GB. Furthermore, most queries can be answered using just the short inverted index. With better encoding and compression of the Document Index, a high quality web search engine may fit onto a 7GB drive of a new PC.

Storage Statistics	
Total Size of Fetched Pages	147.8 GB
Compressed Repository	53.5 GB
Short Inverted Index	4.1 GB
Full Inverted Index	37.2 GB
Lexicon	293 MB
Temporary Anchor Data (not in total)	6.6 GB
Document Index Incl. Variable Width Data	9.7 GB
Links Database	3.9 GB
Total Without Repository	**55.2 GB**
Total With Repository	**108.7 GB**

Web Page Statistics	
Number of Web Pages Fetched	24 million
Number of Urls Seen	76.5 million
Number of Email Addresses	1.7 million
Number of 404's	1.6 million

8.7 CONCLUSION

Thus we have conducted a detailed study of various search engines .The survey clearly shows popularity and wide use of GOOGLE.Hence we have selected GOOGLE as our backend search engine and conducted a detailed study of its working.

9.ANALYSIS

9.1. REQUIREMENT ANALYSIS

The process of establishing the services the system to be developed must provide and the constraints under which it must operate is called Requirements and capture and analysis.

Requirement analysis is a software engineering tasks that bridges the gap between system level requirements engineering and software design.Requirements engineering activities result in the specification of software's operational characteristics (function, data , behaviour),indicates software's interface with other system elements and establish constraints that software must meet.
Analysis is done using the basic diagram like data flow diagram.

Various databases used are:

- **URL server:**This is the server which stores a list of URLs that are to be crawled by the crawler.It also assigns a DOCID to each URL
- **Store server:**This is the server which stores the crawled webpages,forward index and reverse index.
- **Lexicon:**This is the database which stores list for unique tokens(words) and the WORDID for each token.
- **Forward index:**This index stores the DOCID and the list of WORDIDs corresponding to the words in the document with information regarding each word like its location,font size,font type and other features like whether it is bold or underlined which gives importance of the word in the document.
- **Reverse index:**This is prepared from forward index.It includes each WORDID and the list of DOCIDs corresponding to the documentswhich contains the word with the given WORDID along with the specific information.

Various terms(programs) in search engine are:

1.Extractor: This is a continuous running program in the system which extracts URL and stores them in URL server.There are 3 ways to add a URL

- Website developer registers the site
- Links from other sites
- Feedback from seed.

2.Crawler: This is a search engine program which obtains URL from the URL server,scans the rules developed by the web master for that website and downloads the page if permission is granted and stores it in store server.It then scans the animation sites in the webpage to get information from them.

3.Indexer: This is a search engine program which obtains pages from store server and scans these pages to extract tokens.It gives each token a word id if it is a new token and stores it in lexicon.For each web page a doc id is assigned.Indexer prepares forward index and stores it in store server.

4.Sorter: This is also a search engine program which runs continuously.It obtains forward indices from store server and prepares reverse index which will be used by searcher and stores it in store server.

5.Searcher:Searcher: is the program which is invoked when a user submits a query.The searcher searches for the word id in the lexicon and then using the word id searches for the list of URLs in the reverse index stored in store server.

6.Validation: This program is invoked when user submits the query.It checks for spelling mistakes and also if the word used in query is unknown then gives suggestion regarding related words.

9.1.1.FUNCTIONAL MODELLING:

DATA FLOW DIAGRAM

Information is transformed as it flows through a computer based system. The system accepts input in a variety of forms and produces output in varied forms. For depicting this information flow in functional modeling, Data Flow Diagram (DFDs) are used.
A Data Flow Diagram is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output.
The data flow diagram may be used to represent the system or software at any level of abstraction.DFDs may be partitioned into levels that represent increasing information flow and functional details.the DFD provides mechanism for functional modeling as well as information flow modeling.

DFD's are generally done in 3 steps starting from a basic LEVEL 0 to LEVEL 2,each LEVEL giving more details than the previous one.

LEVEL 0 DFD

A level 0 DFD,also called a fundamental system model or a context model,represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows,respectively.

LEVEL 1 DFD

Additional processes(bubbles) and information flow paths are represented as level 0 DFD is partitioned to reveal more detail.

LEVEL 2 DFD

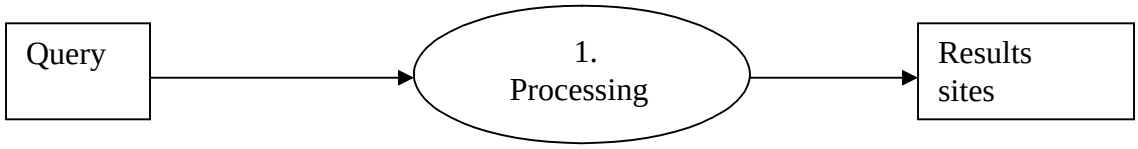
LEVEL 1 DFD is then expanded to reveal more details and this forms Level 2 DFD.

The search engine performs various function which includes the most important USER function of searching related sites,crawling the various websites and indexing and sorting these websites.These functions all exhibit a different flow of information.Hence to reveal all these information flows following 3 DFD's have been drawn.

USER DFD

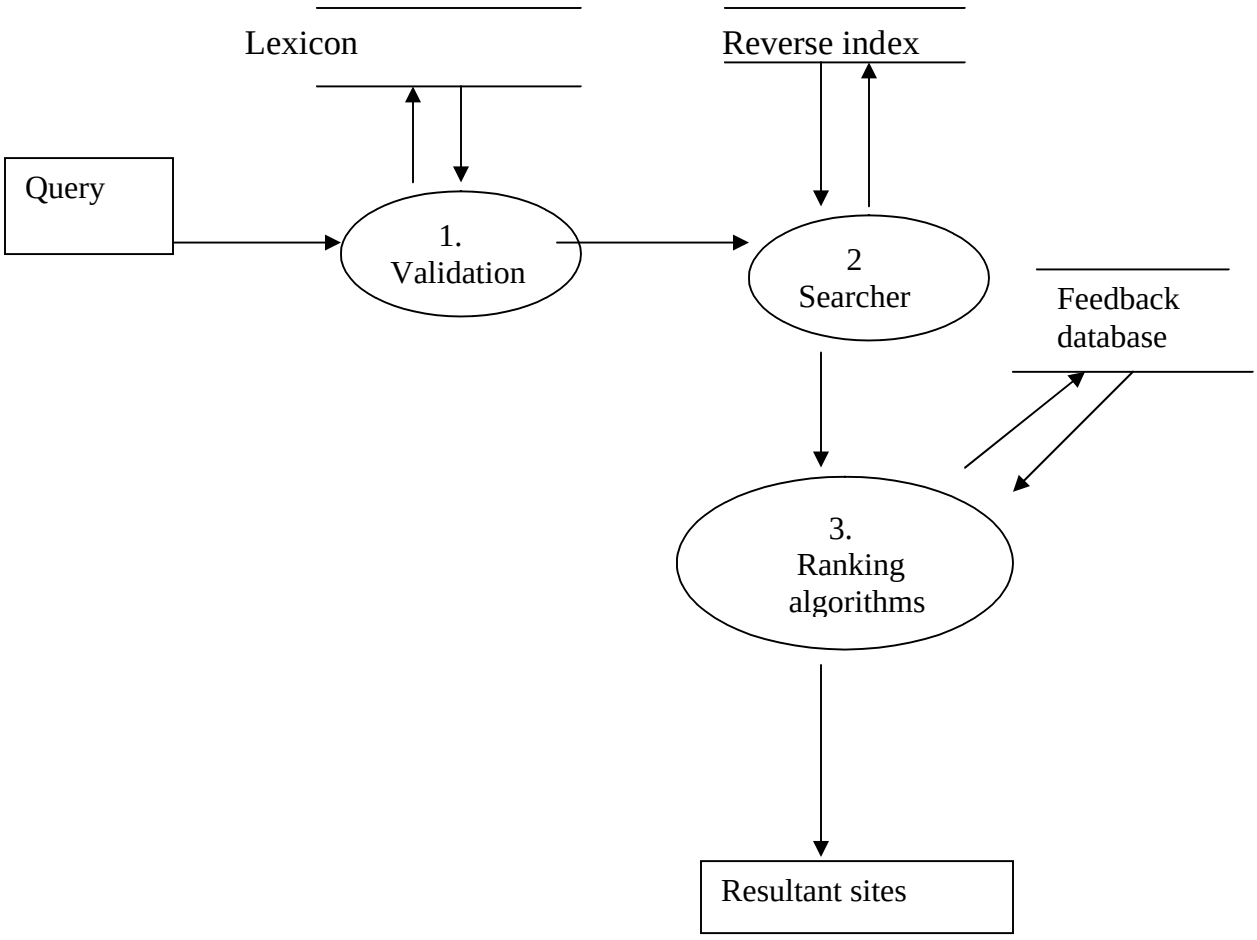
Level 0:

This is the basic DFD for user which shows that the input to search engine is the query submitted by the user which is processed by the system and output is the resultant sites.



Level 1:

The LEVEL 0 DFD shown above is expanded to form LEVEL 1 DFD.
The input and output remains the same but the process is broken down.The query is first sent for validation which refers the database lexicon.after validation the query is given to a searcher which on referring the Reverse index sends the result that is the list of websites to the ranking algorithm which after referring the feedback database ranks the website in a specific order of importance and provides the output.

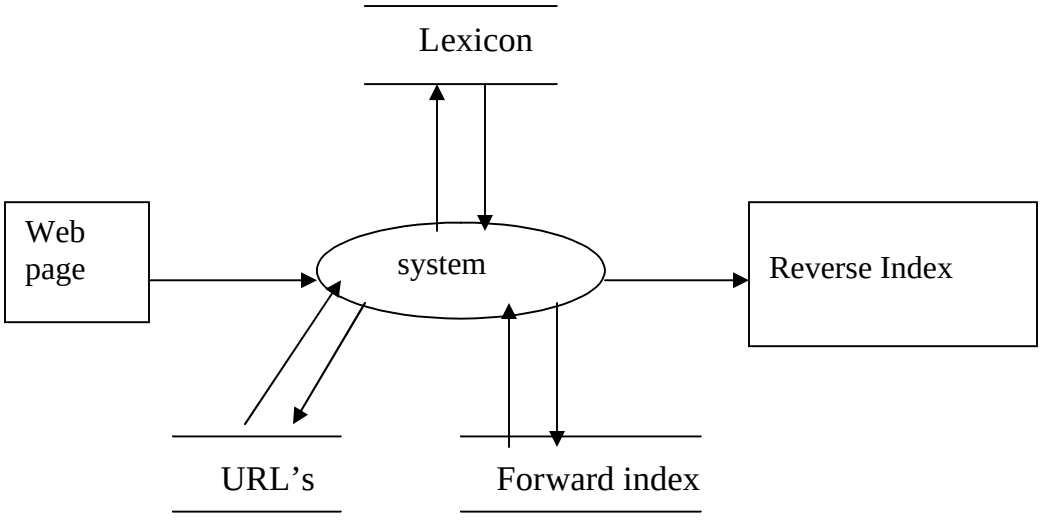


SYSETM DFD:

The another important function of the system is to keep crawling,indexing and sorting the websites.This is a function which keeps running continuously without being triggered by the user.

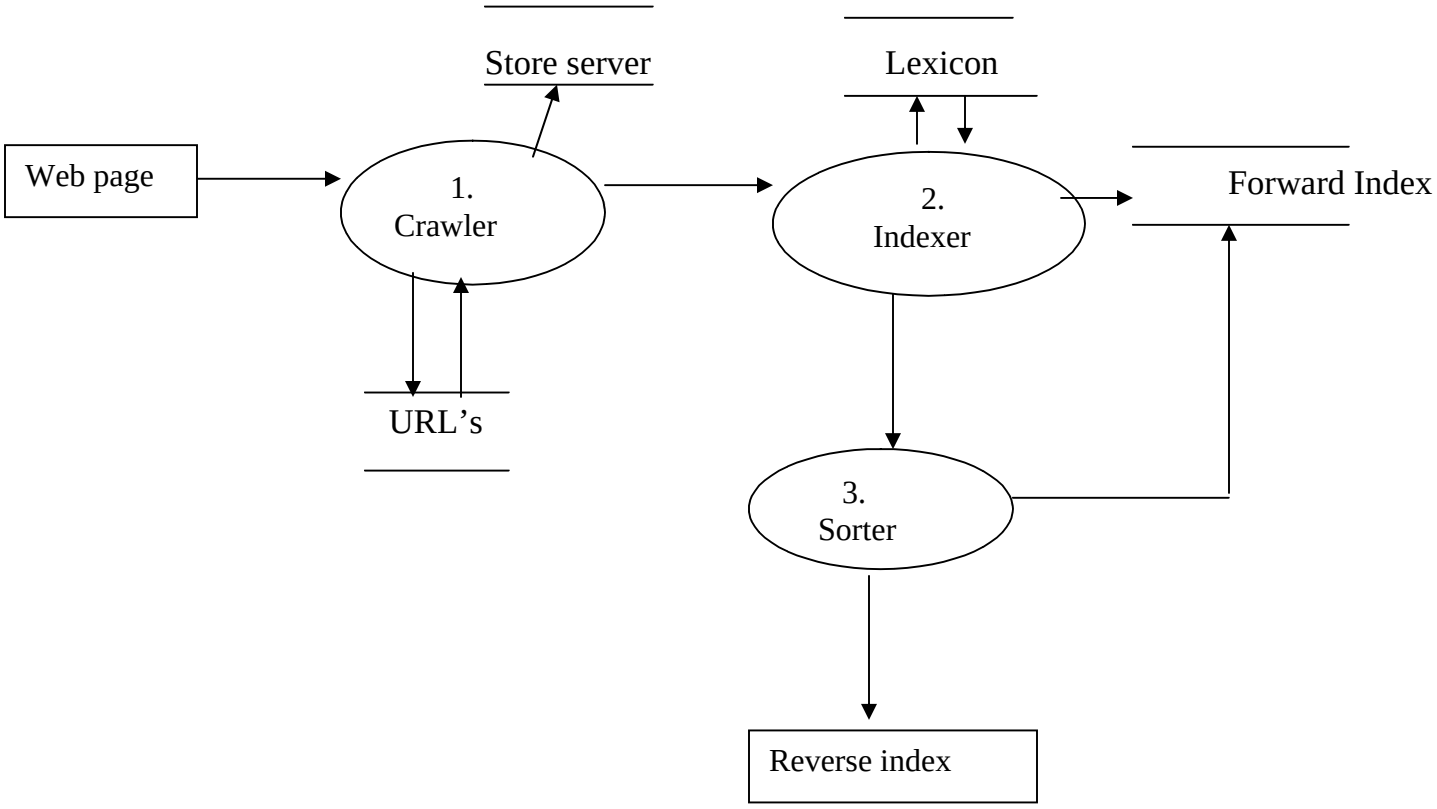
Level 0:

These show the basic functionality information flow in the system.The webpage extracted from the URL server is submitted to the system which on referring databases like lexicon,URL's and forward index generates a reverse index which is reffered by the searcher while providing results to the user.



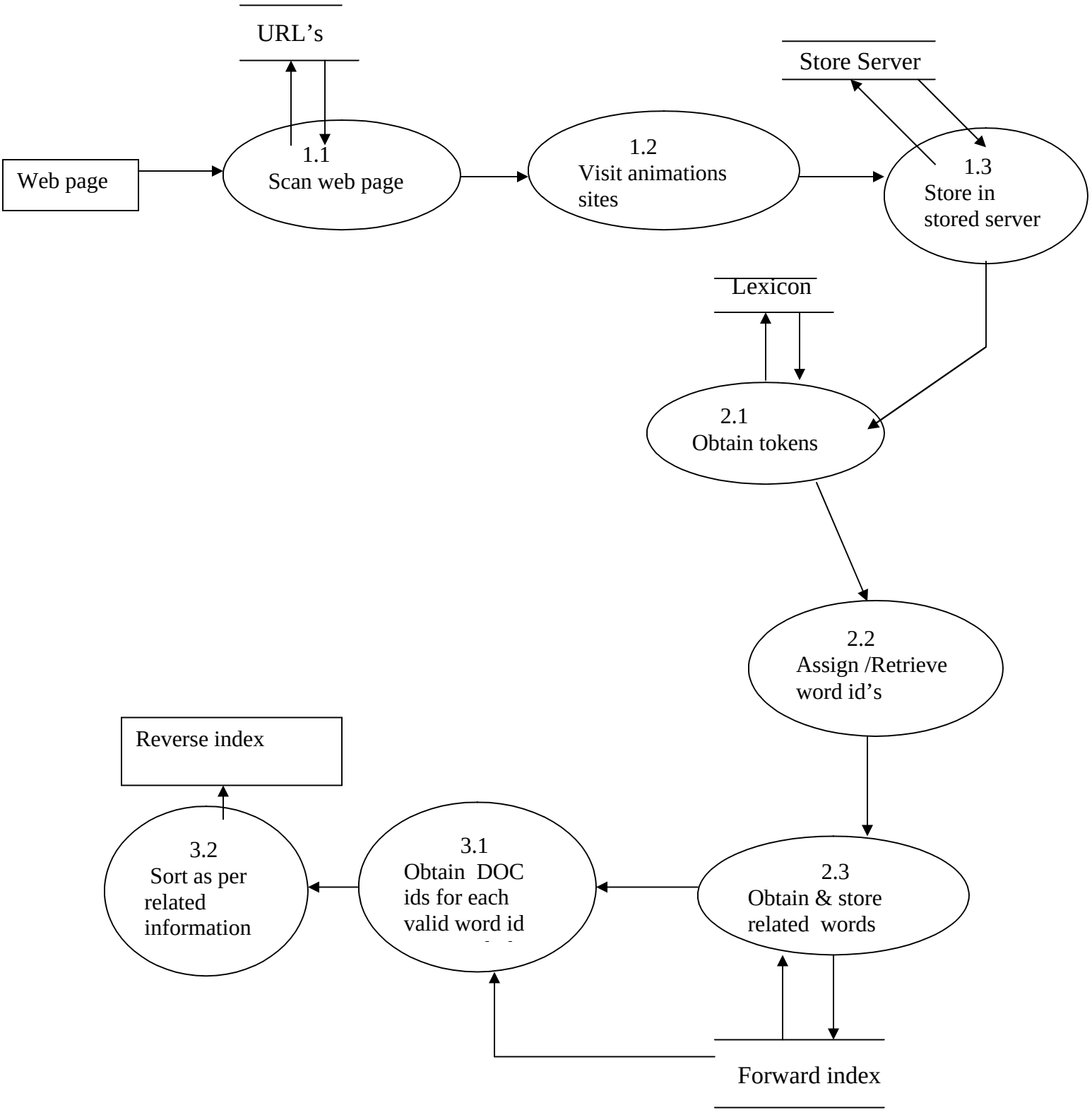
Level 1:

The process system is now broken down into various other processes in the LEVEL 2 DFD.The webpage is now given to the crawler which checks for the webpage accesses and if the access is given for a particular website then downloads the webpage and stores it in store server.The crawler also passes the webpage to the indexer.The indexer now refers to the lexicon for the words it scans in the webpage and then adds new words in the lexicon.It now prepares forward index .This forward index is now used by the sorter to give the reverse index.



Level 2:

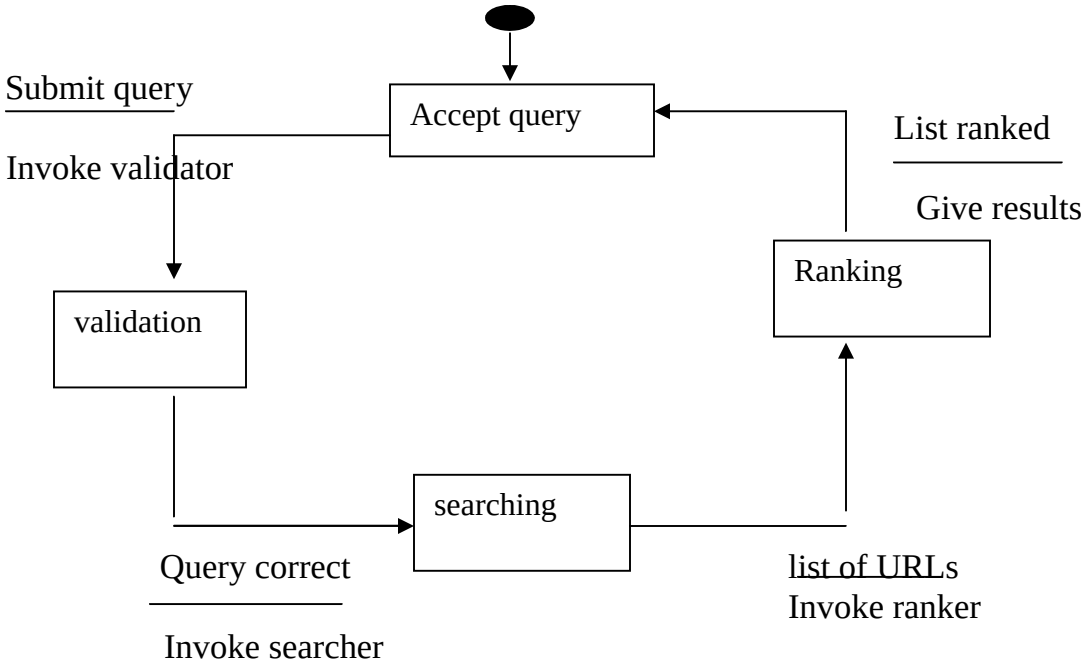
The processes are in explained in more detail in LEVEL 2.The webpage is first scanned and URL database is referred. The animation sites in the given webpage is then listed and scanned for metatags. These sites are stored in store server. The tokens from the webpage are then obtained and added to lexicon if it is new token. The word id's are assigned to each tokens. The words are then stored in forward index along with various details like its location, font size and importance like underlined ,bold etc. The word id's are now considered one by one and docid's for each word id is retrieved from the forward index to prepare reverse index which stores word id and a sorted list of doc id's which contains the given word id.


URL's
Store Server
Web page
1.1
Scan web page
1.2
Visit animations
sites
1.3
Store in
stored server
Lexicon
2.1
Obtain tokens
2.2
Assign /Retrieve
word id's
Reverse index
3.2
Sort as per
related
information
3.1
Obtain DOC
ids for each
valid word id
2.3
Obtain & store
related words
Forward index

9.1.2.BEHAVORAL MODELING

Behavioral modeling is an operational principle for all requirements analysis methods. Yet only extended versions of structured analysis provide a notation for this type of modeling. The state transition diagram (STD) represents the behavior of a system by depicting its states and events that cause the system to change state. In addition, STD indicates what actions are taken as a consequence of a particular event.

The state transition diagram for search engine is as given follows.-



9.2. OBJECT ORIENTED ANALYSIS:

Analysis is an important part of system development.object oriented analysis adds more object oriented features to the standard analysis procedure.It provides flexibility,encapsulation,abstraction and most importantly reusability.

Various Unified Modeling Language(UML) diagrams are used for carrying out object oriented analysis.

UML diagrams like use case,sequence,activity,collaboration are used for analysis.

9.2.1.USE CASE DIAGRAM

Use Case Diagram is the basic analysis diagram.It is the first analysis diagram.it gives interaction of the system with entities outside the system.These entities are called actors.Actors are the users of the system or other systems who give input to the system and take output from the system.The various scenarious in the system are called usecases.They are denoted by oval.

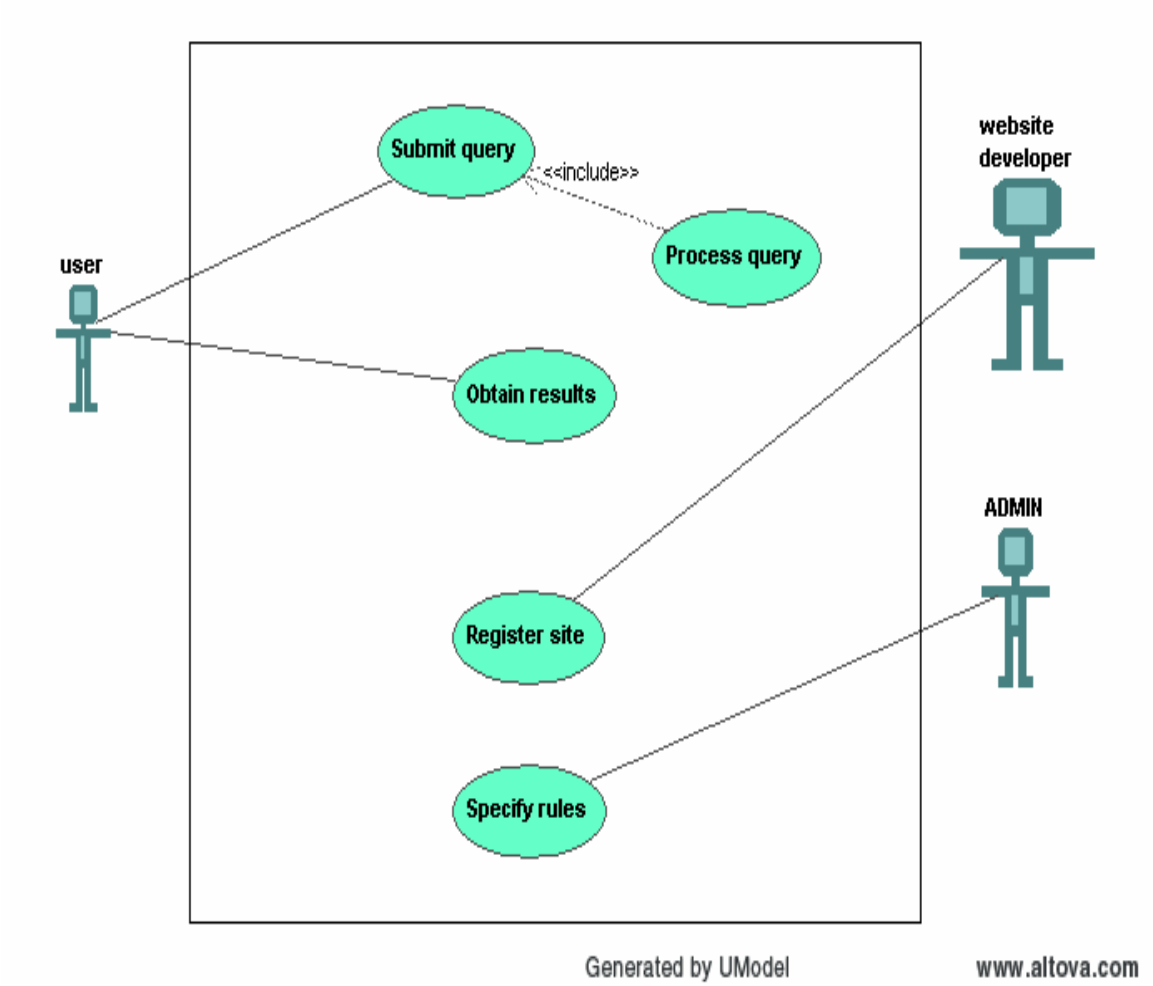
The complete system has been explained using a basic use case diagram which shows the interaction between the main actors and a set of use case scenarios.
The detailed use case diagrams following the basic use case diagram gives the detailed interaction of each actor with the system.

BASIC USE CASE

The following basic use case diagram is shown below.The main actors are

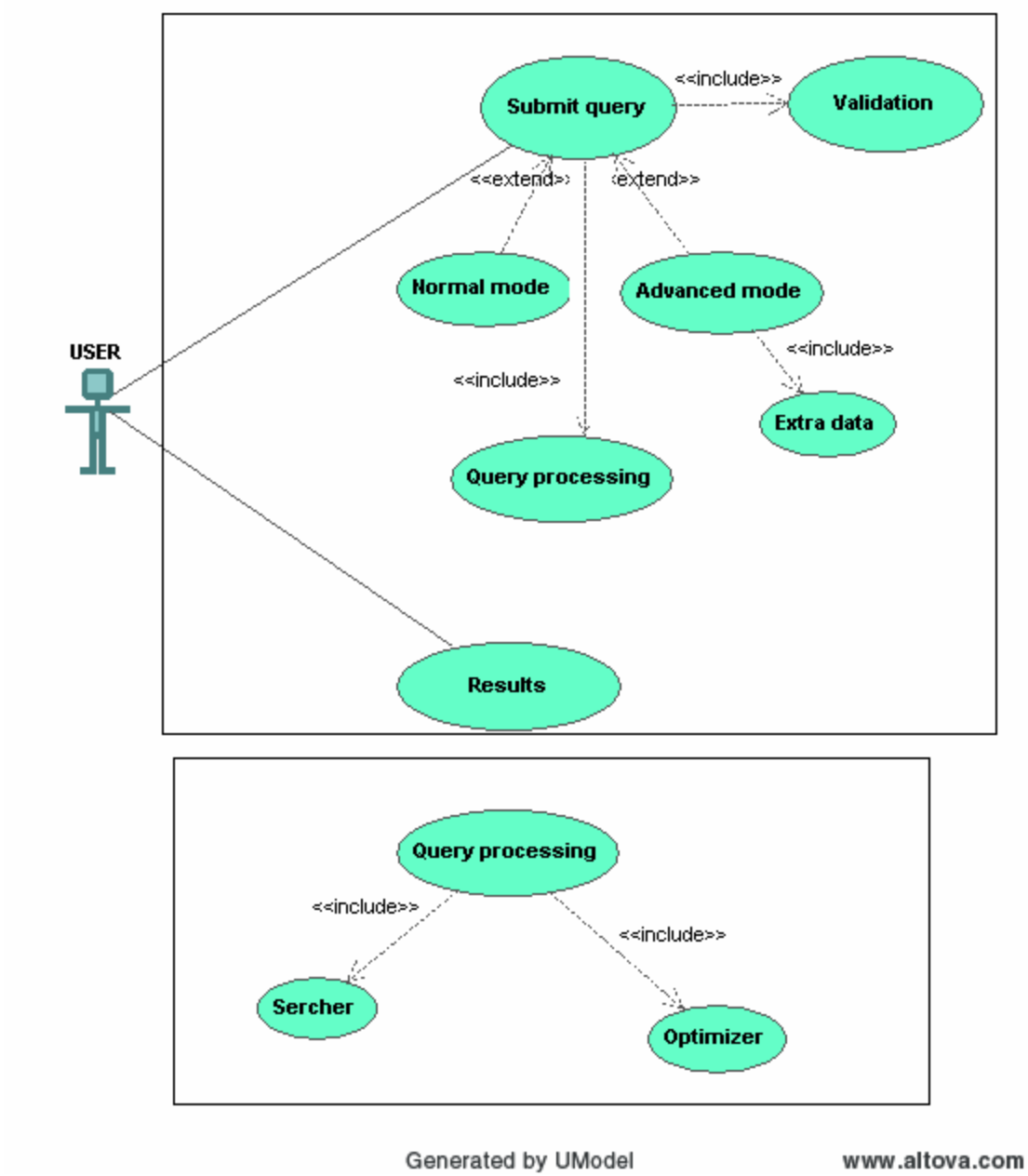
- User
- Website developer
- Administrator(website)

User submits query in the system which is processed by the system.
The user also obtains results from the system which is the list of websites containing the required animation.
The website developer is required to register his website with the search engine in order to enable the crawler to locate the site.
The website administrator specifies certain rules with the website with are taken into consideration by the crawler while crawling a particular website.These information includes various webpage access constraints.



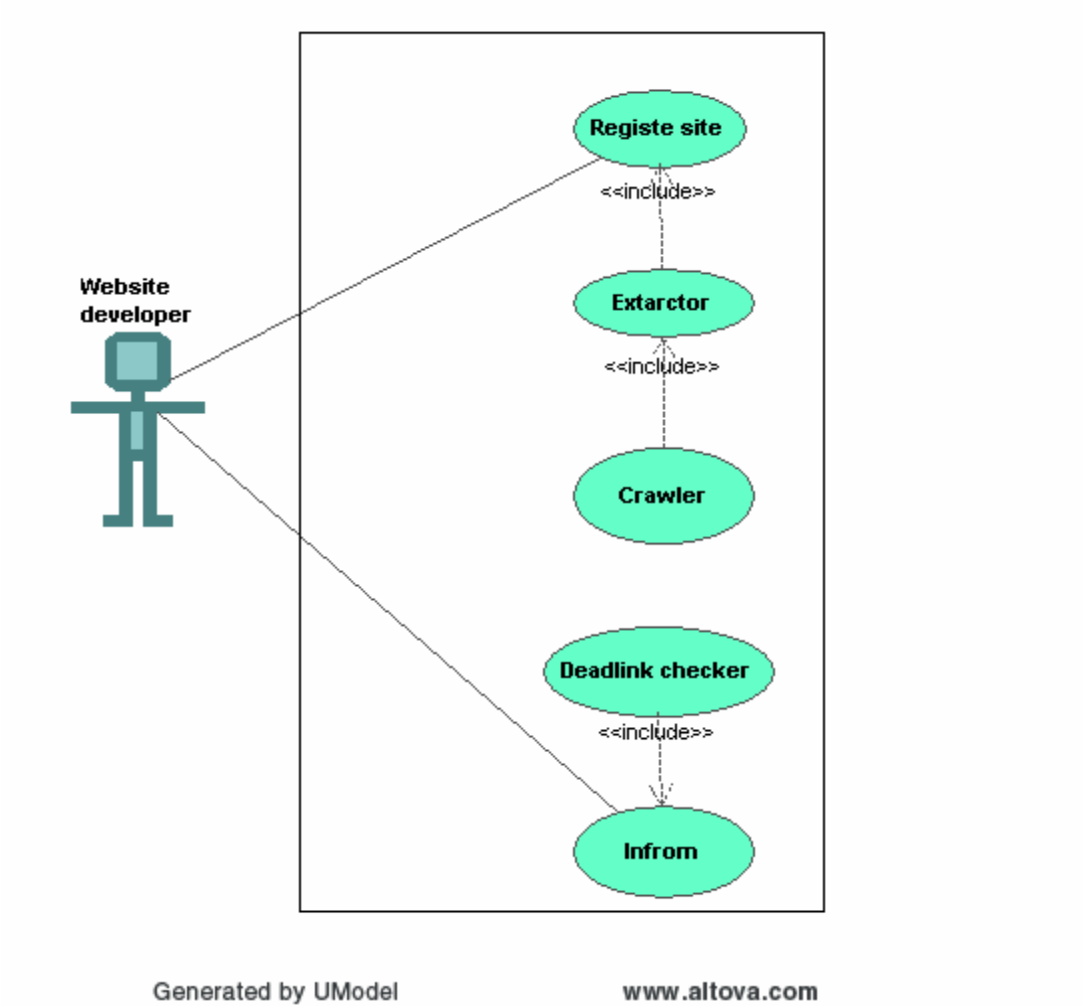
USER USE CASE VIEW

The following is the detailed use case view for showing the user's interaction with the system.User is the actor which submits query in the system to search animation for a particular topic.User can choose the mode of submission.The query can be processed either in normal mode or in advanced mode.Advanced mode allows using proximity search as well as specific file search.Submit query usecase also includes validation.Validation checks for spellings and suggestions regarding similar querries.The user is provided with the results after processing the query.


Submit query
<<include>>
Validation
<<extend>>
<extend>>
Normal mode
Advanced mode
USER
<<include>>
<<include>>
Extra data
Query processing
Results
Query processing
<<include>>
<<include>>
Sercher
Optimizer
Generated by UModel
www.altova.com

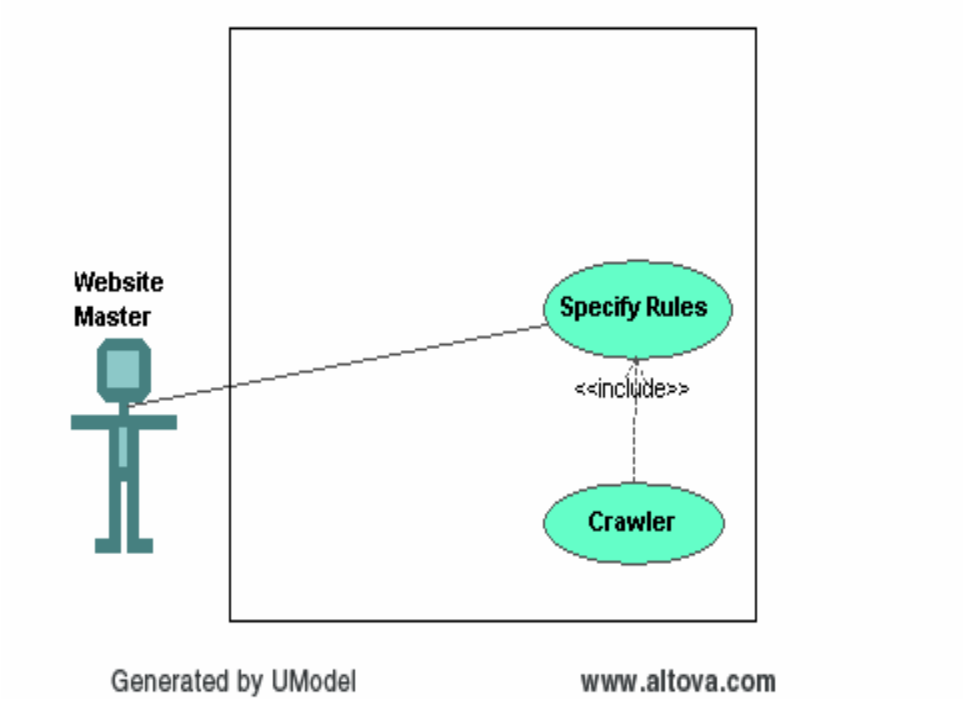
WEB DEVELOPER

The different aspect of the system is with respect to the website developer.The website developer has to register his website with the search engine so that it is crawled and shows up in the result of the query.
The search engine has a special tool called extractor which adds the URL to the URL server.One of the methods to add the URL is through registration by the website developer.The crawler then accepts a URL from the server.It checks for the specified rules for the website and then scans and downloads the page.
The system also includes a deadlink checker program which periodically checks for deadlinks in a webpage.If a deadlink is observed constantly then the particular website developer is informed.



WEB MASTER USE CASE VIEW

The web master or administrator is the actor which specifies various rules for a particular website which includes access information.One such file is robot.txt.This file is first scanned by the crawler and various permissions are checked before downloading a website.



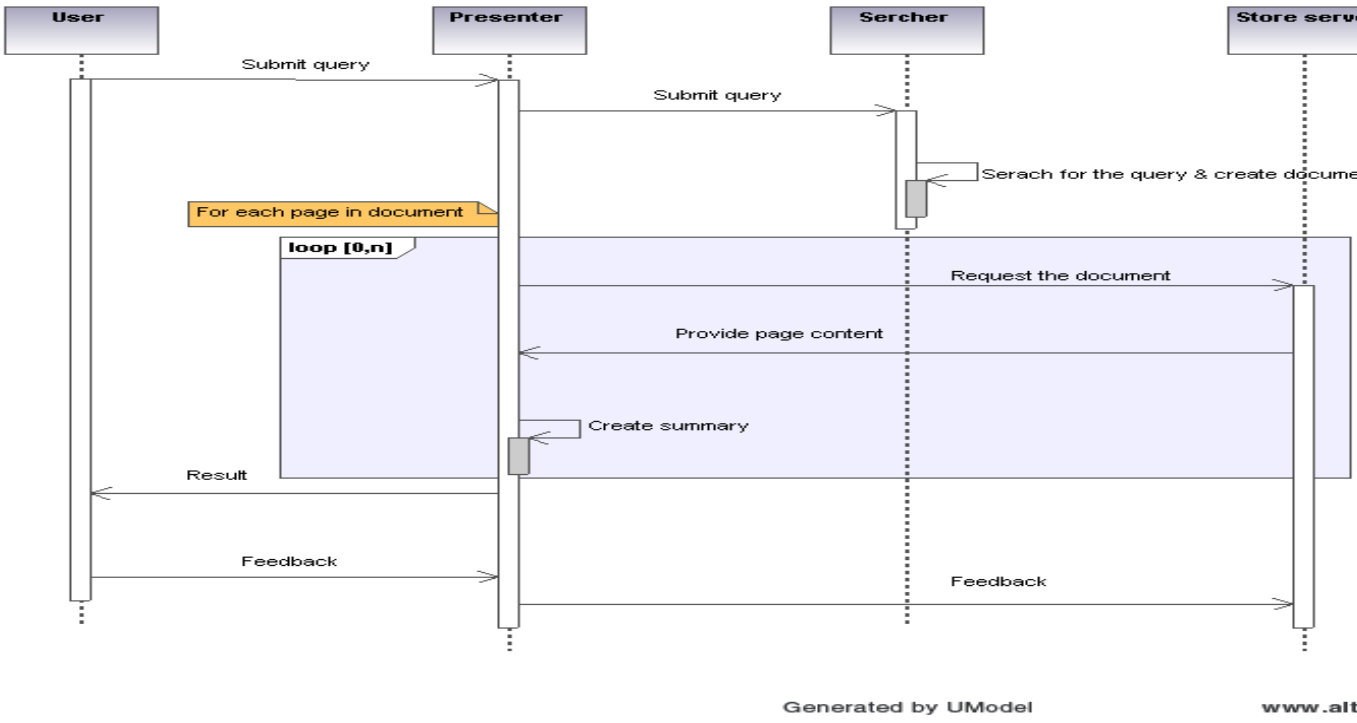
9.2.2. SEQUENCE DIAGRAM

These are the analysis diagram which gives the sequence of events taking place in the system.The diagram gives the objects in the system and their interaction among each other.The diagram also shows the messages passed between the objects.

There are many views of the system and each is described by a different sequence diagram.

QUERY PROCESSING

The main view of the system is query processing.This is the main function of the search engine with respect to the user.The main objects involved in this function are user,presenter,searcher and store server.
The user submits a query which is given to the presenter which in turn submits the query to the searcher.The searcher searches for the keywords and creates a document list which gives the address of all the URL's related to the mentioned keywords.
The presenter then requests the store server to return document at each URL.It then creates a summary for each URL and presents it to the user in the form of a list of URL's along with their summary.
The user can also provide feedback for each website which can then be used for ranking.

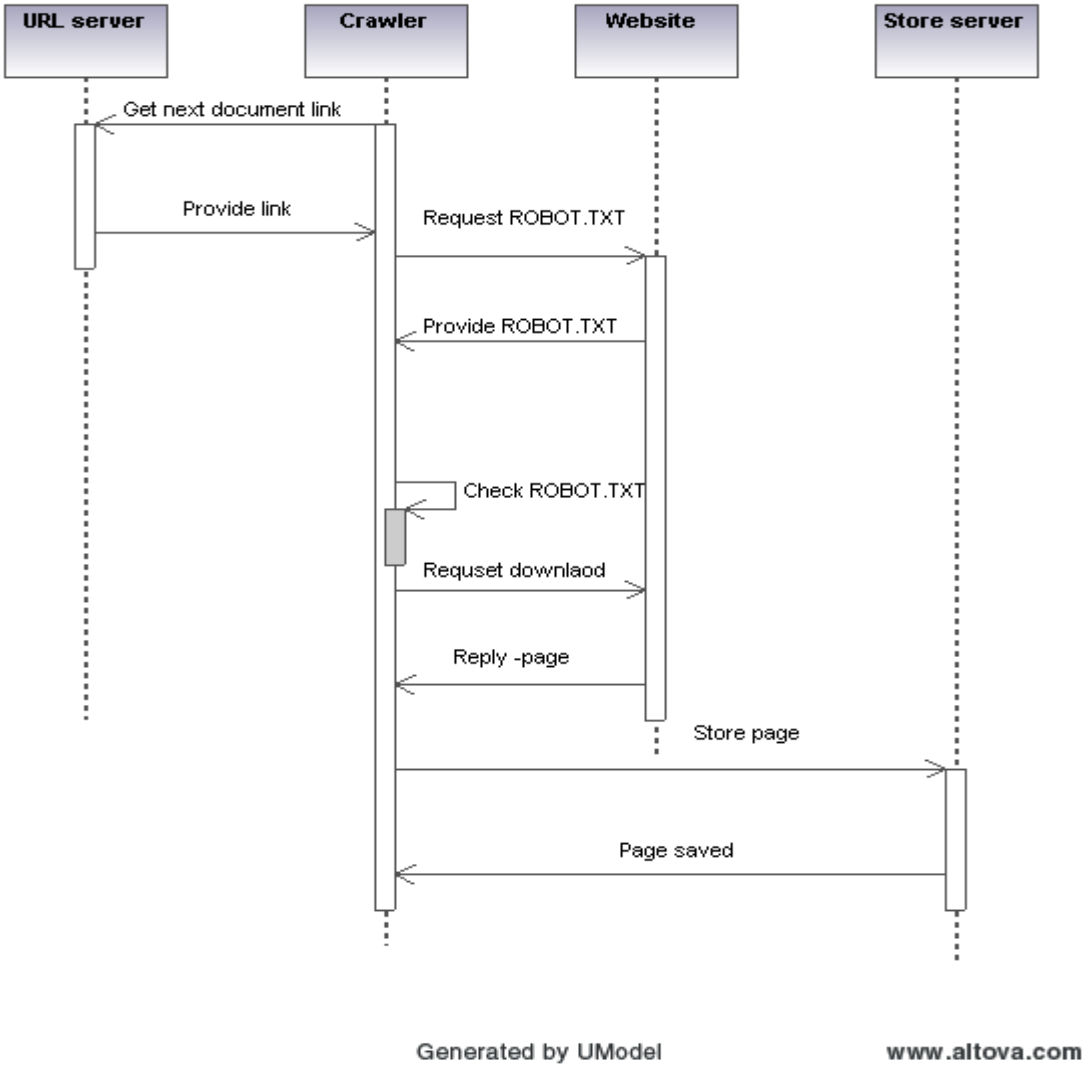


CRAWLER

The search engine has a continuously running program called crawlers which indexes the websites related to animation.The main working of search engine crawler includes various objects like URL server,crawler,website and store server.

The crawler requests the URL server to provide next URL to be crawled which is then provided.The crawler then requests the file robot.txt to the website which it then scans for

access permissions for the webpages. The crawler then depending on access permission requests the webpages and stores them in the store server.

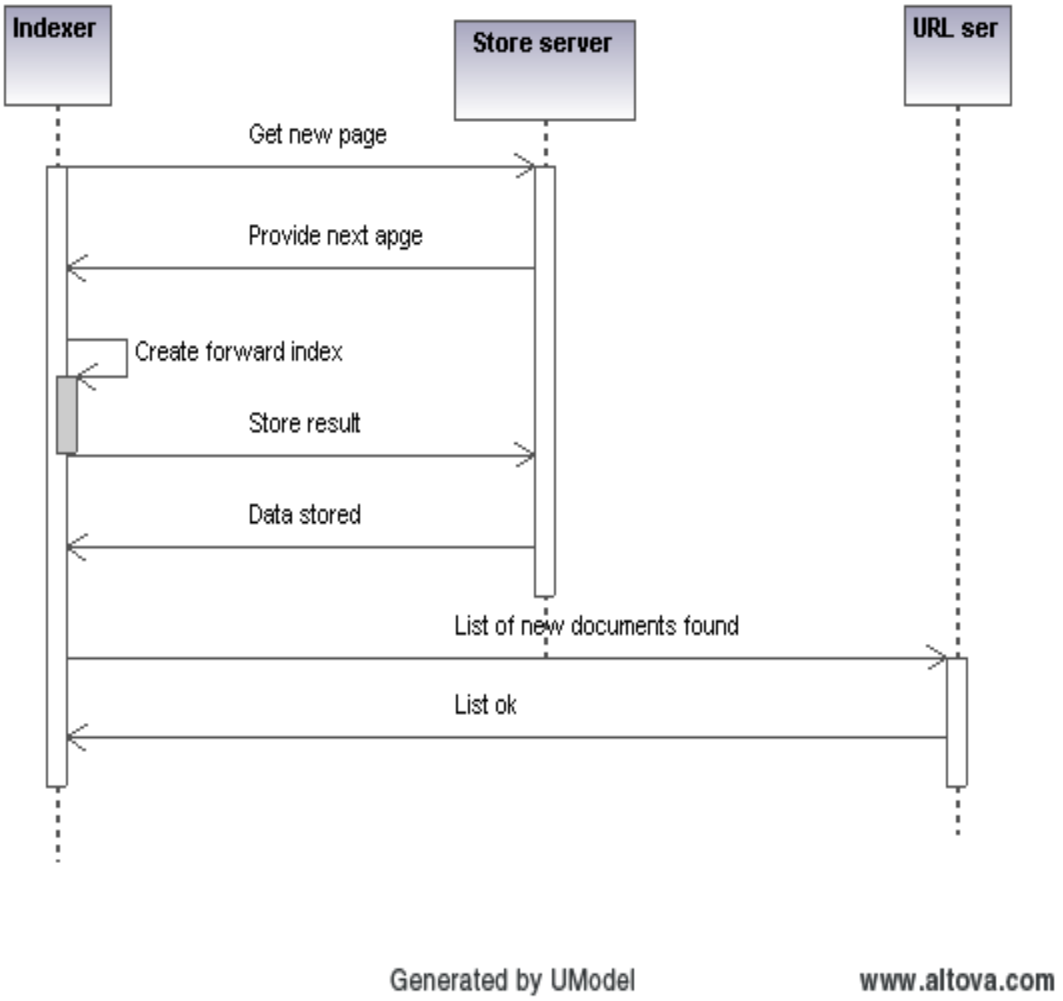


INDEXER

The indexer is another continuously running search engine program which has the functionality after the crawler.Its main task is to index the crawled webpages.The objects involved are indexer,store server and URL server.

The indexer requests a new page from the store server.It then scans the page and prepares forward index for each page.

This forward index is then stored in the store server and the links found in a particular URL are also stored in the URL server for crawling.

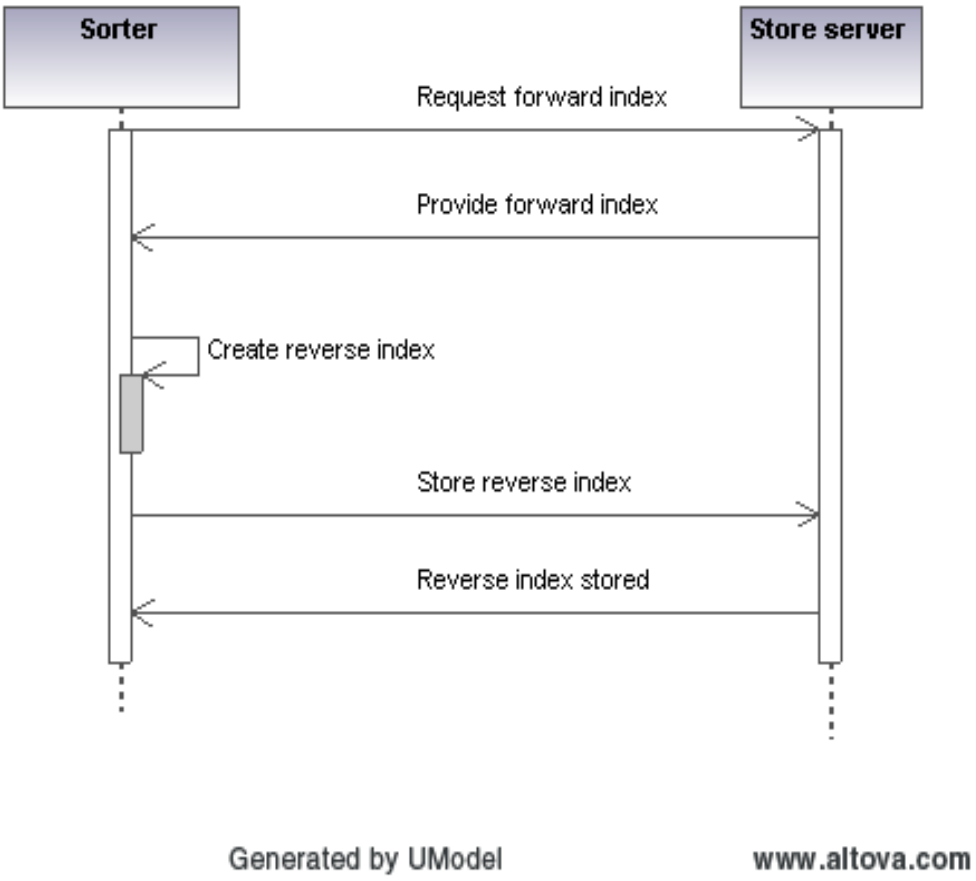


SORTER

Sorter is another main program of the search engine system whose function is to sort the documents according to their significance for words thereby creating a reverse index..

The various objects involved are sorter and store server.

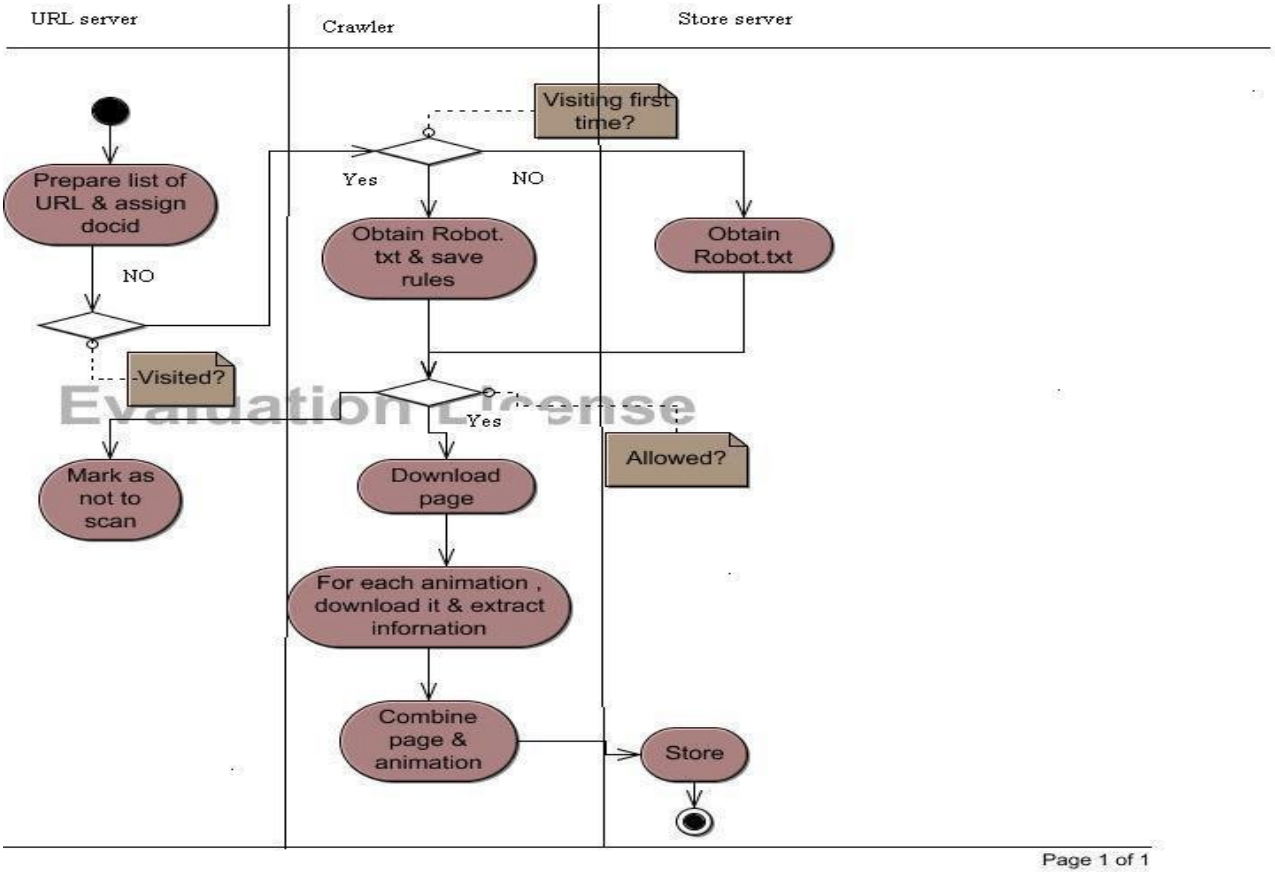
The sorter requests the forward index from the store server and then creates a reverse index for each word and stores the DOCIDs in a sorted order with respect to its importance for a given word.



9.2.3. ACTIVITY DIAGRAM

This is another UML diagram used in object oriented analysis.This diagram gives the list of activities being performed in the system.It starts with a filled circle and ends with bull's eye.The special cases that can be shown in activity diagram are fork and join.A set of parallel activities emerges from a fork and is merged using a join.Every join has a corresponding fork and vice versa.A condition can also be shown in activity diagram using a diamond having more than 1 output.The activity Diagram also tells who performs these activities with the help of swimlanes which are drawn either vertically or horizontally.
The activity of the system is divided into many parts and a diagram is shown for each:

System activity diagram:



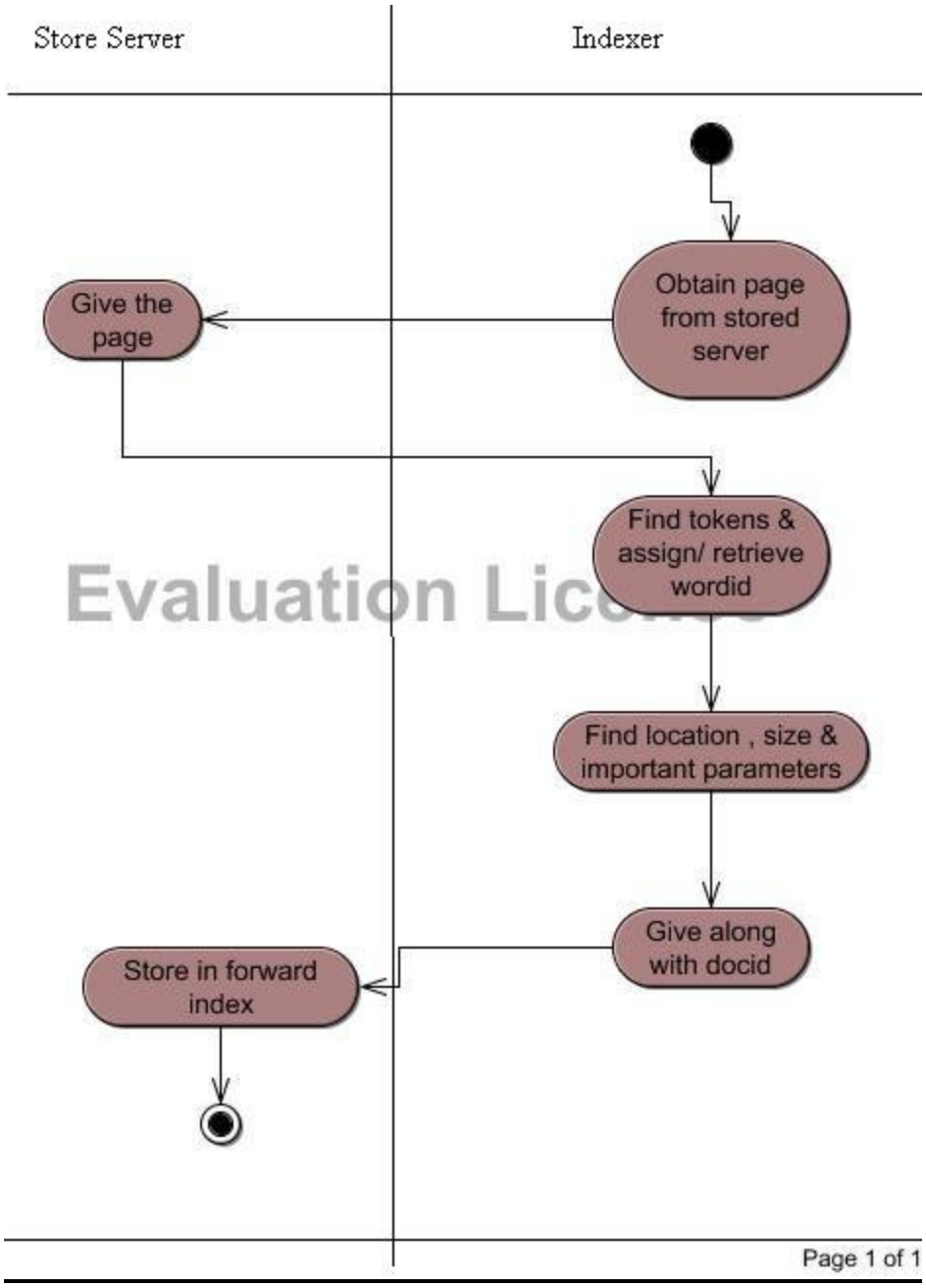
The system activity diagram shows the following activities taking place in the system:

URL server prepares list of URL's and assigns a DOCID to each URL.If the URL is not visited then give it to the crawler.If the crawler is visiting the URL for the first time then the crawler downloads the file robot.txt to see the access permission and saves this file for future reference.Now download the page and for each animation in the page download the animation site and extract the information from this animation site.Now combine this page and various animation pages and store in the store sites.If the page has already been visited then download it from the store server .

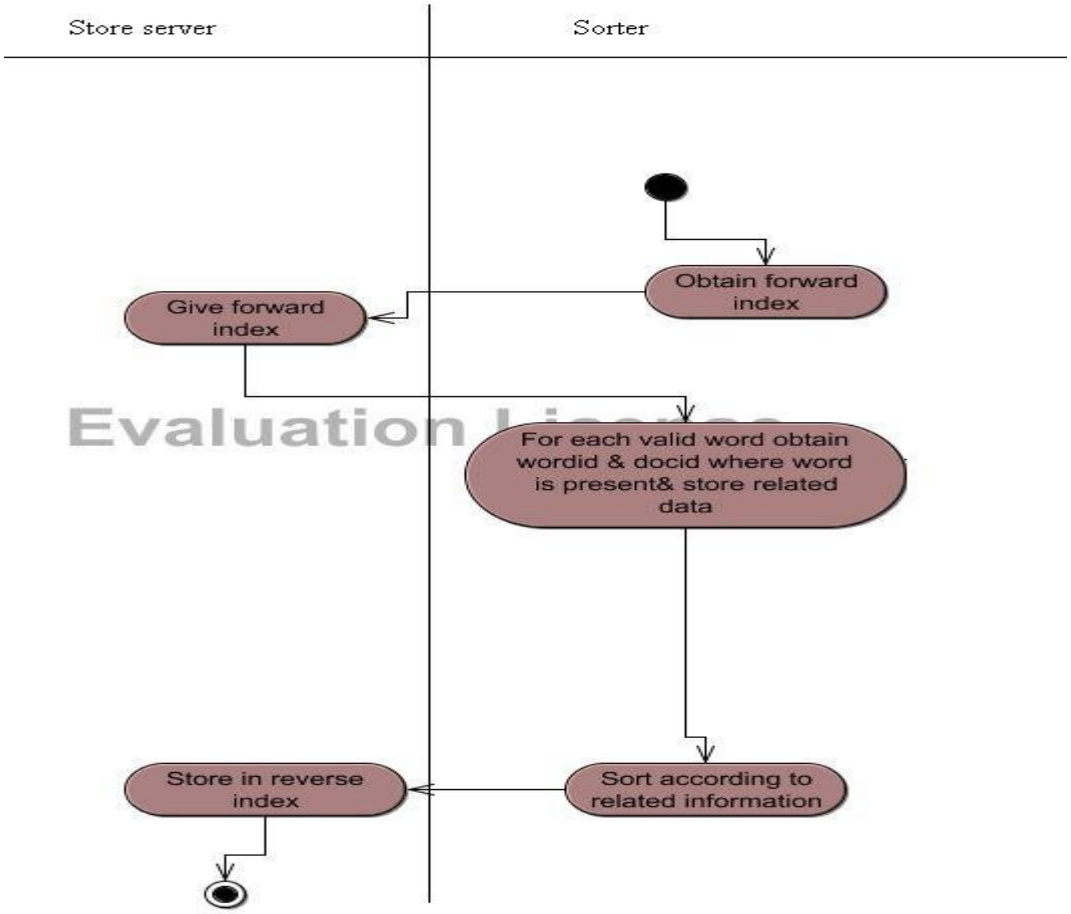
Indexer :

The indexer performs the following actiities:

Indexer asks the store server to give a web page.When the server returns the page indexer finds all the tokens.If the token is not new then it retrieves its WORDID and if it is new then assigns it a new WORDID and stores it in a lexicon.The indexer also finds the information related to each word like its location,font size and font type to know its relevancy information.The indexer then prepares the forward index including corresponding DOCID and words included in the document along with related information.

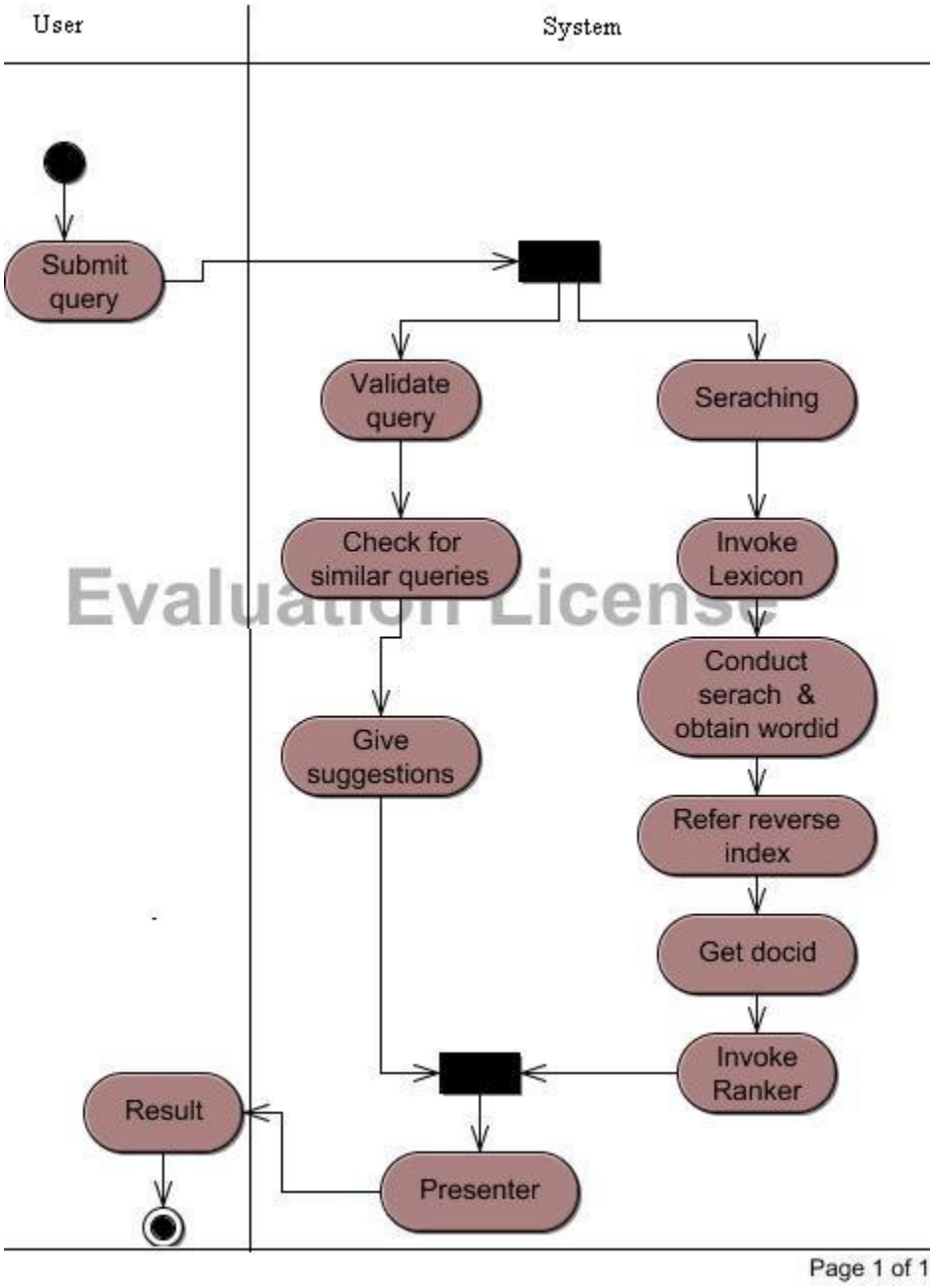


SORTER



The sorter first obtains forward index for all the pages from store server and for each valid WORDID in lexicon find all the DOCIDs containing the word and sorts these DOCIDs in the order of importance of the documents.

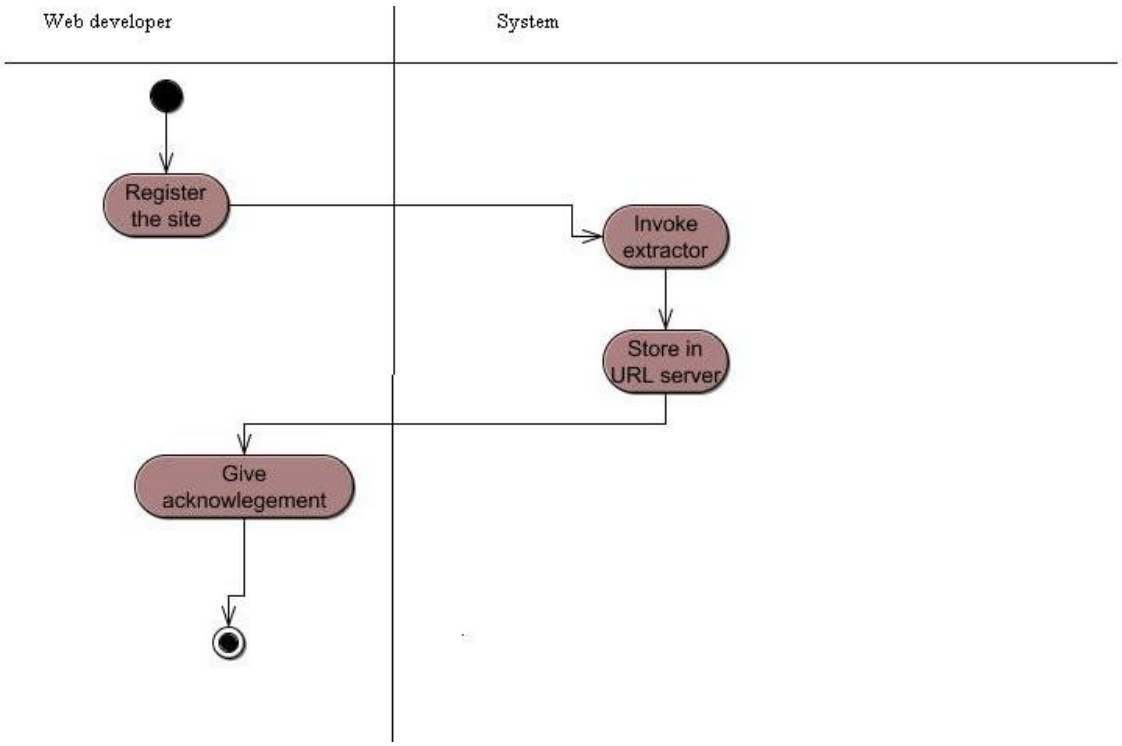
SYSTEM FLOW



The system flow gives the activity diagram for the complete system from user's point of view.The user submits the query to the system.This leads to 2 parallel activities of searching the URLs and validating the query.Validation checks for similar queries and spelling mistakes and also gives suggestions regarding these.Searching invokes lexicon to

find WORDIDs for the words and using these WORDIDs finds the DOCIDs from the reverse index.The searcher now invokes the ranker to rank the documents.These ranked documents are then given to user.

WEB DEVELOPER



The activity diagram of web developer gives the activities while registering a new web site with the search engine.

The web developer registers the website which is extracted by the extractor and stored in the URL server.The system then gives acknowledgement to the web developer.

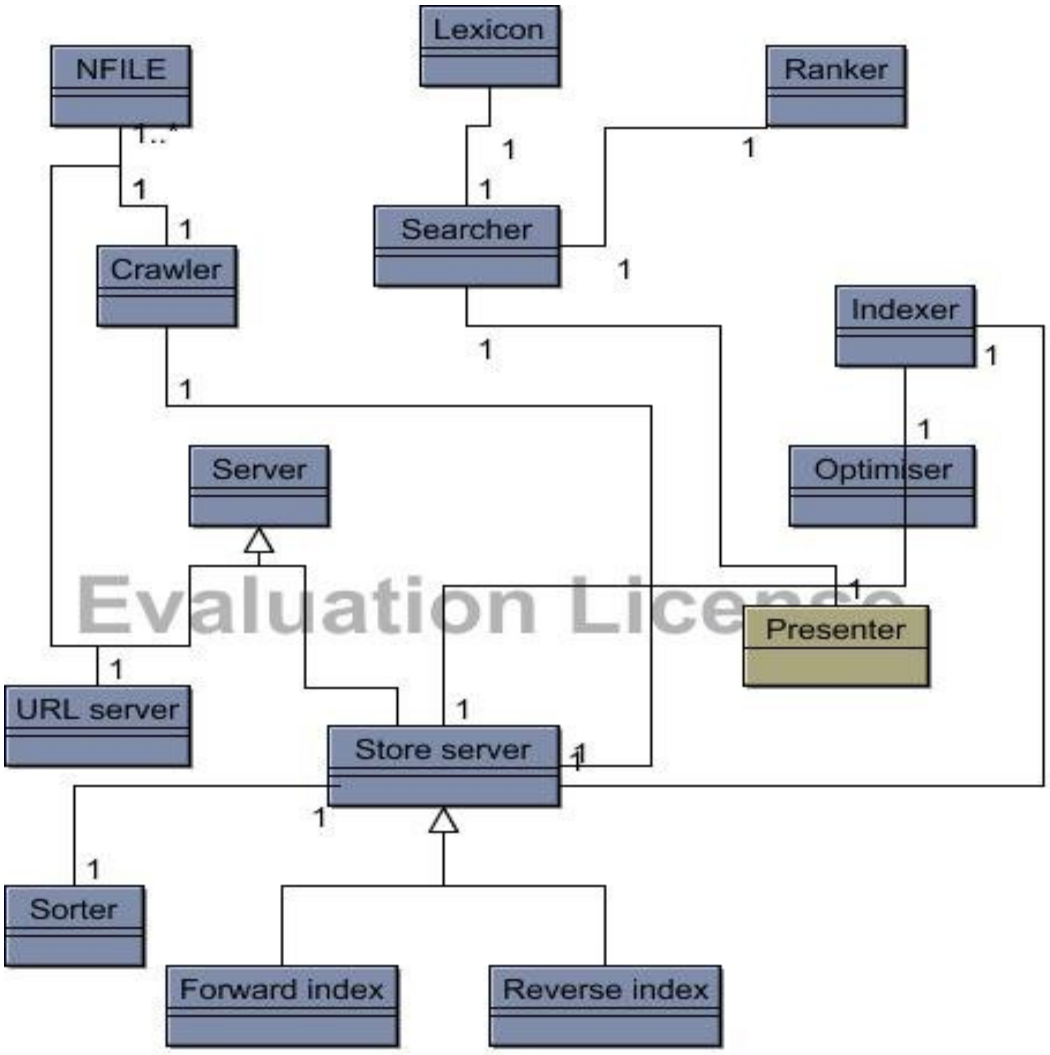
10. Design

Design is the first step in the development phase for any engineering product or system. It may be defined as the process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization.

10.1 OBJECT ORIENTED DESIGN

CLASS DIAGRAM:

The class diagram represents all classes & relationship required for implementation of project. It specifies all classes that are obtained from use case diagram & from application domain. The following class diagram represents basic classes & interfaces that are needed. They are identified from usecase diagram. The properties & methods can be specified as detailed analysis is carried out.



The NFILE class is used extended from File (which is inbuilt class in Java that provides access to file properties & provides methods to retrieve data from file). It is extended to get description of file . The description provides information that web page contains animation or not & hence the web pages that contains animations can be indexed & size of storage server is reduced.

The Crawler class will implement functionalities of crawler. It is related to Nfile. The server class provides base class for 2 types of server.

1. URL server
2. Store server

The store server is responsible for storage of indices.

   a. Forward index
   b. Reverse index

The store server class is used by indexer to access forward & reverse index. The presenter is browser interface that takes input & display output. The presenter is interface. The Ranker considers various criteria for ranking animation & works with optimizer support. URL serve stores list of URL's to be crawled by crawler. The class sorter sorts the index according to various parameters.

11. CONCLUSION

Thus we have completed the first part of this project which includes analysis and design part.The complete project has been done using Software engineering concepts.

12. REFERENCES:

The following list contains list of web sites that are used as references by our project group.

1. http://www.serachtools.com
2. http://www.press.umich.edu/jep/07-01/bergman.com
3. http://www.robotstxt.org
4. http://www.archives/eprints.org
5. http://en.wikipedia.org/wiki/Web_crawler
6. http://www.searchengineshowdown.com/features/google/review.html
7. http://en.wikipedia.org/wiki/Search_engine
8. http://www.openarchives.org/registar/browsesites