DISTRIBUTED WEB INDEXING AND SEARCHING
A PEER-TO-PEER ARCHITECTURE WEB SEARCH ENGINE

STEPHEN NAICKEN

School of Cognitive and Computing Sciences
Supervisor: Dr. Vladimiro Sassone

Table of Contents

1 Statement of Originality

This report is submitted as part requirement for the degree of Computer Science at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

2 Acknowledgements

Many thanks to my supervisor, Dr. Vladimiro Sassone, for all his help and advice throughout this project.

I would also like to acknowledge members of the Open Source community, especially members of the Jython-Users mailing list, who have helped answers some of my questions.

Finally, I would like to thank my Mother and Father for their support.

3 Introduction

3.1 Background

The World Wide Web was invented in late 1990 at the Conseil Européene pour la Recherche Nucléaire (CERN) by Tim Berners-Lee. The motivation for this project was to provide physicists working around the world the ability to share information on large-high energy projects. However, others outside academia and science soon began to publish their information on this medium. The number of web pages increased from under a million in 1996 to over three hundred million in 1998 [1]. With this huge number of Web pages available and number of users growing at exponential rates [1], users needed a means to search the Web. Search engines and Web directories are the two most common means for accomplishing this task.

Web directories such as Yahoo are maintained by humans which means the quality of the data is high as Web pages can be assessed by a maintainer before being categorised, however adding new sites to such directories is a slow process so they often contain volatile data, such as dead links.

Search Engines such as Google, build an index through the use of a Web crawler. The crawler is an application that starts from a set of Uniform Resource Locations (URL). It accesses pages by recursively following hyperlinks from the start set and extracts the information needed from Web pages at these locations to create an index. The index can be searched by using a Web page interface. Significantly more web pages can be indexed by a crawler than can be made categorised in a Web directory, but not all Web pages will ever be indexed because the Web is not complete. Many pages are not linked to from others and can not be reached by the crawler. This part of the Web is often referred to as the "invisible Web".

3.2 Problems posed by the Web

Baeza-Yates, Ribeiro-Neto et al (1998) [1], list six main data related problems with the Web that make searching difficult. The first is distributed data, data contained on many computers on a network with the reliability of connections varying widely. The second is the high percentage of volatile data. The Internet is a dynamic network with computers added and removed constantly, so Web pages may disappear resulting in dead links. Domain and file name changes also add to Web volatility. The next issue is unstructured and redundant data which is common on the Web. The Hypertext Markup Language, HTML, is not structured (especially when compared to XML documents) and approximately 30% of Web pages are near duplicates. The quality of data of Web based information is also discussed and it stressed that the quality can not be guaranteed as there is no editorial process. Finally, heterogeneous data must be taken into consideration as the Web consists of files in multiple formats and different languages. Some of these problems can not be resolved by a search engine or Web directory. For example, a Web search engine can not ensure quality of data, as the information on a Web page may be syntactically correct but semantically incorrect.

3.3 The proposed system

In an attempt to address some of the problems listed above, I propose to create a search engine whereby Web users create a local index of the Web pages that they have visited. The reason for this is that Web users differ greatly in behaviour compared to search engine crawlers. Users selectively choose hyperlinks to follow based on the information contained in and around the link, the URL destination of the link and how they regard the quality of the page containing the link. Users can also access areas of the World Wide Web that crawlers can not. This can be done by accessing the URL of an unreachable site that may have been obtained by word of mouth, by guessing URLs or by other means into a browser. Crawlers can "overwhelm a server with rapid requests and use significant bandwidth ... Crawlers can also have problems with HTML pages that use frames" [1]. Both of these problems would be overcome by having users create the indices.

If there are several users and they create a local index, there will be distributed indices across the Internet. The next challenge is how to make these available for searching. There are three ways to achieve this task:

1. A central server could be used with indices submitted to it after they have been created by users. The submitted indices could then be merged with existing central ones to form a single central index. The central index would be searched identically to existing systems. M. Rio, http://www.cs.ucl.ac.uk/staff/M.Rio/projects.html, proposed this idea.

2. A semi-distributed structure could be used where users can submit to one of a number of different servers. The servers would merge the indices upon receiving a new index in an identical manner to the central server method described above. A query would be made to a proxy server, which would send the query to the servers containing the indices. Each server would reply to the proxy with a result set. The results could then be merged by the proxy and sent to the user.

3. A totally decentralised Peer-to-Peer network could be used. The indices would remain located on the computers that generated them. A search by a given user would be distributed, using viral propagation, to nodes on the network. The nodes would search their index against the given query and return a set of results to the user.

The third option, a totally decentralised system has been chosen for the following reasons:

- The load of central server indexing on CPU and primary and secondary memory can be

distributed among many computers.

- Peer-to-Peer offers greater anonymity. If users had to submit their index to a central server, the sites they visit could be associated to their IP address. Peer-to-Peer by its network topology makes this difficult.

- Centrally stored indices are hard to keep up-to-date. By using Peer-to-Peer, this can be overcome as users can store an index of the latest sites they have visited.

3.4 Aims of the proposed system

By combining the paradigm of browsing and retrieval, this project aims to:

1. Create a means for users to index Web pages that they have visited using a Web browser, such Microsoft Internet Explorer, Netscape or Opera.

2. Make these indices available for searching by other users using a Peer-to-Peer network.

A requirements specification will be presented later in this report. It is important that the application created meets the requirements specification as best possible. If not met completely, the project should not be considered a failure, providing the two goals above are completed.

3.5 Glossary of Terms

The following is the definition of some important term:

Node, Peer - A computer on a Peer-to-Peer network.

Application, Program, System - The software application coded as part requirement for this project.

UML - Graphical Tool to represent software visually.

Index - A file, usually a database, holding information about documents. The index is part of a searching system and the data held in it is used to formulate replies to a query.

Design Patterns - A re-usable software pattern that allows software designers to create a design using existing, working structures.

3.6 The following sections

The rest of the report documents the design, implementation and testing of the proposed system. The report is divided into the following sections:

1. Analysis – This section includes the requirements elicitation, specification and use-case model. Ideas about the proposed system are developed and the basis for the design is created.

2. Design – In this section, the analysis is developed into a design of the system. The structure and behaviour of the system is documented.

3. Implementation – In this section, implementation issues and deployment of the system are discussed.

4. Testing – This section describes the testing of the implementation. Tests are primarily focused on the user through the use of black-box testing. Test results and analysis are given.

5. Conclusion and Further Work - The conclusion and discussion of further work that may be undertaken.

4 Analysis

4.1 Overview

This section begins with a discussion of existing Web search systems. A requirements specification that details criteria the system must comply with is then presented along with a description of how the proposed system will work. Finally, a Use-Case model is presented further detailing functionality from the user perspective.

4.2 Requirements Elicitation

4.2.1 Existing Systems

4.2.1.1 Web Directories

A directory is described in [1] as being a:

> *hierarchical taxonomies that classify human knowledge*

Web directories store links to information in a categorised, ordered and hierarchical manner. When browsing a Web directory it is similar to browsing a tree structure, however the actual structure is a directed acyclic graph [1]. It is now common for Web directories to offer search functions that search the directory and also the Web using a business partner's search engine.

The advantage of Web directories is that humans maintain them, so the quality of Web pages can be verified before being added to the directory. This resolves the problem of quality of data defined by Baeza-Yates et al (1998).

Despite the advantages, Web directories have two significant problems. First, classification of Web pages can be difficult as their interpretation is subject to debate. Second and most importantly, directories will not be able to categorise all Web pages, as the human maintainers cannot cope with the rate of Web growth.

4.2.1.2 Search Engines

4.2.1.2.1 Architecture

Search engines consist of several components, a crawler, indexer, index, query engine and interface. The diagram below describes the Alta-Vista search engine architecture:

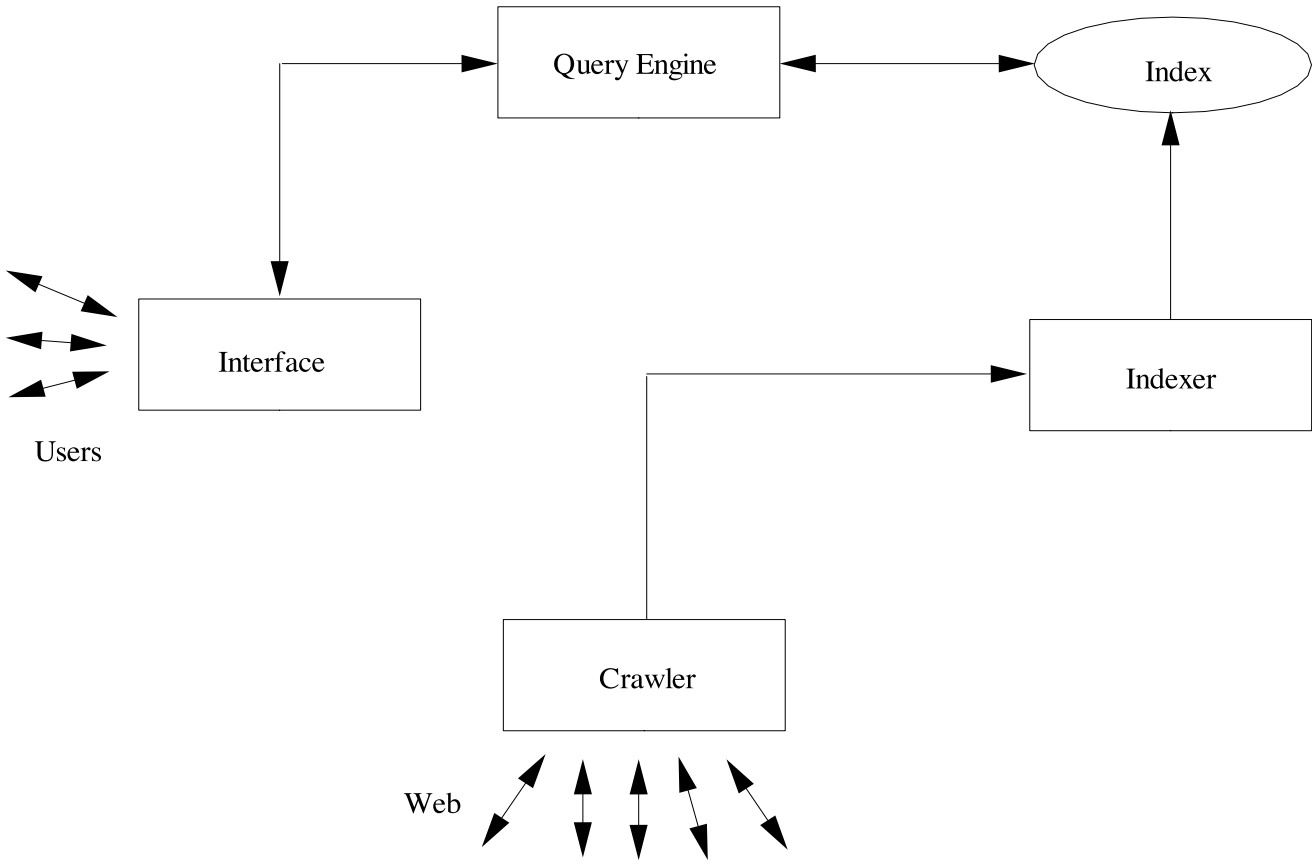


Figure - typical crawler-indexer architecture, taken from [1]

A crawler starts from a set of URLs and recursively visits URLs this start set. The pages located at these URLs are parsed and then indexed. The index can be searched using a Web page interface. A list of results is returned to the user based on a ranking algorithm that displays the best documents first. Users can then analyse the results and visit the Web pages referenced in the results.

Issues regarding each component of the architecture are described below.

4.2.1.2.2 Crawler

The functions of crawlers have been stated in earlier paragraphs. Search engines are the most common means of searching the Web and this is in part due to the ability of the crawler to index pages en masse, but there are several disadvantages to this technique:

1. Crawlers often have difficulty with pages using frames or image maps [1] because of the complex HTML used to implement them.

2. Crawlers account for a substantial amount of Internet traffic [2], and can degrade a Web server's resources [1].

4.2.1.2.3 Indexer

There is very limited information available on how search engines index documents. The search engine market is competitive, with most search engines owned by corporations. However, some information about the techniques use is available.

Search engines remove the most common words in the English language from indexed documents. These words are called stop words and by removing them, the size of the index can be reduced greatly. Compression techniques are also used to reduce index size. Normalisation of documents is also used, this includes converting words to lower case and reducing words to their canonical form e.g. going becomes go.

Indexing is a computationally expensive task. The fastest indexing algorithm is memory-based. This algorithm takes six hours to index five gigabytes of text but uses 4Gb of primary memory in the process [2]. Other alternative algorithms that compensate on indexing time to improve memory usage are available. In 1998, AltaVista were using 20 multi-processor machines, each having more than 130Gb RAM and over 500Gb of hard disk space [1]. To centrally index a large number of Web pages, extremely high-spec machines are required because indexing and handle queries is resource intensive.

4.2.1.2.4 Query Engine

As with the Indexer, companies that develop Web search engines often do not release the details of their query engine. It is know however that most query engines use variations of the boolean-model or vector-model for ranking. The boolean model supports boolean queries, such as “a AND b OR c NOT d”. The vector model allows users to specify a list of words related to their search goal. The query engine compares the query and indexed pages. The more similar they are, the higher the ranking of the

document.

4.2.1.2.5 Interface

The interface of a search engine can be split into two sections, the query interface and the results interface. The query interface allows the user to submit a query to the search engine by entering their query into a text box and clicking a submit button. Results are then displayed to the user on a page with the best results ranked higher up the list. User often have difficulties specifying queries using complicated syntax such as boolean queries [1], [2], so interfaces are designed to allow them to submit a list of words appropriate to their goal.

4.2.1.2.6 Performance

Search engines are the most popular method of finding information on the Web and can index a much larger number of Web pages than can be added to a Web directory. Recent statistics from SearchEngineWatch.com, http://www.searchenginewatch.com, show the following:

Search Engine	No of indexed pages (billions)
Google	1.37
FAST	0.625
Altavista	0.55
Inktomi	0.50
Excite	0.38
Northern Light	0.35

Stats as of August 15th 2001, http://searchenginewatch.com/reports/sizes.html (25/11/2001)

These statistics will include semantically identical documents, for example mirrors. Approximately 30% of Web pages are near duplicates [1]. The quality of the data cannot be verified as it is collected by a crawler so it is often not what the user is looking for. The volatile nature of the Internet means the relocation of files and dead links result in many results from a search engine being non-existent. Finally with growth of the Web currently at an exponential rate. The indexing and storage of a large amount of data centrally can pose a problem. By using Peer-to-Peer in this project and distributing the indices, the latter will be eliminated.

4.2.1.3 Comparison of Existing Systems Against Proposed System

Users using the system described in this document should notice several advantages when compared to the Web retrieval methods described above.

The proposed system eliminates the need for a crawler. This means that the disadvantages of crawlers stated above are eliminated. In particular, users will be indexing sites they are visiting, so this will not add any unnecessary load on Web servers for indexing purposes.

Central servers creating an index of many Web pages have to have advanced system specifications. However, in the proposed system, by distributing the indices the processing load is also distributed.

Pages unreachable by crawlers will have more chance of being indexed and becoming a site in the distributed search engine. The reason for this is that a user may intentionally (e.g. word of mouth) or unintentionally visit the site and index it.

4.2.2 Peer-to-Peer Systems

4.2.2.1 Gnutella (including JXTA, formerly InfraSearch)

Gnutella is a peer-to-peer file sharing protocol. It is commonly used for the distribution of Motion Picture Experts Group-1 Layer 3 (MP3) files, but can be used for other files as well. Users can search for and download files from other peers. A peer will submit a search query to peers it is connected to. These peers can then check to see if they have any filenames that match the query and if they do, they can reply to the node. They also forward the query to other peers and the responses are passed through the nodes that broadcasted them until they arrive at the source node of the query. Figure 3 and 4 below describe this principle in more detail.

> "Freenet can be described as a bandwidth disk space - sharing concept with the goal of promoting free speech. Gnutella is a searching and discovery network that promotes free interpretation and response to queries." [2]

Gnutella developers recently developed InfraSearch, now part of Sun JXTA. Based on Gnutella, Infrasearch is distributed search system that works as follows:

- Information providers would be nodes on a Peer-to-Peer network.
- A user would through a Web page, submit a query that would be broadcast to information provider nodes on the network.
- Each node would evaluate the query and respond with appropriate results.

- The results from nodes are html code that is processed into a complete Web page at the client side.

4.2.2.2 Freenet

Freenet is another peer-to-peer file sharing system. The idea behind this is that a peer has disk space containing documents that are shared with other peers. Users can access documents using Uniform Resource Indicators, URIs, which usually consist of the key of the document. Although it has a similar network topology to Gnutella, the system is not designed for search-based retrieval.

4.3 Requirements Specification

The following is a list of functionality that the system must meet:

4.3.1 Functional Requirements

1. The user must be able to initiate the indexer at will or by scheduling the task.

2. The following should not be indexed:

   2.1. Web pages stored on the local file system must not be indexed, as these are not accessible externally.
   2.2. Web pages that must not be indexed because they use the robot exclusion rules.
   2.3. Secure pages accessed using HTTPS to protect user privacy.

3. Only the software generated index and no other files will be made accessible to external users of the system.

4. A Web page will not be duplicated in the index. If a user visits http://www.google.co.uk/ twice, this site will not appear twice in the index.

5. A user must be able to submit a query to the system and view the results. The following are the characteristics of the results:

   5.1 The result set can be empty or non-empty.
   5.2 Duplicate results must be discarded.
   5.3 Each result must provide the user with meta-document information.
   5.4 If possible, results should be ranked with the best listed first.

4.3.2 Non-Functional Requirements

4.3.2.1 User-Interface

An interface must be provided so that users can perform tasks of indexing and searching. At this stage, it has been decided that the interface for searching the system and retrieving results should be Web-based. The reason for this is that all Web search engines use a Web page interface. It is important to keep this theme in this system as users are accustomed to and feel comfortable with this concept. The interface will be developed further at the design stage.

4.3.2.2 Error Handling

Regardless of the design of the user interface, error and unexpected situations will occur. It is important that these are handled gracefully with the minimum inconvenience to the user.

When an error occurs, the system should attempt to handle it appropriately without user assistance if possible. However, if the user must be informed of the situation, or his/her input is required to solve the problem, then user interface should be used to this affect.

4.3.3 Extensions

4.3.3.1.1 Indexing of Other Document Types

Although most of the Web consists of HTML documents, there is an increasing amount of other types of documents. Documents such as Adobe Acrobat's PDF file and PostScript, as well as images, sound and music files are available on the Web. If possible, some of these formats could be indexed as well as Web pages.

4.3.3.1.2 Cached Sites

In the Google, http://www.google.com/ query results, a link to a cached version of a site is available. If the site in a set of results that the user wishes to visit is no longer accessible, he/she can access a cached version. This functionality could be implemented in this proposed system.

4.3.3.1.3 Further User Indexing Control

The indexer does not allow users to select sites they do not wish to index. Users may feel uncomfortable with this as they may visit pages that they wish not to be associated with such as adult content. A function could be implemented to allow users to select pages not to be indexed.

4.4 Proposed System Overview

As stated in the introduction, the goals of this project are to create a system that indexes web documents and to make these indices searchable using Peer-to-Peer. The system can be divided into two subsystems accordingly:

- The Indexing of Web documents.
- The Searching of Web documents.

4.4.1 Indexing

There are two ways in which Web pages could be indexed. The first is to index pages as soon as they are visited by the user [4]. The second is to index the Web pages stored in the browser's cache directory.

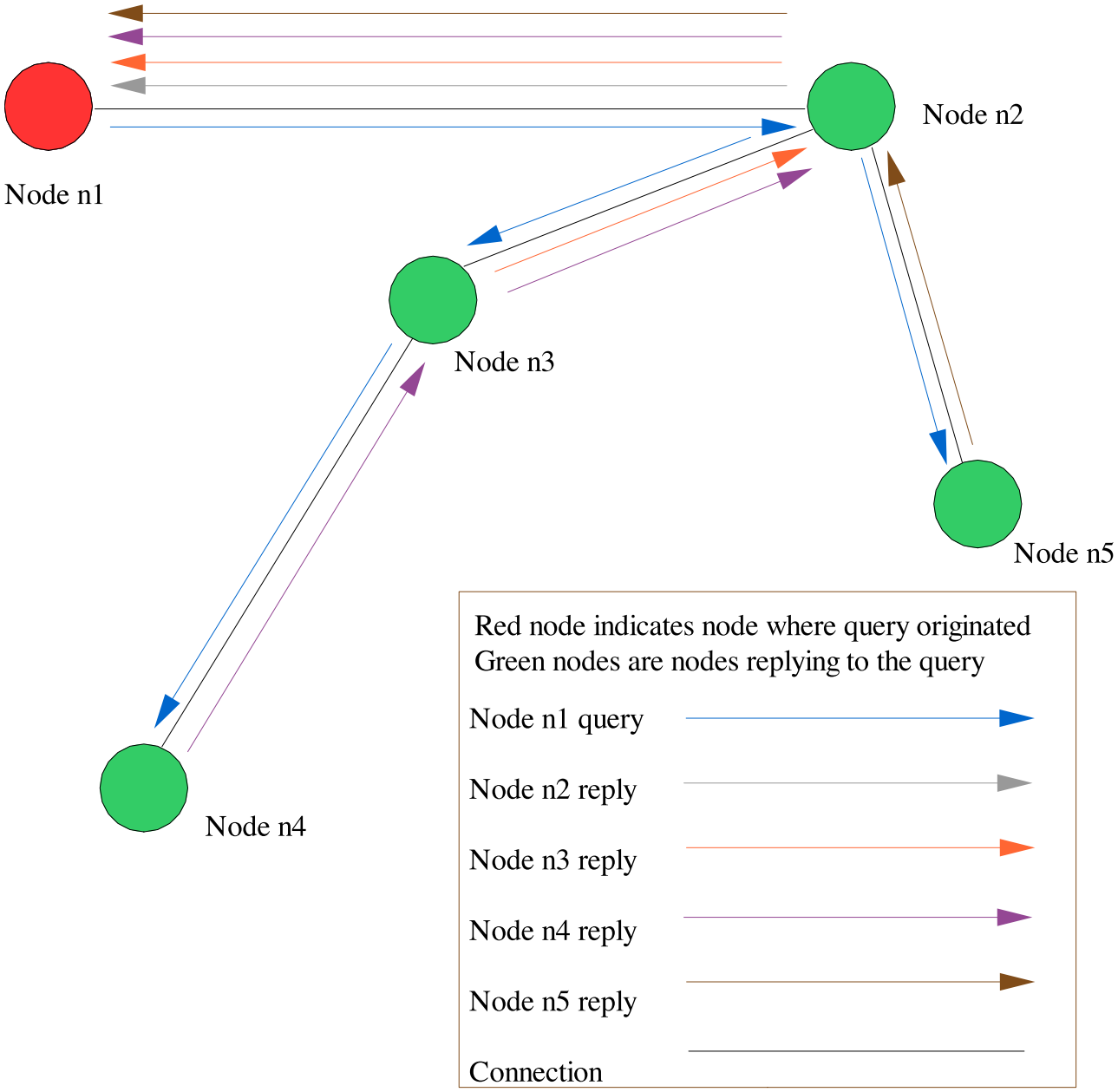
For the first method proposed, a browser plugin could be built. This plugin would index pages as soon as the user visits them. Alternatively, if a plugin for this task cannot be created, the indexing functionality could be hard-coded into the freely available Mozilla source code. Both techniques would require C programming, which is a skill that would have to be learnt.

The alternative solution is to index Web pages stored in cache. Web browsers store all visited pages in a cache directory apart from local files and those accessed using HTTPS, Hypertext Protocol over Secure Sockets Layer. The indexing process could be initiated by the user at will or by a scheduler. The user could set times at which the scheduler would index the cache.

The standalone application would certainly be easier to implement as it could be done in a language known to the developer, such as Java or Python. The browser plugin would require learning C, which would take some considerable time. Although C code is regarded as easy to write, it can take a long time to debug, which is not ideal in this situation because time is limited. For these reasons, a standalone application will be implemented for the indexing of Web pages.

4.4.2 Searching

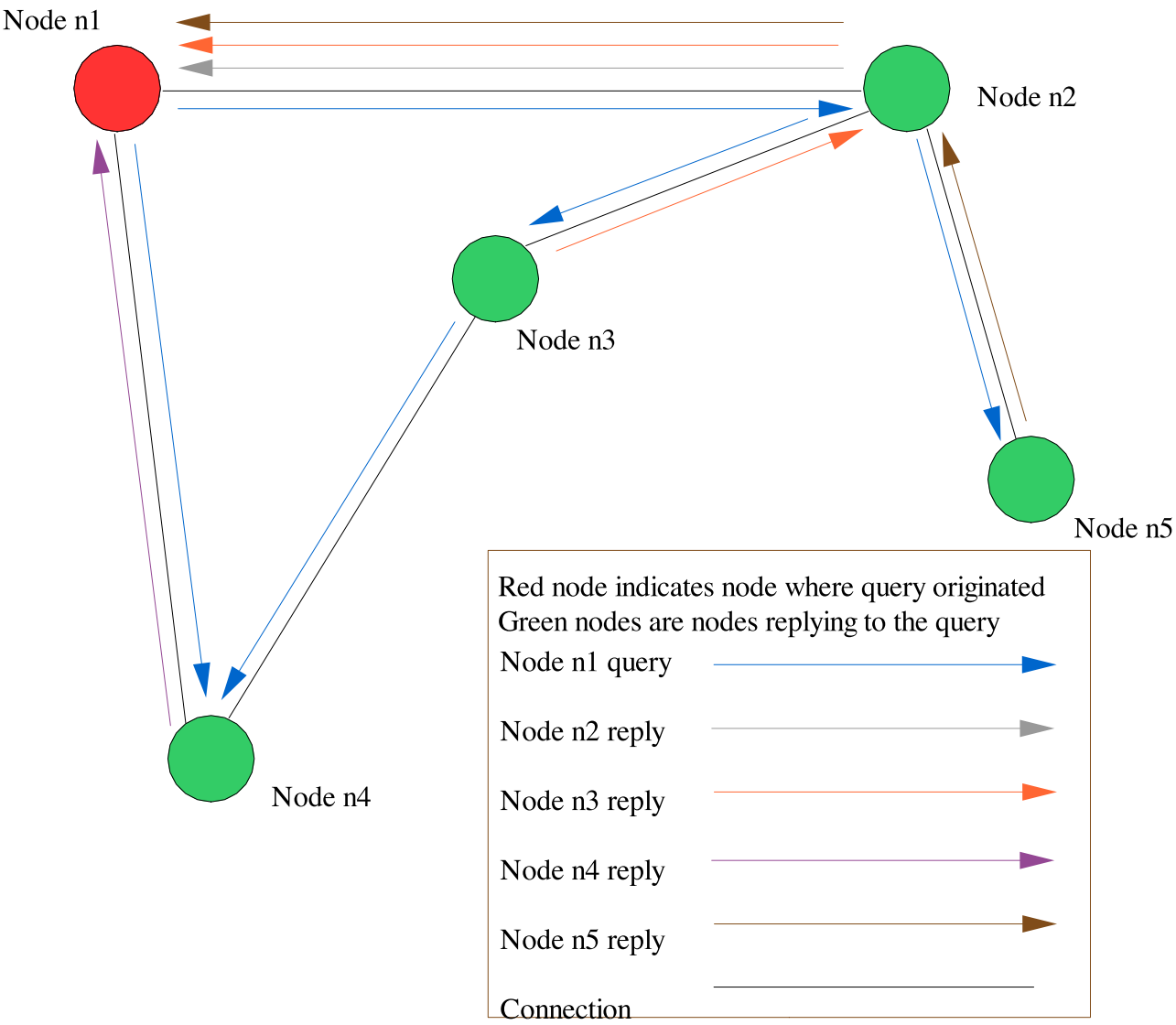
If many users utilise the indexing application, there will be "distributed intelligence" [2]. That is there will be information, indices of Web pages, at various nodes on the network. These nodes of information can be connected to each other using a Peer-to-Peer architecture. When a user searches for a given subject, the search is propagated through the network and results returned.



Distribution of Queries and Retrieval of Results.

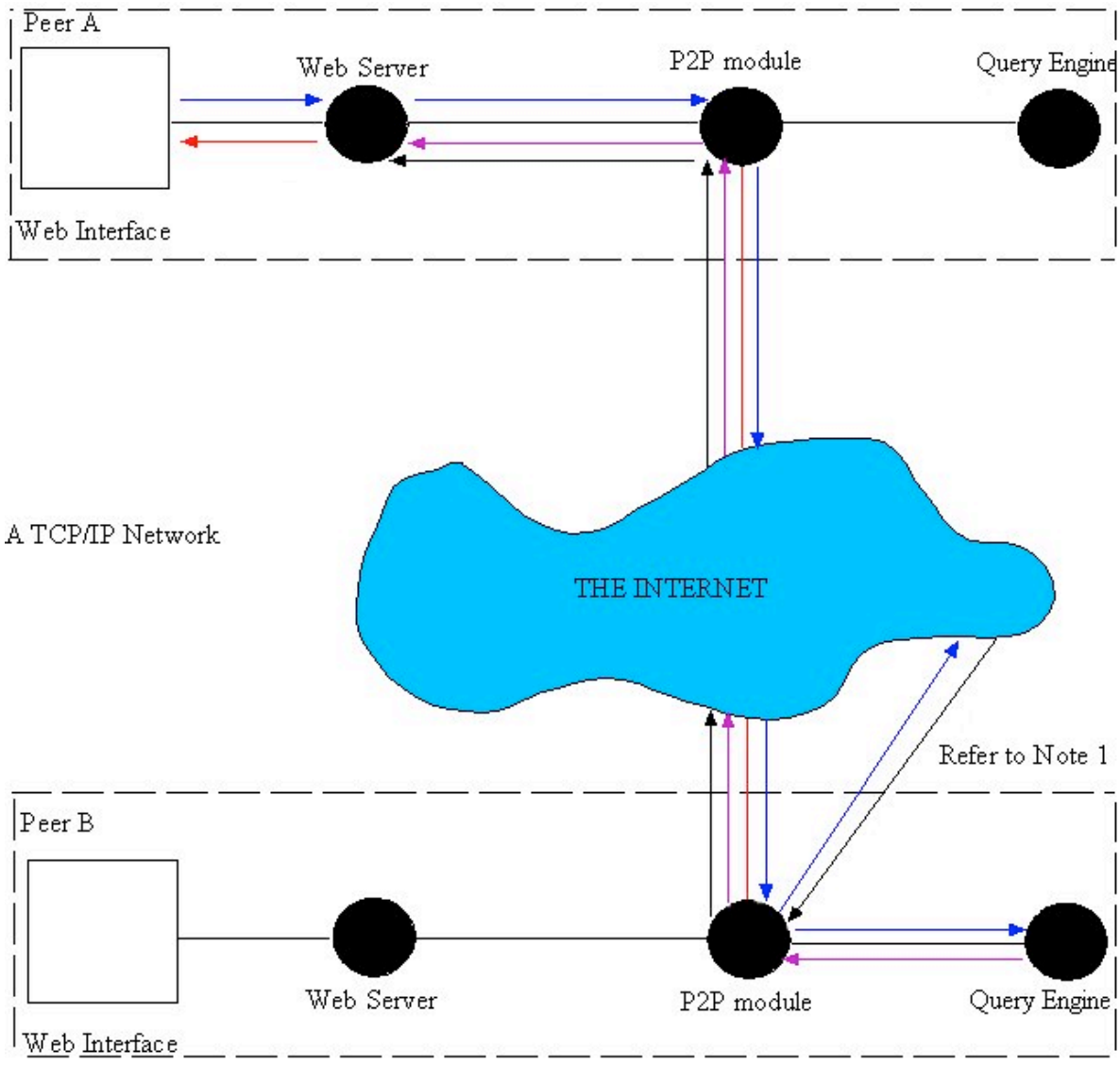
The diagram shows a set of nodes as part of a Peer-to-Peer network. The network consists of 5 nodes. Node n1 sends a search query to the peers it is connected to, n2 and n3. n2 and n3 reply to the query but also forward the query to n5 and n4 respectively. These nodes reply to the query by sending their results to the nodes that queried them, n3 and n2. The responses received from n4 and n5 are not for n3 and n2, so the responses are forwarded to n1 who the reply is for.

A peer-to-peer network is subject to cyclic conditions. So a node may see a query more than once, but it can only reply to the same query once.



Distribution of Queries and Retrieval of Results in a cyclic P2P network

The import thing to note here is that n4 only responds to n1's query once. On the second time of seeing the message, n4 ignores it. Gnutella uses a system of Global Unique Identifiers, GUID, for this. Each message broadcasted on Gnutella has a GUID, which uniquely identifies it. For each message the node receives, it stores the GUID of it. When a message is received and the GUID has been seen before by the node, it is ignored. This same method should be implemented in the proposed system. This technique importantly reduces network traffic.



Note 1. These two lines show the Peer B forwarding the search query of Peer A to other peers through the Internet.

Key

Peer A Search Query (q1) → Merged Query Results

Peer B Query Reply (r1) → Component Link

Peer B Query Reply (r2) → Network Connection

the process of submitting a query and receiving the response at component level

Query q1 is a search query from peer A. The search is sent to peer B over a TCP/IP network, here the Internet. Peer B searches its index against the query and returns a response r2 to the peer A. Peer B also sends q1 to other peers via the Internet. The results are returned to peer B and then forwarded to peer A. Results are merged into a single result set by the Web server on peer A before being displayed to the user.

Clearly the Peer-to-Peer element of this system could be represented using the Gnutella protocol. It offers Peer-to-Peer functionality identical to that required by this system and should therefore be used instead of creating a new protocol.

4.5 Use Case Model

4.5.1 Overview

Object oriented techniques will be used throughout this project. The Unified Modelling Language, UML, will be used to provide different views of the systems. These views will aid developers who which to understand or further develop the system presented in this report.

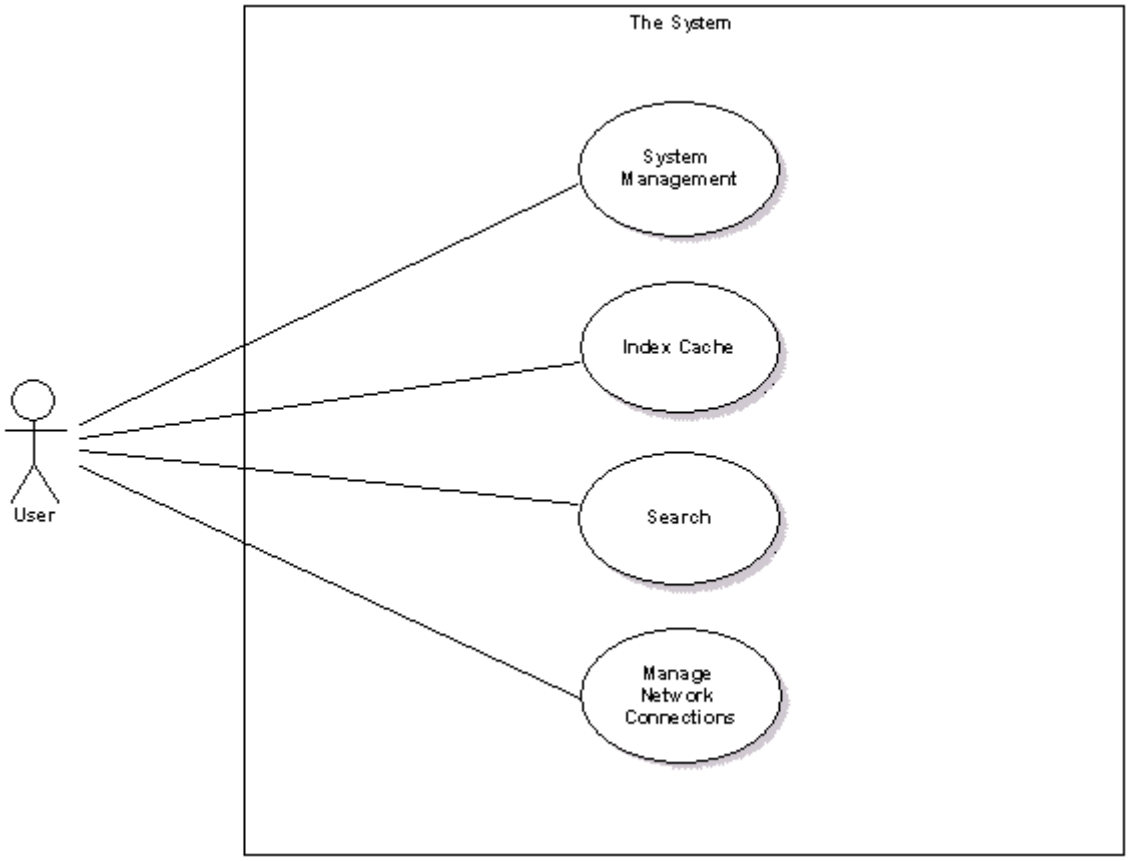
This chapter provides use cases of the proposed system that describe the functionality of the system

4.5.2 Introduction

The use-case model is important for two reasons. The first is that it allows the context of the model to be visualised [5]. This means that users outside the system that interact with it can be specified with their roles. The second is that it allows the modelling of the system requirements. Functionality can be described without referring to the internal workings.

4.5.3 High-Level Use Case Diagram

This high-level use case diagram shows the context of the system. Each use-case within the boundary of the system can be decomposed further into low-level use-case diagrams that give a more detailed view of the system functionality.



4.5.4 Low-Level Use Case Diagram

4.5.4.1 Low-Level Use Case 1 – Start Program

Goal: The user wishes to start the program

Actors: User

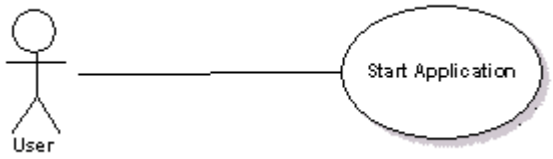
Description: Before the user can perform any functions, the system must be started. The exact details of this will be subject to the operating system being used and the language the system is implemented in.

Primary Flow: The user starts the application. The interface of the application is displayed upon successful loading.

Secondary Flow: If the application cannot be started for whatever reason, such as file corruption, an error should be displayed to the user.

Preconditions: The application is installed.

Use Case Diagram:



4.5.4.2 Low-Level Use Case 2 – Connect to Gnutella

Goal: The user wishes to connect to the Gnutella Network

Actors: User

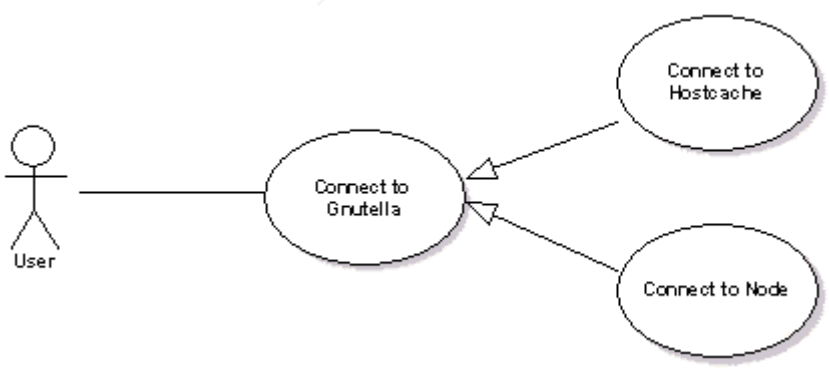
Description: Before a search can made, the user must be connected to the Peer-to-Peer network. There are two standard ways to connect to the network. The first is to connect via a host cache and the other is to connect directly to a node on the Gnutella network that is accepting incoming connections.

Primary Flow: The user has already started the application. The graphical user interface is displayed on screen allowing the user to either select a hostcache to connect to, or enter the hostname and port of a node that is accepting connections on the network.

Secondary Flow: If the IP of the node is invalid, because it does not exist, is not accepting connection, or has been entered incorrectly, a dialog box appears informing the user of this. The same also applies to connections to a hostcache.

Preconditions: Use-Case 1

Use Case Diagram:



4.5.4.3 Low-Level Use Case 3 – Submit Search

Goal: The user wishes to search the system and view the results to the query.

Actors: User

Description:

Primary Flow: Using a Web form, the user enters the search query and then submits it to the system. The results to the query are then displayed on a new Web page but in the same browser instance.

Secondary Flow: If the search fails for any reason, then an error message should be displayed to the user.

Preconditions: Use Case 1 and 2.

Use Case Diagram:

4.5.4.4 Low-Level Use Case 4 – Index the Cache

Goal: The user wishes to index the documents in their browser's cache directory

Actors: User

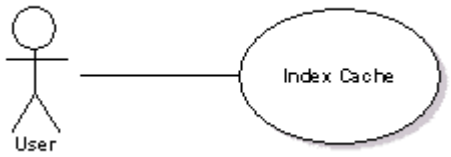
Description: Creating an index of the Web pages stored in the cache directory allows the other users on the network to search the pages available in the indexed cache. There are two ways that a user can index the cache, by initiating the command at will or by scheduling the task for a specific time.

Primary Flow: The user using the user interface starts the indexing operation by clicking on a button. The cache is then indexed with the user given feedback about the operation's progress.

Secondary Flow: If the operation fails for some reason, then the user must be informed of the failure using graphical user interface widgets such as a dialog box.

Preconditions: The application must be loaded, Use Case 1.

Use Case Diagram:



4.5.4.5 Low-Level Use Case 5 – Schedule the Indexer

Goal: The user wishes schedule the indexing task.

Actors: User

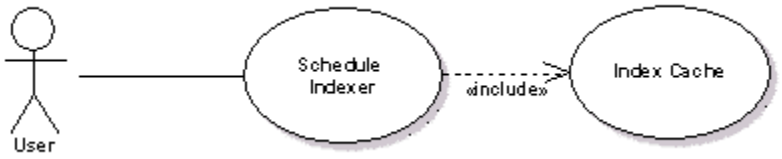
Description: Instead of the user having to start the indexer manually, he/she can select times at which the indexer should run. It will then run in the background with no input required from the user.

Primary Flow: The user sets the times that the indexing task should be scheduled for. The indexer will then run in the background with no other user input required.

Secondary Flow: If an error occurs with indexing, the user should be informed of this through the use of a dialog box.

Preconditions: The application must be running, Use Case 1.

Use Case Diagram:



4.5.4.6 Low-Level Use Case 6 – Disconnect from Gnutella Node

Goal: The user wants to disconnect from a particular Gnutella node.

Actors: User

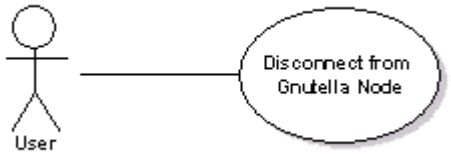
Description: A connection to the Gnutella network is actually a number of connections, one or more to other computers (nodes), using the Gnutella protocol. It is possible for a user to disconnect from one of these nodes and still remained connected to the Gnutella network through other nodes.

Primary Flow: The user attempts to disconnect from a node. The connection is closed and the relevant GUI components are updated to reflect the changes to the network state.

Secondary Flow: If the connection cannot be terminated, the user is informed of this through the use of a dialog box.

Preconditions: The application is running, Use Case 1, and connected to the Gnutella network, Use Case 2.

Use Case Diagram:



4.5.4.7 Low-Level Use Case 7 – Disconnect from Gnutella Network

Goal: The user wishes to disconnect from the Gnutella Network

Actors: User

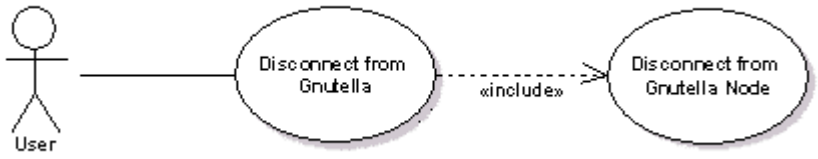
Description: Disconnect from all nodes that the application is currently connected to.

Primary Flow: The user executes a command using the interface that terminates all connections to Gnutella nodes.

Secondary Flow: If the user is not connected to the Gnutella network, then an error message informing the user of this should be displayed.

Preconditions: The application is running, Use Case 1, and connected to the Gnutella network, Use Case 2.

Use Case Diagram:



4.5.4.8 Low-Level Use Case 8 – Close Program

Goal: The user wishes to terminate the program

Actors: User

Description: To terminate the program, the user can simply choose this function using the Interface. Any existing connections to the Gnutella network will be closed before the application is terminated.

Primary Flow: The application is terminated, with the interface no longer displayed and all resources freed.

Secondary Flow: If the application can not be closed because the user has forgotten for example to save his/her updated preferences, then the user is asked to complete this task before the application closes.

Preconditions: The application has been started, Use-Case 1.

Use Case Diagram:

5 Design

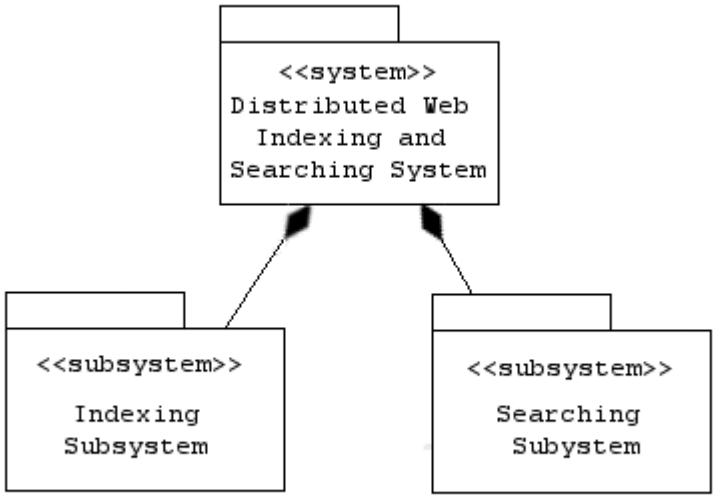
5.1 Overview

The structure and behaviour of the system must be modelled to ensure that the functionality specified in the requirements specification and use-cases can be implemented.

This section begins by describing an abstract view of the application structure. Each identified subsystem is decomposed and the design presented. Following this is the design of the graphical user interfaces that are designed with consideration to the requirements and use-cases presented in the analysis.

5.2 Application Structure

In the analysis section, some work was done to identify the sub-systems and components that constitute this system. The two subsystems indexing and searching are nearly totally independent:



The components that make up the subsystems are summarised by the following table:

Components	*Indexing Subsystem*	*Searching Subsystem*

Components	*Indexing Subsystem*	*Searching Subsystem*
Indexer	Yes	No
Indices	Yes	No
Query Engine	No	Yes
Web Server	No	Yes
Web Interface	No	Yes
Peer-to-Peer Core	No	Yes

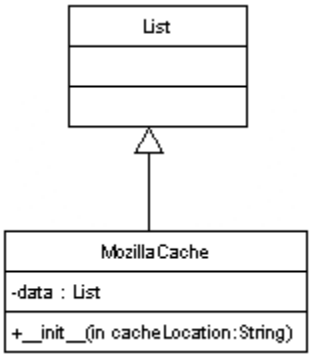
Due to the fact that the indexing and searching subsystems work near independently of each other as shown in the analysis, they are designed independently.

5.3 Indexing Subsystem Design

5.3.1 Cache

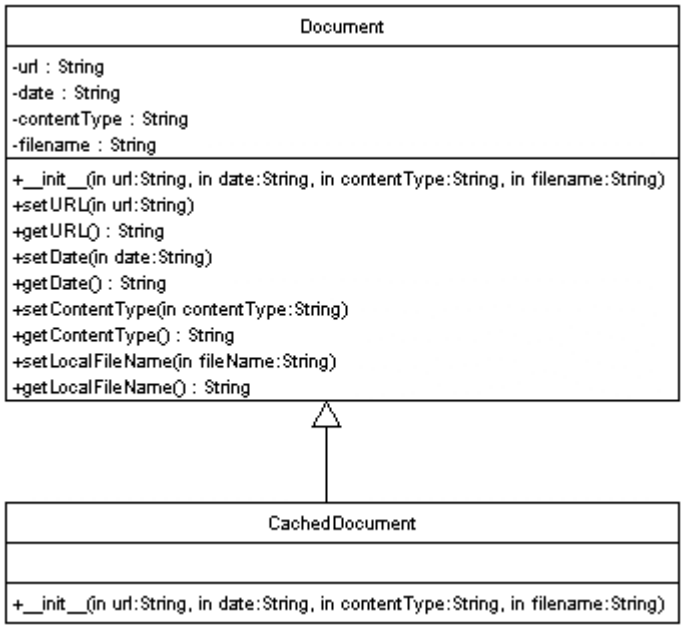
The first part of indexing is to create a representation of the pages in cache. Web browser cache implementations vary and have no standard, yet it is important to offer support for as many browsers as possible. This can be done by using a plug-in style architecture and inheritance.

All browser caches are a list of documents of various content-types and can be represented by the system in this way. A cache class can be designed that extends a List class. The cache class now has the properties of a List and can be used as a structure to store the documents in the Web browser cache.



In this class diagram, MozillaCache is a class that models the Mozilla pre version 0.92 cache implementation. MozillaCache is a subclass of List and therefore behaves exactly like one. MozillaCache is simply a list holding CachedDocument objects.

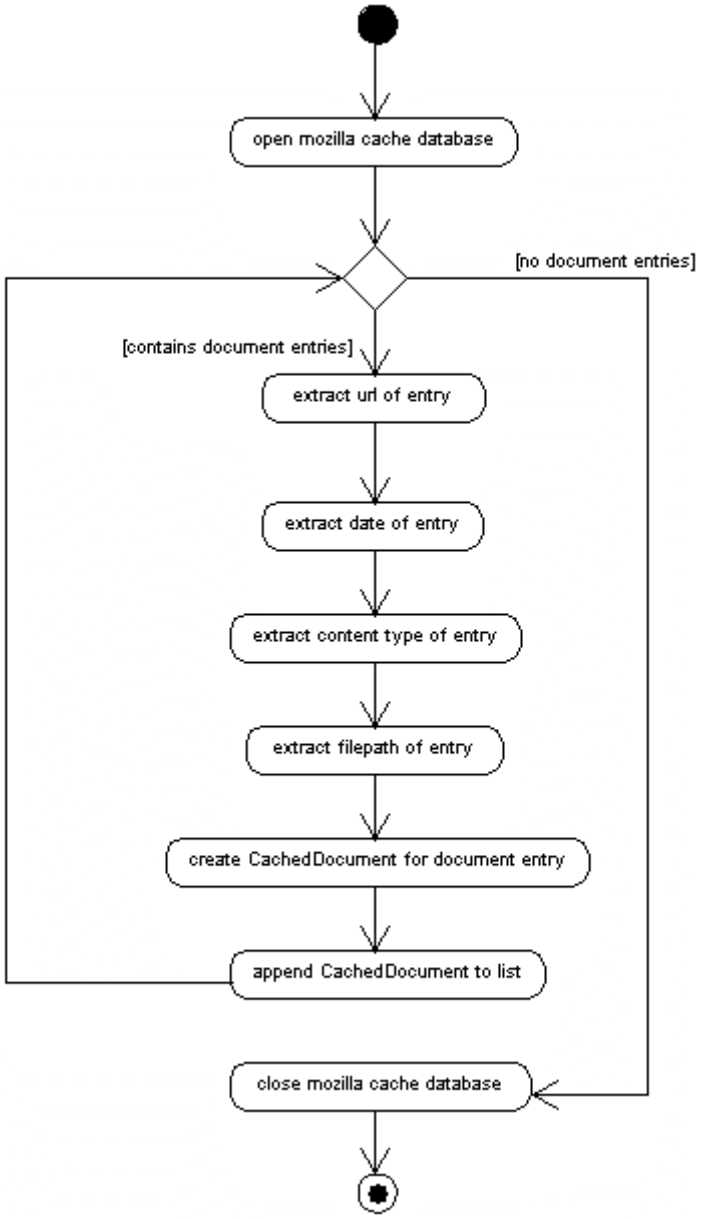
The CachedDocument class is used to model documents in a Web browser cache. A CachedDocument contains information about the document that can be extracted from the Web cache.



Each document in the browser cache can be represented using a CachedDocument object. It contains the URL, date, content-type and filename of the document in cache. The filename is actually the full path to the cached document on the local file system.

MozillaCache is one type of cache implementation. Other implementations can be created for other browsers by simply extending List and ensuring the cache class represents a list of CachedDocuments.

The Mozilla cache implementation uses Berkeley DB 1.85 to store information about cached pages. The activity diagram below shows how the information is extracted from the cache database to create a CachedDocument object.



This activity occurs when the MozillaCache class is instantiated. Once instantiated the MozillaCache will contain a list of CachedDocument objects. Each CachedDocument represents a file cached by the browser.

5.3.2 Parser

All Web pages in the cache must be parsed. The requirements of the parser are:

1. Check that Web pages can be indexed using robot exclusion protocol rules.

2. Title and description meta-data must be extracted from the Web page

3. Extract the text from the Web page.

4. Remove scripting language functions such as Javascript so that this is not indexed.

The parser must parse Web pages written in the HTML format. HTML is a subset of the Standard Generalized Markup Language, SGML and can be parsed using an SGML parser. Python contains an SGML library that includes an SGMLParser. This parser can be extended to create a suitable HTML parser for this system.

The parser can check that a Web page can be indexed two different ways. The first is to examine the robot exclusion Meta tag. The contents of the attributes of this tag can be examined to determine if the page can be examined.

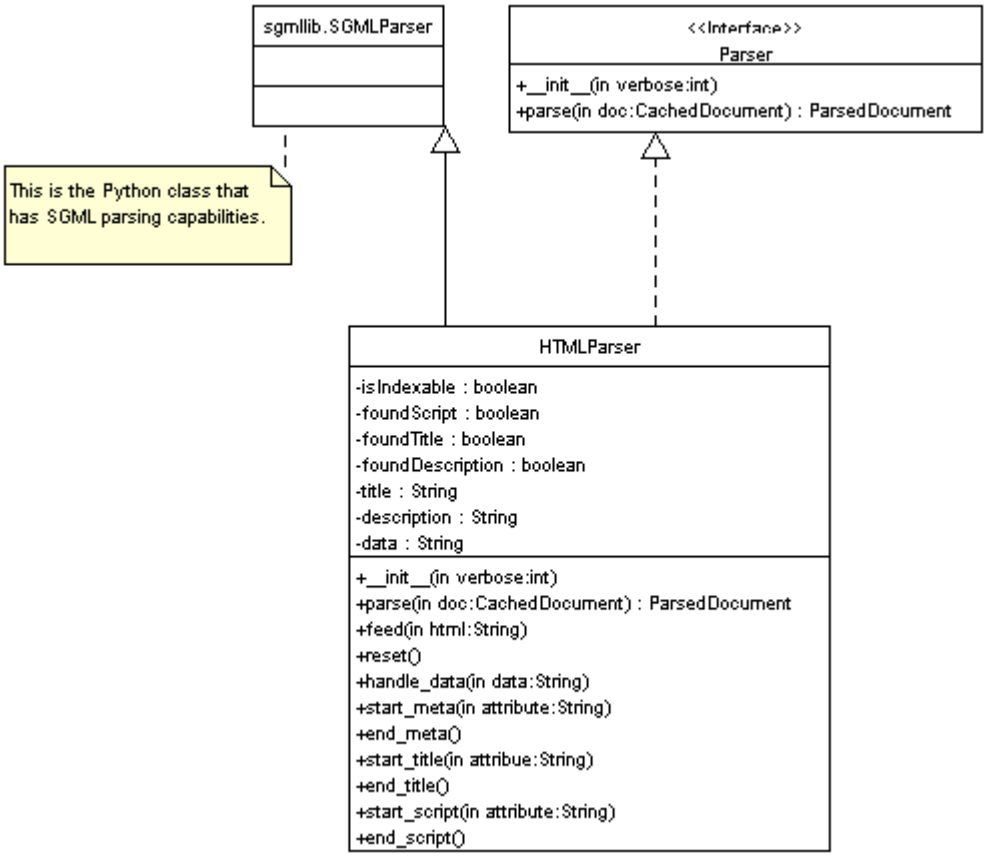
<meta name="robots" content="index,follow">

If the content attribute contains “noindex”, the Web page must not be indexed.

The other is to examine the robots.txt file at the root of the Web site containing the Web page being indexed. The robots.txt file contains information about which pages of a site must not be indexed. Libraries exist to examine this file and will be used by the parser.

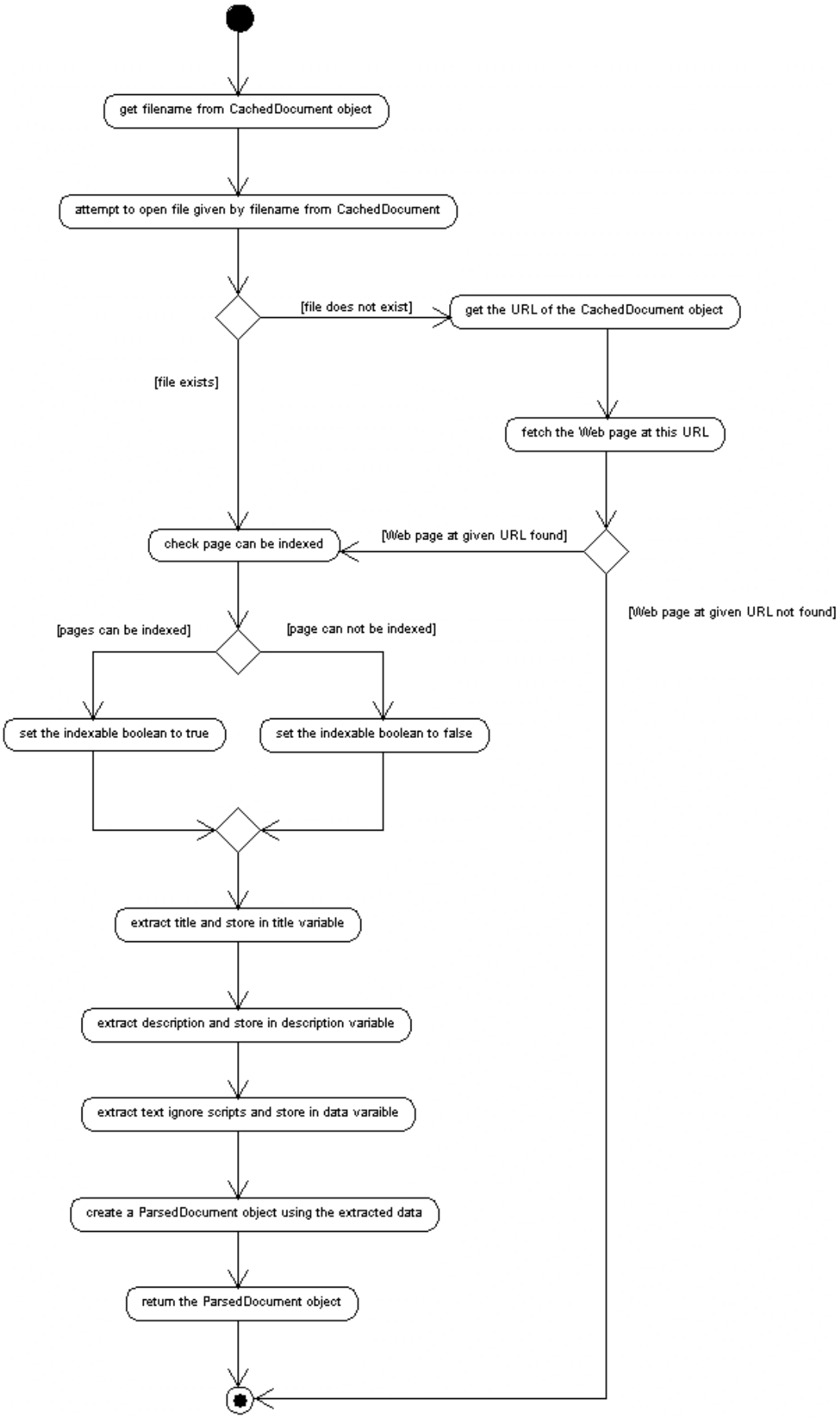
The parser must extract information from the <TITLE> tag for the title of the document and from the content attribute of the description Meta tag, <META name=description content=”a description”>, to obtain a description of the document.

All the text of the document must be extracted from the cached Web pages stored locally. This text will be indexed at a later stage of the indexing. The extracted text is stored as a string for further processing. It is also important to ensure that any scripting language functions used are ignored, as they have no relevance to the content of the page. Text contained in the standard HTML tag for scripting, <SCRIPT>, will be ignored by the parser.

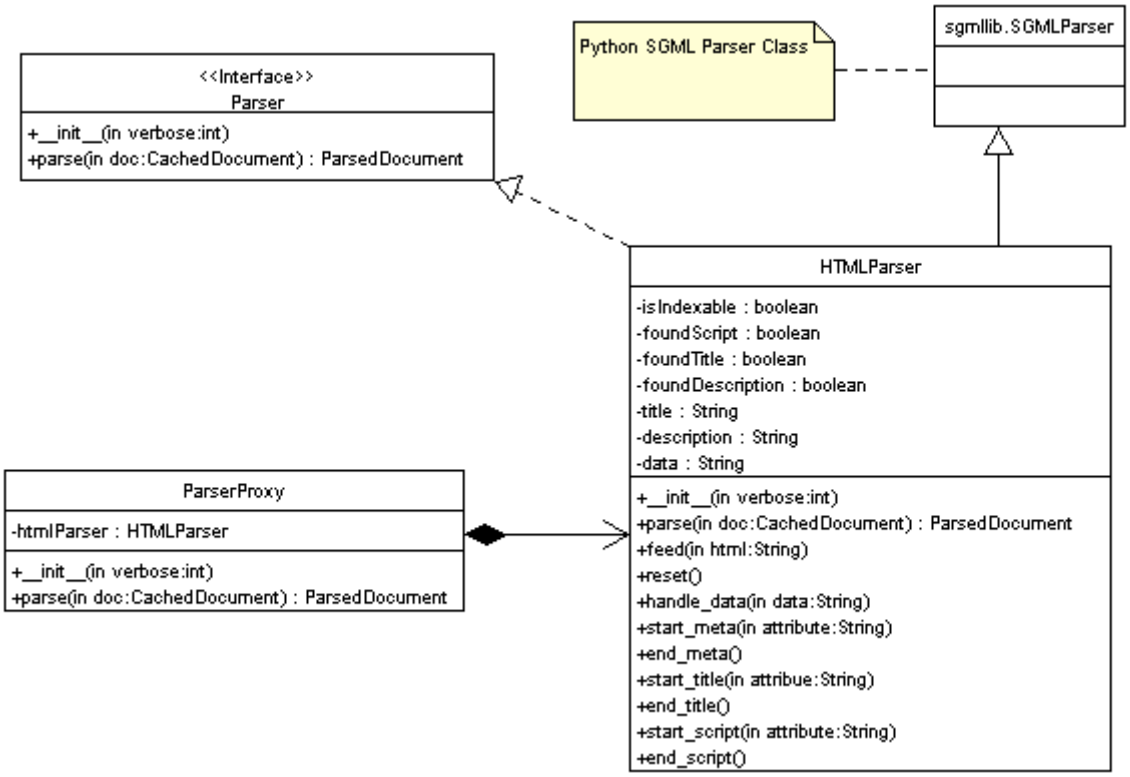


The only element of the above diagram that has not yet been discussed is the Parser interface. Parsers must implement this interface so the system can utilise them.

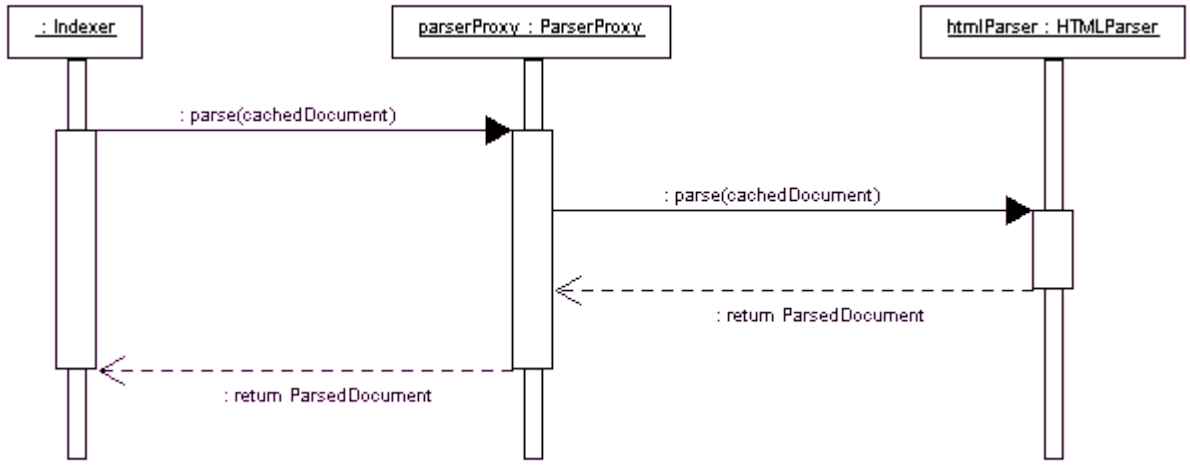
The details of the HTML parsing process is best described by the activity diagram below:


get filename from CachedDocument object
attempt to open file given by filename from CachedDocument
[file does not exist]
get the URL of the CachedDocument object
[file exists]
fetch the Web page at this URL
check page can be indexed
[Web page at given URL found]
[Web page at given URL not found]
[pages can be indexed]
[page can not be indexed]
set the indexable boolean to true
set the indexable boolean to false
extract title and store in title variable
extract description and store in description variable
extract text ignore scripts and store in data varaible
create a ParsedDocument object using the extracted data
return the ParsedDocument object

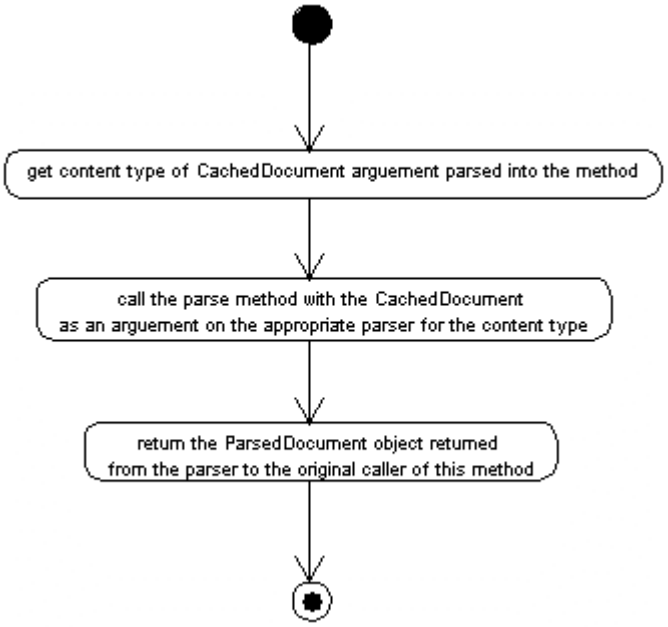
The ParserProxy class is used to act as a proxy between the indexer and the parsers.



The class diagram shows the structure of the ParserProxy and it's relationship to the HTMLParser.

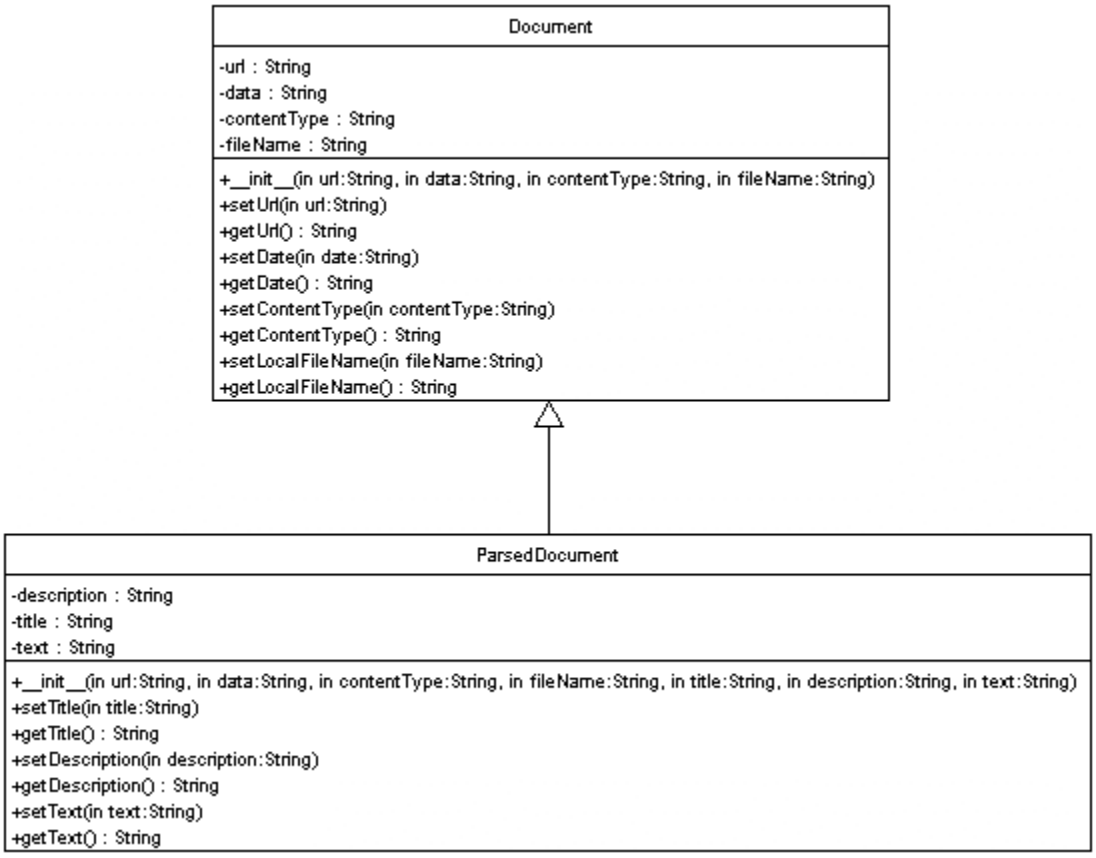


This diagram shows the behaviour of the ParserProxy. When a CachedDocument object needs to be parsed, the indexer passes it as an argument of the parse method of the ParserProxy. The ParserProxy then decides which parser to pass the CachedDocument object to for parsing. The following activity diagram further explains this behaviour:



Additional parsers can be added to the ParserProxy by creating a class that implements the Parser interface. This new parser class would then have to be registered with the ParserProxy. This is done by ensuring the ParserProxy instantiates it.

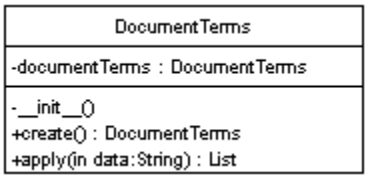
The ParserProxy takes as input a CachedDocument object as an argument to the parse method. This method returns a ParsedDocument. The ParsedDocument contains all the necessary information required about a document so it can later be normalised.



The ParsedDocument is a type of Document and so extends the Document class. In addition to the URL, date, content type and file name of the document being parsed, it also contains information extracted by the parsing process, the title, meta description and text of the document.

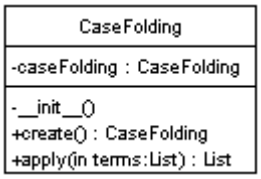
5.3.3 Transformation Manager

The text contained in a document (ParsedDocument) must be split into terms. The definition of a term in this system is a word of a maximum of 256 alphanumeric characters. The string containing the text of a document in a ParsedDocument can be split into a list of terms. These terms can then be normalised. This behaviour can be implemented as a function but Object-Oriented principles are used, so the Singleton pattern is utilised.

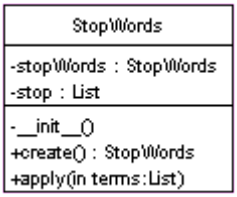


To reduce the size of an index, documents to be indexed can be normalised. There are three normalisation techniques, case folding, removal of stop words and stemming. Each of these are functions that can be modelled using the Singleton pattern. This ensures only one instance of the class is created and avoids the need for static functions.

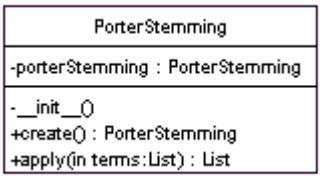
Case Folding simply means converting each term of the document into lower case. The argument of the apply method, is a list of terms to be case folded. A list of case folded terms is returned by this method.



Stop words are a list of the most commonly used words in the English language. By removing these words from documents, the size of the document and index are reduced greatly. The argument of the apply method, is a list of terms. A list of terms without any stop words is returned by this method.

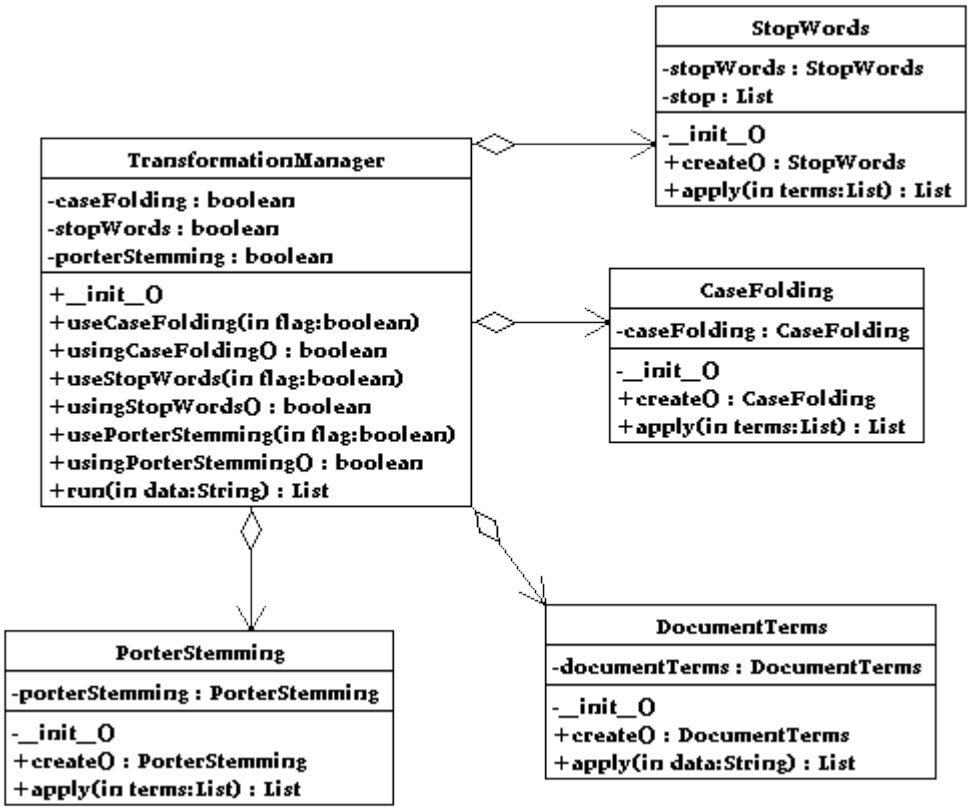


Stemming is the process of reducing a word to it's canonical form, e.g. going becomes go. The argument of the apply method, is a list of terms. A list of terms with all terms reduced to canonical form is returned by this method.

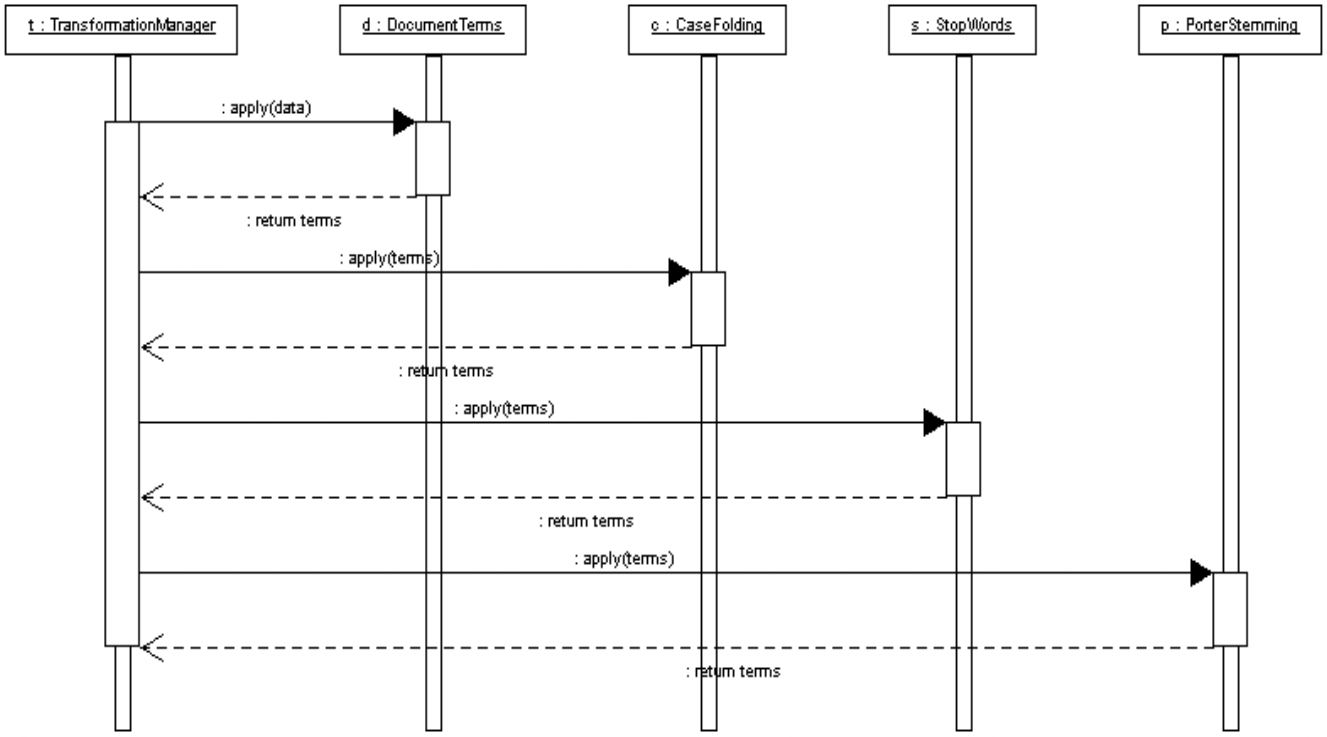


It can be argued that with modern compression techniques such as gzip and zlib, the techniques above are not necessary as indexes can be significantly compressed [2]. However, it is important to ensure the index size is kept to a minimum, as this software is intended to run on standard PC workstations.

The TransformationManager class described below allows users to choose the normalisation techniques used. When indexing takes place, only selected normalisation techniques will be applied. The TransformationManager contains an instance of the classes used to perform normalisation.



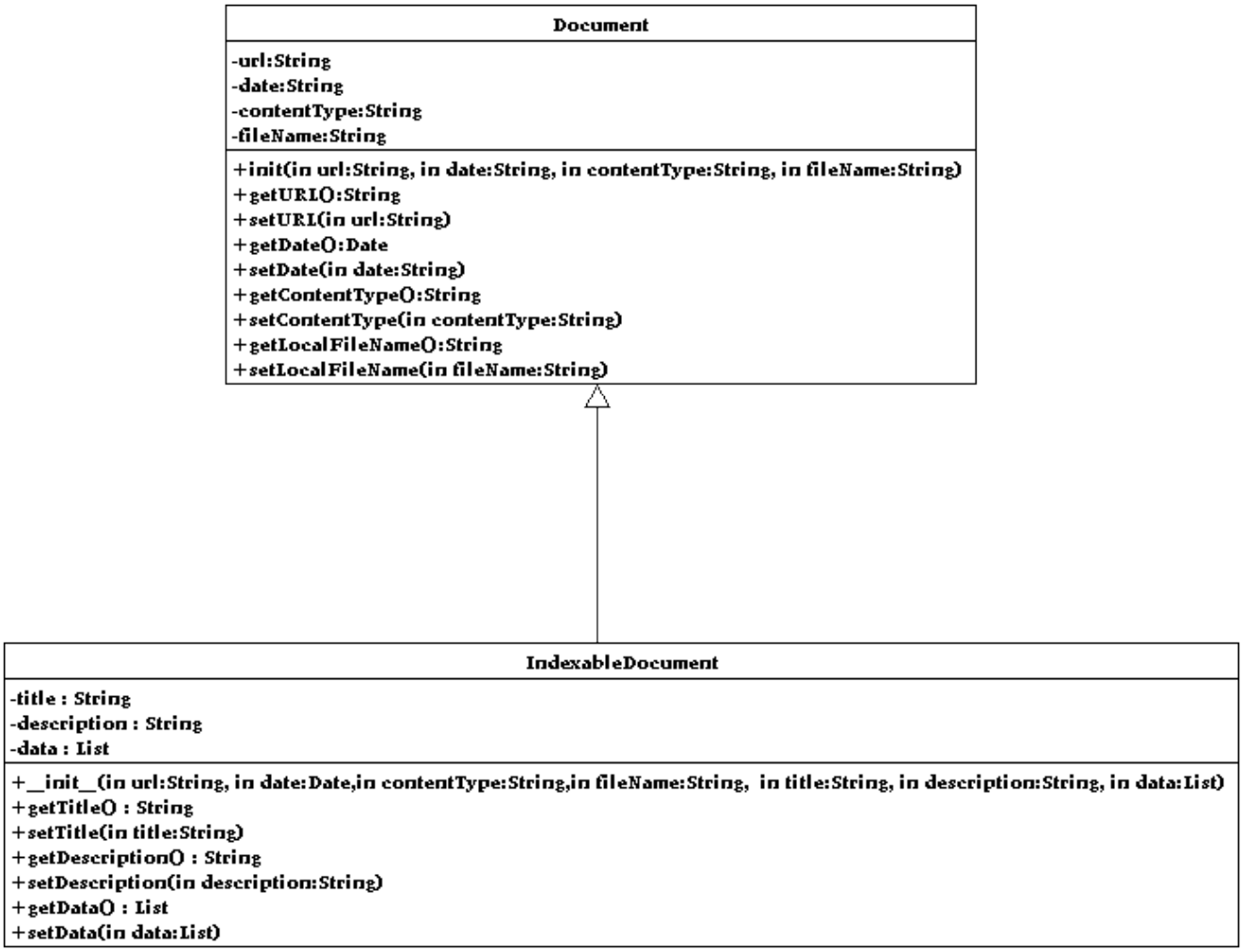
The sequence diagram shows how a ParsedDocument is normalised by the TransformationManager creating a document that can then be indexed, an IndexableDocument.



First, the document text is converted to a list of term by the apply method of DocumentTerms. Then case folding, removal of stop words and stemming are applied in order.

Once the TransformationManager has applied the necessary transformations to the data attribute of a ParsedDocument object (this is the text of the cached Web page), an IndexableDocument is created and returned.

The IndexableDocument has all the required information for an index entry for the cached document that this IndexableDocument represents.



5.3.4 Indexing

The indexing process creates the indices used to provide results for search queries. The indices must contains the following information:

The index consists of three files, lexicon, document and inversion.

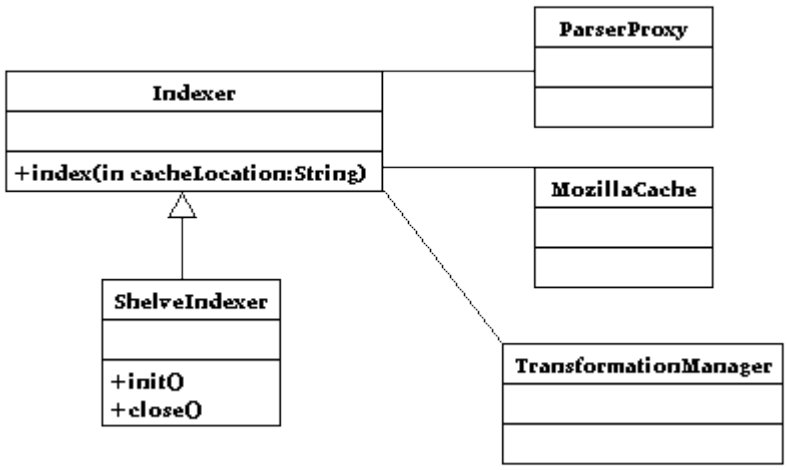
The lexicon is a list of tuples. Each tuple stores the word and the frequency of the word over all documents indexed.

The document file simply contains a list of tuples. Each tuple contains information about a document. A tuple is of the format (document URL, document title, document description, date document was cached, document content type, local file path where document can be found). All of this information is obtained from the IndexableDocument object.

The inversion contains a list of tuples of the following type, (word, inversion entry). Each word in the

lexicon has an inversion entry in this file. The inversion entry is a list of tuples of the type (document id, frequency of term in document).

The index can be stored in three different ways. A flat-file, DBM such as BerkeleyDB or relational database such as MySQL can be used. It is for this reason that the indexing algorithm and index opening and saving functionality are separated. This separation allows for the same indexing algorithm to be used with different physical index implementations. The class Indexer will contain the core indexing functionality, but will not open or save indices. Classes will need to index this Indexing class and implement open and close methods containing code specific to the implementation of the index.



The class diagram shows the ShelveIndexer class. This class uses serialisation to save the systems representation of the index to physical indexes using DBM. The class also shows that the Indexer communicates with three other classes. The ShelveIndexer uses the constructor __init__ to open the physical indexes. It also implements a close method to close access to them and release any locks. It inherits from Indexer, so is able to use the index method of its parent to run the indexing algorithm.

There are many algorithms that exist that produce indices. The easiest to implement and most common is the In-Memory inversion algorithm. Only primary memory is used in this application, which means it is extremely fast when compared to those that use secondary memory. The disadvantage is that systems often have much less primary than secondary memory. This is a problem in systems indexing many gigabytes of information centrally, but due to the distributed nature of this system, this is not the case. A browser cache is often a maximum of a hundred of megabytes and most modern computer systems can cope with this demand, many without resorting to virtual memory. It is also assumed that users will want the indexing to finish as quickly as possible.

The In-Memory inversion algorithm from Moffat et al. (1999) page 228 modified for use with the class presented in this design. This algorithm will be implemented in the index method if the Indexer class.

1. /* Initialization */

Create an empty dictionary structure S.

2. /* Phase one – collection of term appearances */

   Create a MozillaCache instance.

   For each document D_d in the MozillaCache, $1 \leq d \leq N$,

   (a) D_d = Parse the document by calling ParserProxy.parse(D_d)

   (b) D_d = parse the document into terms and normalise by calling TransformationManager.run(D_d)

   (c) D_d is now an Indexable Document.

3. /*Phase two – output of inverted file */

   For each term $1 \leq t \leq n$ in D_d.getData() (list of terms in document)

   (a). Start a new inverted file entry

   (b). For each (d, $f_{d,t}$) in the list corresponding to t, append (d, $f_{d,t}$) to this inverted file entry.

4. Create index on disk. Subclasses must handle this.

5.4 Searching Subsystem Design

5.4.1 Peer-to-Peer Network Core

Searches and query results are sent and received using the Gnutella peer-to-peer network. To design, implement and test a Gnutella library for use in this project would consume a great deal of time. This could result in the project being delayed, so an existing Gnutella library should be re-used. There are many Gnutella libraries available in most languages including those considered for the implementation of this system at the analysis phase, Java and Python. The libraries are also often freely available with source code.

The specific Gnutella API discussed here is the JTella API, http://www.kenmccrary.com/jtella/. The site also contains JavaDoc documentation for the library. The library can be used for connecting to Gnutella, disconnect from the network, searching and replying to result. The library provides all the necessary Peer-to-Peer functionality.

The API has three classes which are of key importance, GNUTellaConnection, SearchSession, and MessageReceiverAdapter.

GNUTellaConnection represents a connection to the Gnutella network. This connection is composed of NodeConnections which are connection to Gnutella nodes. The connection to the network can be started and stopped using the start() and stop() methods respectively. Connections to nodes can be added using the addConnection(ipAddress, port) method.

A GNUTellaConnection can listen for search queries by associating via a method call an instance of a class implementing MessageReceiverAdapter to it. This class then allows the peer to handle query, push and query reply messages. When listening to query replies only the method handling queries should be implemented. In the case of this system, when a query is received, this method would pass the query to an instance of the QueryEngine. The returned value, a list of ResultDocument objects, which would then be sent in a message to the node performing the search.

The SearchSession class allows a search to be created for a given query string. The SearchSession can be instantiated and given a query string. The query can then be sent to connected nodes and replies can be received by creating a class that extends MessageReceiverAdapter and implements the query reply method.

5.4.2 HTTP Server

The role of the HTTP Server is to serve the Web page search interface. This Web page uses a form to allow the user to submit a query. The form will use the GET method, so the server must support this method. When the query is submitted of the following URL type, the server searches the system using the query for the given time in minutes.

http://<server>:<port>/search?query<terms>&time=<time in minutes>

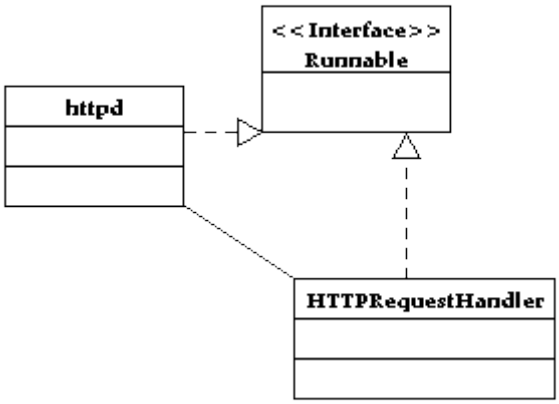
<terms> = the terms + sign separated.

<time in minutes> = an integer specifying the time in minutes the search is to last for.

The HTTP server uses the Connection Pool pattern to ensure that the HTTP Server can handle a certain number of connections at a given time.

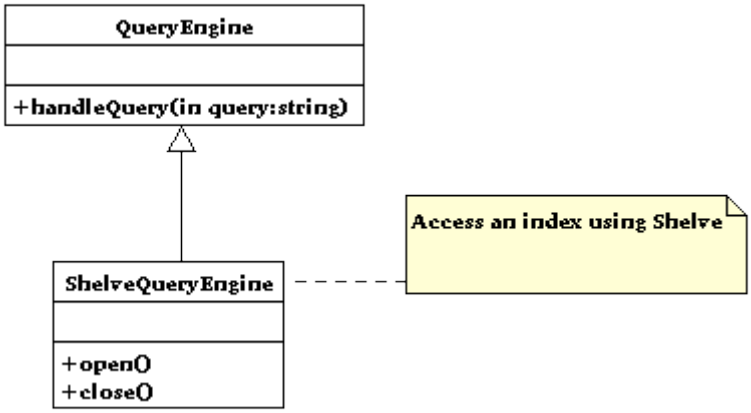
The httpd class is the server implementation. Its main purpose is to run as a service and wait for incoming connections. Upon receiving a client request, it uses an HTTPRequestHandler thread to manage the connection.

The HTTPRequestHandler Class is a class that handles the connections to clients. It is responsible for sending the correct pages to users, searching the system and displaying the results.

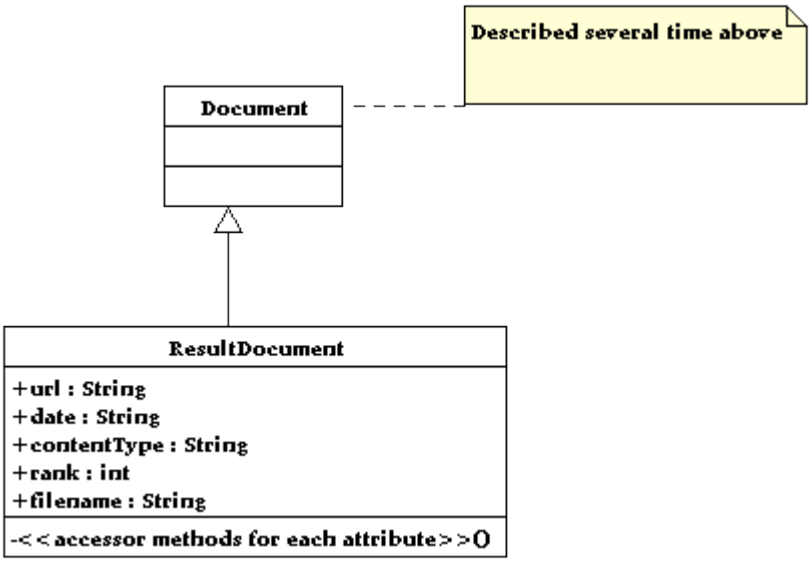


5.4.3 Query Engine

The query engine receives a query and finds documents in the local index that match the query. For the same reasons as the Indexer, the query class contains only the algorithm to find query matches from a dictionary/hash table structure representation of the index. Classes must extend the QueryEngine class and provide the functionality that loads the indices into the dictionary structures so they can be used by the QueryEngine.



ResultDocuments are objects are objects that contain all the information about a result. A set of result documents is sent to the source node of the query as the result set of the query.



5.5 Graphical User Interface

5.5.1 Design Options

The graphical user interface will be to implement a means for each use case of the use-case model to be implemented. There are two possibilities that exist for the design and implementation of the graphical user interface, each is described below.

The first is to have one application responsible for all these tasks implemented using toolkits such as Swing, GTK or QT. The significant disadvantage of this is that the user would have to submit searches using this interface, however he/she is certainly going to be familiar with and comfortable using a Web page to search for Web pages. All existing Web search engines work in this way, so this is not a good option.

The alternative to this is to split the user interface into two sections. The first is will be the main program. It will allow the user to manage connections to the Gnutella network and index the cache. When this application is running, using a web browser, the user can access the search interface which will be Web based. This Web interface will allows the user to search the system and view any results in an identical fashion to standard Web search engines.

5.5.2 The Web Interface

The interface consists of Web pages for searching and viewing results, however, more work is required than simply creating these pages. A poorly designed interface will hinder the user in completing their

goal, in this case, finding sites relevant to their query. Standard Human-Computer Interaction issues help in creating a usable interface, but there are other principles specific to information retrieval that must be considered, such as the interaction model, starting points, query specification and context. All of these will addressed during the design of the interface. At the end of this section, a proposed interface is shown.

5.5.2.1 Human-Computer Interaction

HCI design principles are useful tools in the design process of Web pages. Principles that should be considered are browser compatibility, monitor resolutions and colour depths, font types and size, use of colours and the use of images.

One of the requirements is that if possible the software product should be multi-platform. This means that the interface will potentially be viewed by many different browsers and must appear the same in each view. The three main browsers, Microsoft Internet Explorer, Netscape and Opera all support HTML 4.0, so only standard HTML tags and no browser specific ones will be used in the coding of pages. This rule will ensure that the pages when viewed in these browsers will appear similar to each other.

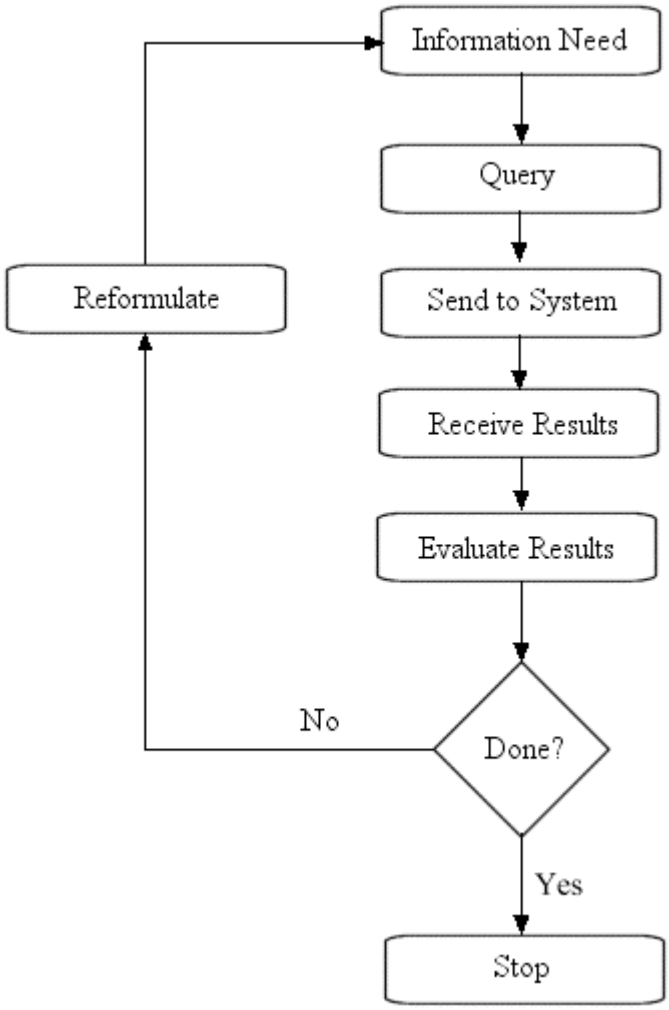
The resolution that the Web pages are designed for must be suitable for the majority of most monitors. Statistics from show that 800x600 is the most common resolution, so the pages will be designed for this. The same statistics also show that more than 10% of Web users still use 8-bit colour displays, therefore it is important that Web pages use 8-bit colour to ensure compatibility.

The choice of colours to be used on the page is important. If they do not contrast, they will be difficult to see on the page. The three main colours used on the page will be white (background), black (text), blue (hyperlinks). Hyperlinks will turn from blue to purple when visited so the user knows he/she has visited them before. Images will be used sparingly so that they do not clutter the interface. The ALT tag must be set for all images, so that an alternative text description is displayed in text-only browsers such as Lynx.

The aesthetic view of the interface has been considered, but the structure of the site is also an important factor. Most Web search engines have successfully used the scaffolding technique to structure their interface. Two interfaces are provided to the user, simple and advanced. The simple interface is the first to be displayed to the user and is the most commonly used because of its ease of use. The advanced interface is more complicated to use and requires learning, but it allows more complex queries to be generated. Although Nielsen (http://www.use-it.com/), recommends only offering one simple interface, scaffolding will be used in the interface of this system.

5.5.2.2 Information Access Process

The creation of a good information retrieval interface requires some understanding of the interaction model between user and interface. The standard model described in [1] is used by most web search engines and is shown in the diagram below.



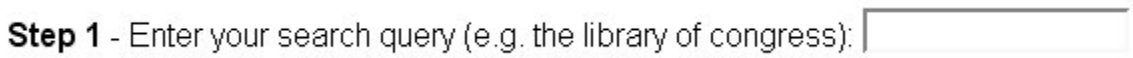
The user interface abides by the simple information access model above as follows:

Information Need – The user visits the Web interface with an information need.
Query – The user specifies the query and selects how long they wish the search to last.
Send to System – The user submits the query by clicking a button to the system.
Receive Results – The results to the query are displayed on a new Web page.
Evaluate Results - The user can evaluate the results returned, visiting sites in the results.
Done? - If the user finds a site in the results that satisfies their information need, then goal has been completed.
Reformulate - If the user is not satisfied with the results returned, he/she will reformulate the query and the process starts afresh.
Stop – The use closes the interface with the search complete.

This process may be commonly used but has its problems. Nielsen states that most users give up searching for what they are looking for if they cannot find it the first time. This goes against the reformulation part of the model.

5.5.2.3 Starting Points

The starting point in this interface is the search page. A good starting point must help users specify a query. To do this, the interface must include a link to a help page if the user requires assistance in getting started. Each widget on the page will also be accompanied by short string aiding the user in specifying a query.



5.5.2.4 Query Specification

There are two interface related problem with query specification. The first is type of query to be used, boolean or free text and the second is the technique used for query specification. Schneiderman proposes 5 methods for the latter, command language, form filling, menu selection, direct manipulation and natural language. A combination of these will be used for the interface.

Boolean queries are used heavily in bibliography retrieval systems, in particular by librarians [1]. Ranking of results cannot be applied to a boolean search, so the results are usually displayed in chronological order. This is a disadvantage as users expect results from a search to be ranked with the best results first. Users only look at the first twenty results from a search and if they cannot find what they want, they usually give up. The other problem with boolean queries is that they are often difficult to formulate. An example of a user specifying an incorrect boolean is query is given in [1]:

> 'dogs and cats' may imply a request for documents about dogs and documents about cats, rather than both topics at once. [1]

Research shows that users tend to have very limited knowledge of query formulation even though search engines are the most commonly used tool on the Web.

The alternative to boolean queries is free-text. The user simply submits a list of words that are related to their search goal. A user searching for Web pages about Brighton & Hove Albion Football Club would formulate a query such as "Brighton and hove football club seagulls". Seagulls being the nickname of the club. Free-text queries appear to offer less control than boolean queries as the user cannot specify which words must appear in the query and which words are more important than others, but there are solutions to this. Users can place inverted commas around words that must be contained in result documents and the + sign is used to indicate that a word in a query has greater importance.

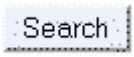
Due to the fact that boolean queries are complicated for users, I have decided to use free-text in the simple search page and offer complicated boolean queries in the advanced search.

Queries will specified and submitted using HTML Forms. Form widgets include text input boxes, drop-down menus, lists and buttons.

The simple search requires the user to enter a query, select how long they wish the search to last and a means to submit the search to the system. The best way to allow the user to enter a query is by using a text box. The user simply enters the query into the box.

The user will also need to specify how long the search will last for. One option is to allow the user to enter the time into a text box, but the disadvantages of this is that some users may choose times that are too small or too large and they may enter in a time in seconds when they system expects a time in minutes. The better alternative is to use a drop-down box, also known as combo-box. A string next to the box will indicate the time is in minutes format.

To submit the query, a button with the word search on it will be used. Buttons are widely used for form submission on the Internet and the user will most likely be used to this concept.



The advanced search page is more complicated than the simple version. The page is split into two sections. The first allows boolean queries, and the second free-text ones. The latter is identical in design to the simple page. The boolean search allows users to specify

5.5.2.5 Context

The result of a search is a set of documents. The set of documents alone in the results page is not useful. The user will want to know additional information such as the URL of the document, the title and a description. In order to achieve this, the result set will be represented using HTML on a new Web page, but in the same browser instance as the search was made. Each document in the result set will contain the following information:

- The title of the homepage will be displayed; hyperlinked to the URL of the site the result refers to.
- A description of the site will be displayed. This description of the site will be taken from the description META tag. If not available, no description will be displayed.
- The URL of the site will be shown, but it a considerably smaller font as knowing the URL is not a vital part of the information access model and code like strings may confuse the user.

5.5.3 The Standalone Application Interface

The rest of the application can be integrated into a standalone application with it's own user interface. An important and commonly used design pattern for the design of user interfaces is the Model View Controller pattern, MVC. The model is the data displayed by the user interface component, the view. Controllers are used to trigger events that update the view when the model has changed and vice-versa. The Java Swing toolkit has support for the MVC pattern built into every component, so custom

components supporting the model do not have to built. The MVC model is not used for the design of the entire interface. Instead, parts of the interface use the MVC model where appropriate. An example of it use is as follows.

The Connection view shows a list of Connections. The data for this view is in the JTella library, so the JTella ConnecionList that stores the list of connections can be modified to extend a Model so the system knows it is one. Then the model is registered with a JTable like component. Finally, listeners are added to the model and view. One listener added to the model will update the view when the model changes, the listener may have to update the view when a connection fails. The other will update the model when the view changes, perhaps when the user disconnects from a node.

The key criteria for the design of this interface are that it must allow users to execute the functionality described in the analysis use cases and requirements.

6 Distributed Ranking

6.1 Overview

This section describes the method used in this system for the implementation of ranked query results, the cosine measure. The cosine measure is stated and then the implementation and challenges faced during this are discussed.

6.2 The Cosine Measure

The cosine measure is a means of comparing two documents for similarity. Documents and queries can be represented as n-dimensional vectors. The angle between these vectors can then be calculated. The smaller the angle between the two, the greater the similarity.

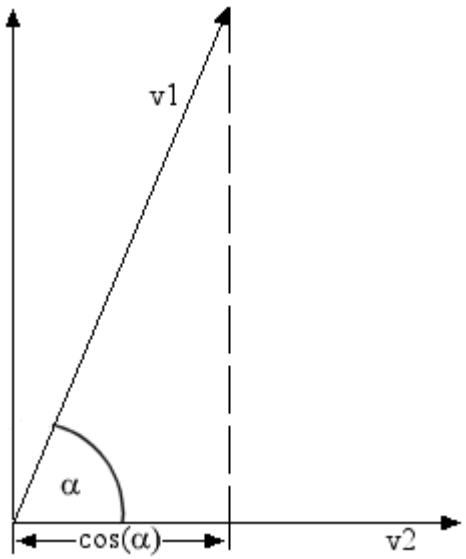


Figure reproduced from Monz & De Rijke(2001), page 15.

6.3 Distributed Ranking

Moffat et al (1999) give an algorithm that performs the cosine measure for a query against indexed document.

To retrieve r documents using the cosine measure,

Set $A \leftarrow \{\}$. A is the set of accumulators.

For each query term $t \in Q$,

Stem t.

Search the lexicon.

Record f_t and the address of I_t, the inverted file entry for t.

Set $w_t \leftarrow 1 + \log e(N, f_t)$

Read the inverted file entry, I_t.

For each $(d, f_{d,t})$ pair in It,

If $A_d \notin A$ then

Set $A_d \leftarrow 0$

Set $A \leftarrow A + \{A_d\}$

Set $A_d \leftarrow A_d + \log_e(1+f_{d,t}) * w_t$.

For each $A_d \in A$,

Set $Ad \leftarrow A_d/W_d$.

A_d is now proportional to the value $\text{cosine}(Q,D_d)$.

For $1 \leq i \leq r$.

Select d such that $A_d = \max\{A\}$

Look up the address of the document d.

Retrieve the document d and present it to the user.

3.Set $A \leftarrow A - \{A_d\}$

Moffat et al(1999) p 202

This algorithm is easy to perform in a centalised environment as all the values can be calculated from the central index.

> "maintain no central information and simply accept the similarity scores that arise from the use of local weights at each librarian. Retrieval effectiveness may suffer because all of the term weights used are local, but flexibility is maintained and the use of network resources is minimized."
>
> Moffat et al. (1999) p 220

The problem is that 2(d) gives a local value for the weight of a term, w_t, and not a global one including other indices. However, it is possible to implement the cosine ranking for a Peer-to-peer network.

This algorithm is performed by the peer searching the system.

Submit query to peers

Wait for reply messages

For each (t, f_t) where t is term in reply messages:

accumulate f_t values for each term

Send message containing list of terms with their global frequencies, f_t

Receive the ranked document set.

This algorithm is performed by nodes replying to queries (performing the cosine measure).

Set $A \leftarrow \{\}$. A is the set of accumulators.

For each query term $t \in Q$,

Stem t.

Search the lexicon.

Record f_t and the address of I_t, the inverted file entry for t.

Return all query terms with f_t values to node submitting query.

Receive message from query peer, for each term t in the message.

Set $w_t \leftarrow 1 + \log e(N, f_t)$

Read the inverted file entry, I_t.

For each $(d, f_{d,t})$ pair in It,

If $A_d \notin A$ then

Set $A_d \leftarrow 0$

Set $A \leftarrow A + \{A_d\}$

Set $A_d \leftarrow A_d + \log_e(1+f_{d,t}) * w_t$.

For each $A_d \in A$,

Set $Ad \leftarrow A_d/W_d$.

A_d is now proportional to the value $\text{cosine}(Q,D_d)$.

For $1 \leq i \leq r$.

Select d such that $A_d = \max\{A\}$

Look up the address of the document d.

Retrieve the document d and present it to the user.

Set $A \leftarrow A - \{A_d\}$

Four messages are used to implement the centralised algorithm on a Peer-to-Peer network.

Message	*Description*
0	A message containing a tuple (message ID, query) is sent to peers on the network.
1	Peers reply with a message containing a tuple (message ID, list of (term, term frequency))
2	A message containing a tupe of (message ID, dictionary of term to global term frequency values mapping)
3	A message containing the document result set is sent to the peer querying the system.

The messages have a message ID 0-3 so that the peer knows how to respond appropriately. For example, a peer receiving message ID 1 knows to reply with a message of type 3.

The messages are implemented using Python's serialization technique which allows objects and primitive types to be stored as a String. Messages 0 and 2 can be implemented by serializing the tuples described above and placing the string in the search criteria of a query message. Messages 1 and 3 can be implemented by serializing the tuples described in the table and placing the resulting string in the file name attribute of a record set of a query hit message. Below is an extract from the Gnutella protocol document detailing these two messages:

Query (0x80)
Minimum Speed Search criteria
Byte offset 0 1 2 …
Minimum Speed
The minimum speed (in kb/second) of servents that should respond to this message. A servent receiving a Query descriptor with a Minimum Speed field of n kb/s should only respond with a QueryHit if it is able to communicate at a speed >= n kb/s
Search Criteria
A nul (i.e. 0x00) terminated search string. The maximum length of this string is bounded by the Payload_Length field of the descriptor header.

QueryHit (0x81)
Number of
Hits Port IP Address Speed Result Set Servent
Identifier
Byte offset 0 1 2 3 6 7 10 11 … n n + 16
Number of
Hits
The number of query hits in the result set (see below).
Port *The port number on which the responding host can accept incoming connections.*
IP Address *The IP address of the responding host.*
This field is in big-endian format.
Speed *The speed (in kb/second) of the responding host.*
Result Set *A set of responses to the corresponding Query. This set contains Number_of_Hits elements, each with the following structure:*
File Index File Size File Name
Byte offset 0 3 4 7 8 …
File Index *A number, assigned by the responding host, which is used to*

uniquely identify the file matching the corresponding query.
File Size *The size (in bytes) of the file whose index is File_Index.*
File Name *The double-nul (i.e. 0x0000) terminated name of the file whose index is File_Index.*
The size of the result set is bounded by the size of the Payload_Length field in the Descriptor Header.
Servent Identifier
A 16-byte string uniquely identifying the responding servent on the network. This is typically some function of the servent's network address. The Servent Identifier is instrumental in the operation of the Push Descriptor (see below).
QueryHit descriptors are only sent in response to an incoming Query descriptor. A servent should only reply to a Query with a QueryHit if it contains data that strictly meets the Query Search Criteria.
The Descriptor_Id field in the Descriptor Header of the QueryHit should contain the same value as that of the associated Query descriptor. This allows a servent to identify the QueryHit descriptors associated with Query descriptors it generated.

Clip 2, Gnutella Protocol Document 0.4, http://www.clip2.com/GnutellaProtocol04.pdf

Other alternative implemntations included using C structs, but this would have caused problems. C structs can contain null characters for padding purposes. This could have caused message corruption as Gnutella uses null characters for string termination in query and query hit packet. Padding could have been misinterpreted as string termination.

7 Implementation

7.1 Programming Language

The intention from the start was to use Java or Python to implement the design. Unfortunately, two key problems with using either of these languages arouse.

1. Working Java Gnutella libraries such as JTella exist, but the Java Berkeley DB API seemed to have difficulties accessing the Mozilla cache database.

2. Python has a Berkeley DB module, bsddb, that can easily access data held in the Mozilla cache database, but no fully working Gnutella libraries could be found.

At first, the solution to the problem appeared to be Jython. Jython is Python implemented using only Java. Jython programs can use Java libraries, so a Java Gnutella library could be used to implement the Peer-to-Peer functionality. Jython programs can also support Python libraries, so it was assumed the Python Berkeley DB module could also be used. However, the latter was not the case as the bsddb module uses Berkeley DB C implementation libraries and this is something Jython cannot support.

Referring back to the design, two subsystems make up the system, indexing and searching. The indexing subsystem requires a working Berkeley DB library, but not any Gnutella functionality. The searching subsystem requires a working Gnutella library but does not need Berkeley DB support. By using Python to implement the indexing and Jython to implement the searching the problem can be resolved. The index created by the indexing subsystem has to be accessed by the searching subsystem, so it must implemented using a format accepted by both Python and Jython.

7.2 Peer-to-Peer Implementation

Rather than design and implement a Gnutella library for the systems Peer-to-Peer functionality, the decision was taken to reuse an existing one. With Jython chosen as the language of choice for the search subsystem, a Java Gnutella library was needed. A working and simple API called JTella was used. JTella is described in more detail in the Design section.

7.3 Index Implementation

The index had to be implemented using a format supported by both Jython and Python. There were three possible choices for index implementation, flat-file, DBM or a relational database.

A flat-file solution is easy to implement and when compression such as zlib and gzip is used, the file can be reduced to a small size. However, the entire flat-file must be loaded into primary memory for it to be used, unlike databases. This is a large cost and for this reason this technique was not chosen.

DBMs such as Berkeley DB offer lightweight database support to programmers through programming functions. There is no complicated set-up of the database and knowledge of SQL is not required. Hashing functions are used to store and retrieve items from a database.

Relational databases are powerful database systems, but perhaps too complicated for use in this project. The installation size of relational databases can be very large and this is not suitable for an application designed for use on standard desktop PCs. Other disadvantages are the use of some times complex SQL queries and the need for drivers to enable a program to communicate with a relational database.

The DBM solution was chosen for the implementation of the indices. GNU GDBM was specifically chosen. Unfortunately, Microsoft Windows support for DBMs is non-existent so the flat-file solution was also implemented to offer support to potential Windows users.

7.4 Graphical User Interface Implementation Issues

There are two interfaces of the system as described in the design. The first is a Web interface used for searching the system and viewing results. This interface is served by the httpd class in the searching subsystem. The other is a standard standalone application loaded using the operating system responsible for network management and indexing. It is this application that starts the HTTP Server so the Web interface can be accessed. It is also this interface that is used to manage network connections and index the cache. If this interface is implemented using Jython, how can it index the Web browser cache if the indexing subsystem is implemented in Python? Two possible solutions exist:

1. XML-RPC
2. Run the Indexing as separate process

XML-RPC allows distributed applications to communicate with each other using remote procedure calls in XML. This could be used to allows the Jython implemented GUI to run the indexing process. The second option is however much simpler to implement and is used in the implementation.

Using Java's Runtime.exec with Jython, it is possible to run the indexer as a separate process. The output of this process can be read and displayed using the GUI to inform the user of the indexing status.

7.5 Deployment

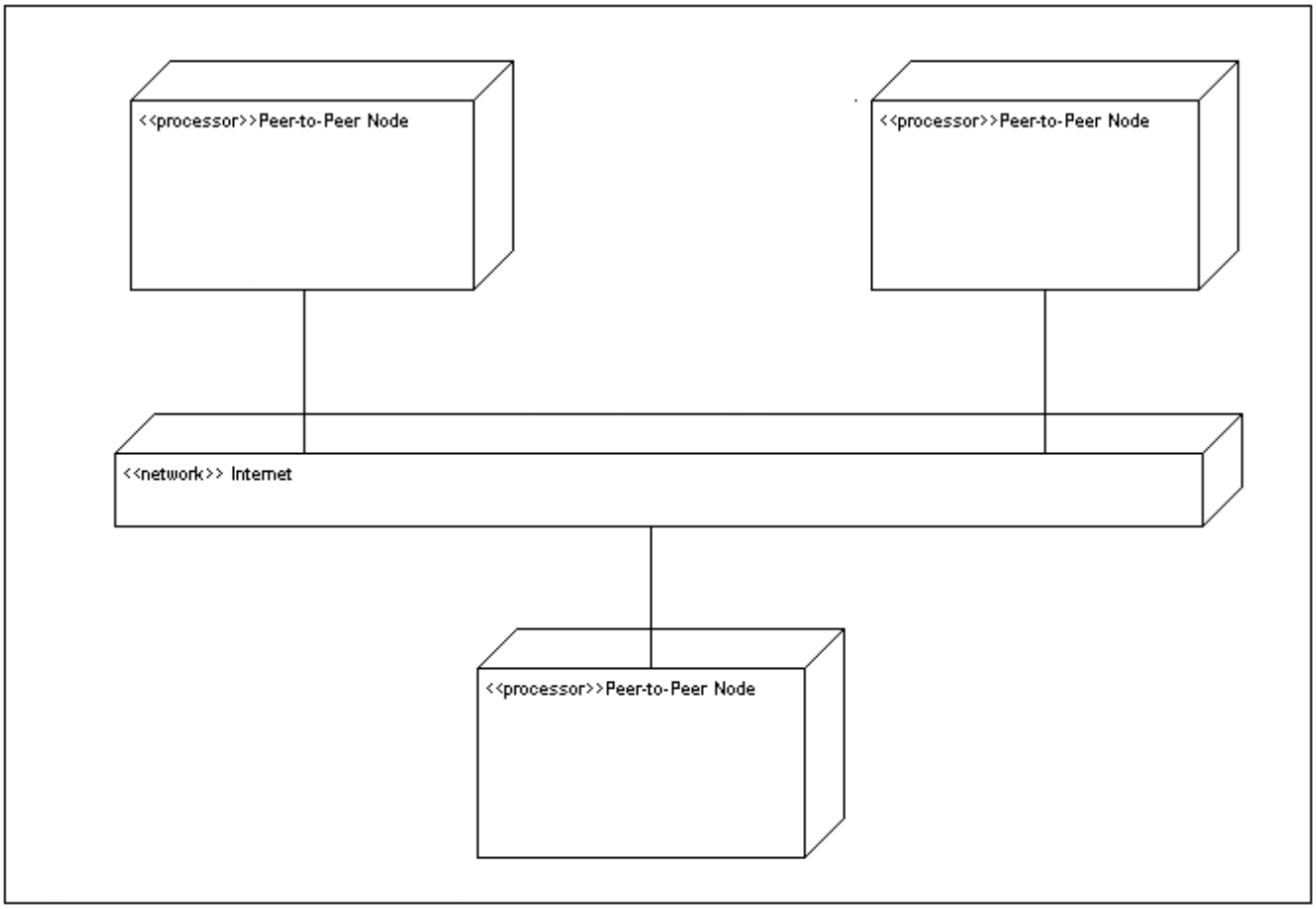
The components specified were deployed on three Sun Microsystems servers, tsunx.ctn.cogs.susx.ac.uk (IP Address 139.184.50.12), tsunb.ctn.cogs.susx.ac.uk (IP Address 139.184.50.11) and tsunc.ctn.cogs.susx.ac.uk (IP Address 139.184.50.14) running SunOS 2.7. Each server participated as a node in a Peer-to-Peer Gnutella network. Internet connections were used to connect to the nodes. This deployment was made for testing purposes. The components can be deployed on any computer with the following specification:

- Unix Operating System including Linux, FreeBSD and Solaris.

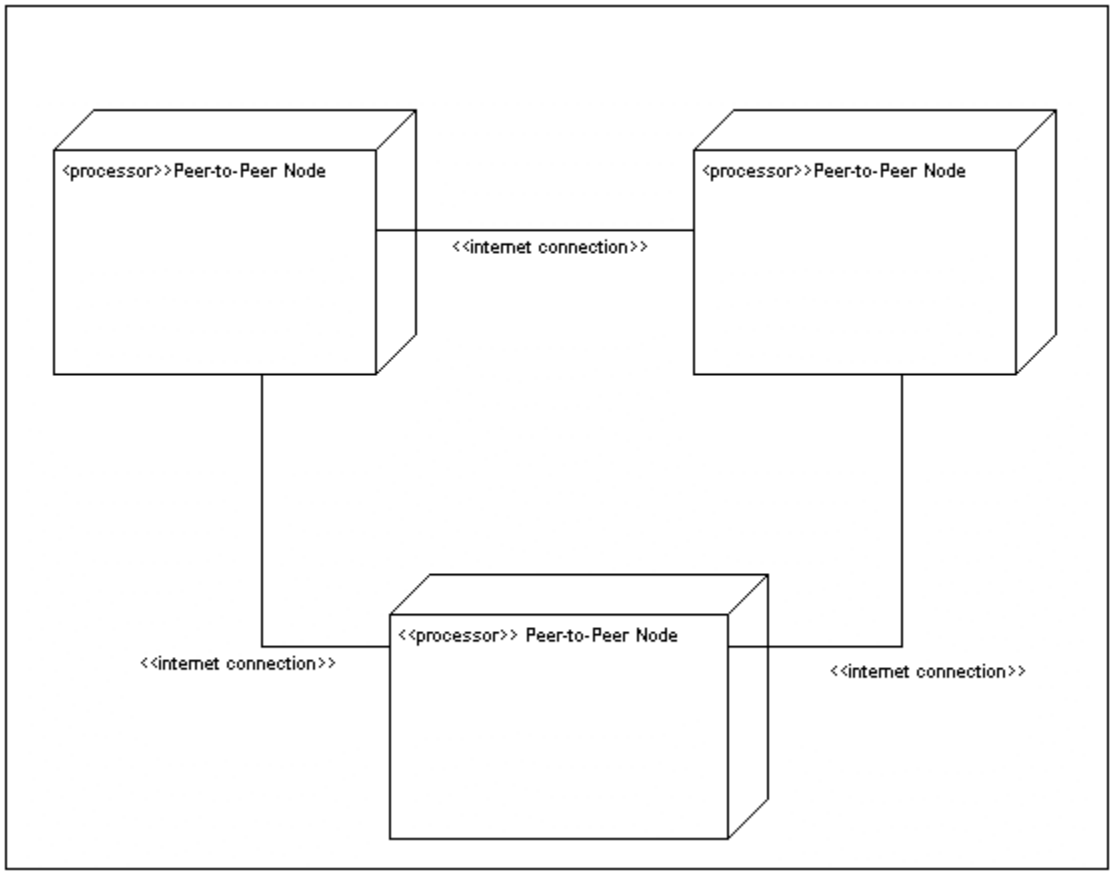
- Python 2.2
- Jython 2.1
- Java Standard Development Kit or Runtime Environment 1.3 or higher.
- An Internet or Local Area Network Connection

There are no hardware requirements, but the application will perform faster on systems with more CPU and main memory. In particular, this will greatly improve the performance of the indexing of the Web pages, as this is a CPU and primary memory intensive task. Computers with limited hard disk space may have trouble storing the indices.

Each node in the deployment diagram executes the components described above. This diagram shows the system topology:



This deployment diagram shows that the computers executing the components are connected to the Internet. The diagram hides the fact that the nodes use the Internet to connect directly to each other. The next deployment diagram shows the topology more clearly.



It is important to note that this describes the deployment used for the testing. In reality, Peer-to-Peer nodes will connect and disconnect and network connection may fail and restart. Also there is likely to be many more nodes as part of the system topology. This results in the system topology being dynamic and impossible to document fully.

8 Testing

8.1 Overview

The testing section primarily consists of black-box tests, these are tests that do not consider the internal workings, but focus on fulfilling user requirements. Two types of tests are conducted, normal and extreme value. Normal value tests uses data that users will enter into the system on a regular basis. Extreme value tests uses data that is expected to cause problems such as boundary values.

8.2 Use-Case Testing

8.2.1 Test 1 - Use Case 1 Testing

Use Case Description: Start Program

Test Number	1.1
Test Type	Normal Value Testing
Test Description	The program is executed.
Expected Result	The system starts and the application interface is displayed on screen.
Actual Result	As expected.
Test Status	Passed
Test Failure Analysis	Not required.

Test Number	1.2
Test Type	Extreme Value Testing
Test Description	The program is executed, however required components at start-up are missing.
Expected Result	The system does not start and an error message is displayed.
Actual Result	The system does not start and an error message is displayed.
Test Status	Passed.
Test Failure Analysis	Not Required.

8.2.2 Test 2 – Use Case 2 Testing

Use Case Description: Connect to Gnutella Network

Test Number	2.1
Test Type	Normal Value Testing
Test Description	A connection attempt is made to the Gnutella peer, hostname **Error! Hyperlink reference not valid.**, port 6346.
Expected Result	A successful connection is made to the Gnutella peer and the GUI is updated to reflect this.
Actual Result	As expected
Test Status	Passed.
Test Failure Analysis	Not Required

Test Number	2.2
Test Type	Extreme Value Testing
Test Description	A connection attempt is made to a this.server.does.not.exist.com on port 6346, hostname. , port 6346.
Expected Result	An attempted connection to this server is made, but fails.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not Required.

8.2.3 Test 3 – Use Case 3 Testing

Use Case Description: Submit Search

Test Number	3.1
Test Type	Normal Value Testing
Test Description	A search for COGS is submitted to the system with the search taking a maximum of 1 minute.
Expected Result	A list of search results to the query COGS, with the COGS homepage as the first result.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not required.

Test Number	3.2
Test Type	Extreme Value Testing

Test Number	3.2
Test Description	An empty search is made for 1 minute.
Expected Result	A results page with empty result set is displayed in the browser.
Actual Result	A result pages is not displayed.
Test Status	Failed.
Test Failure Analysis	This test failed because the system could not handle the boundary value of 0. The failure occurred when the system attempted to calculate the ranking

8.2.4 Test 4 – Use Case 4 Testing

Use Case Description: Index the Cache

Test Number	4.1
Test Type	Normal Value Testing
Test Description	The system attempts to index the Web browser cache.
Expected Result	The Web browser cache contents are indexed and the user is informed when the process has finished.
Actual Result	As expected
Test Status	Passed
Test Failure Analysis	Not required.

Test Number	4.2
Test Type	Extreme Value Testing
Test Description	An incorrect value for the Web browser cache is given and the system is attempts to index the cache.
Expected Result	An error message indicating to the user the indexing failed.
Actual Result	No error message.
Test Status	Failed.
Test Failure Analysis	The indexing is run as a separate python process. The output of this process is read and displayed to the user. However only standard output stream is read and not an error stream. Any error messages from the Python process cannot be read and therefore cannot be displayed in the interface.

8.2.5 Test 5 – Use Case 5 Testing

Use Case Description: Schedule the Indexer

Test Number	5.1
Test Type	Normal Value Testing
Test Description	The indexer is scheduled to index the Web browser cache one hour from the time the test began.
Expected Result	The indexer runs in one hour.
Actual Result	As expected.
Test Status	Passed
Test Failure Analysis	Not Required.

8.2.6 Test 6 – Use Case 6 Testing

Use Case Description: Disconnect from Gnutella Node

Test Number	6.1
Test Type	Normal Value Testing
Test Description	An attempt is made to disconnect an existing connection to the Gnutella node, tsunx.ctn.cogs.susx.ac.uk port 6346.
Expected Result	The connection to tsunx.ctn.cogs.susx.ac.uk is disconnected and the GUI is updated to reflect this.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not required.

Test Number	6.2
Test Type	Extreme Value Testing
Test Description	An attempt is made to disconnect, but no connection has been selected for disconnection.
Expected Result	An error message is displayed informing the user that the disconnection attempt failed.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not Required.

8.2.7 Test 7 – Use Case 7 Testing

Use Case Description: Disconnect from the Gnutella Network (i.e. Disconnect from all connected nodes).

Test Number	7.1
Test Type	Normal Value Testing
Test Description	While connected to the Gnutella network, an attempt is made to disconnect from the entire network.
Expected Result	All connections to the Gnutella network are terminated and the GUI is updated to reflect this.
Actual Result	Connections were not terminated and the GUI continued to show connection to Gnutella were alive.
Test Status	Failed.
Test Failure Analysis	The application netstat was used to list network connections while connected to Gnutella and after the disconnect command had been issued. It showed a Gnutella connection to tsunx.ctn.cogs.susx.ac.uk on port 6346 from my computer system active before and after the test. TCP goran:3336 tsunx.ctn.cogs.susx.ac.uk:6346 ESTABLISHED The most likely cause of this is an error in the JTella, Java Gnutella API used or incorrect interpretation of an API method from this library.

Test Number	7.2
Test Type	Extreme Value Testing
Test Description	An empty is made to disconnect from the Gnutella network when not connected to the Gnutella network.
Expected Result	This operation should not be allowed by the system.
Actual Result	The application does not allow the operation.
Test Status	Passed.
Test Failure Analysis	Not Required.

8.2.8 Test 8 – Use Case 8 Testing

Use Case Description: Close the Program

Test Number	8.1
Test Type	Normal Value Testing
Test Description	An attempt is made to close the program.
Expected Result	An attempt is made to close the program. After giving confirmation of this action, the program terminates.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not required.

8.3 Requirements Specification Testing

8.3.1 Test 9 – Requirements Specification 1

Requirement: The user must be able to initiate the indexer at will or by scheduling the task.

Test: This requirement is covered by test 4 and 5.

8.3.2 Test 11 – Requirements Specification 2.1

Requirement: Web pages stored on the local file system must not be indexed, as these are not accessible externally.

Test: A file on the local system was accessed using the Mozilla Web Browser, file://tsunx/home/ug/cs99/stephenn/index.html. A test application was then coded which showed that this file was not cached by the browser. Both the HTML page and test application code can be found in the test evidence section.

8.3.3 Test 12 – Requirements Specification 2.2

Requirement: Web pages that must not be indexed because they use the robot exclusion rules.

Test:

Test Number	12.1
Test Type	Normal Value Testing
Test Description	A Web page that cannot be indexed because it uses the robot exclusion meta tag is visited using the Mozilla browser and cached. An attempt is made to index the cached page.
Expected Result	The page is not indexed.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not required.

Test Number	12.2
Test Type	Extreme Value Testing
Test Description	An attempt is made to index a page using the robot exclusion meta tag, but the tag does not prohibit indexing.
Expected Result	The pages are indexed.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not required.

8.3.4 Test 13 – Requirements Specification 2.3

Requirement: Secure pages accessed using HTTPS to protect user privacy.

Test: This test is not required as Web pages accessed using HTTPS. The Mozilla FAQ of the Caching Mechanism in the 331 release, http://www.mozilla.org/docs/netlib/cachefaq.html, states, "no disk caching of SSL objects takes place".

8.3.5 Test 14 – Requirements Specification 3

Requirement: Only the software generated index and no other files will be made accessible to external users of the system.

Test: A test is not needed, as the application contains no file sharing functionality. User cannot download files from other users, so no files are accessible.

8.3.6 Test 15 – Requirements Specification 4

Requirement: A Web page will not be duplicated in the index. If a user visits http://www.google.co.uk/ twice, this site will not appear twice in the index.

Test: The Mozilla caching system will only have one cached instance of a given Web page at a given time. Therefore, it is not possible for the system to index a cached Web page twice.

8.3.7 Test 16 – Requirements Specification 5.1

Requirement: The result set can be empty or non-empty.

Test: Refer to test 3.2

8.3.8 Test 17 – Requirements Specification 5.2

Requirement: Duplicate results must be discarded.

Test:

Test Number	17.1
Test Type	Normal Value Testing
Test Description	Two indexes were searched on two different nodes were searched for COGS. Each containing the COGS homepage.
Expected Result	The COGS homepage is listed in the results once.
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not required.

8.3.9 Test 18 – Requirements Specification 5.4

Requirement: If possible, results should be ranked with the best listed first.

Test:

Test Number	18.1

Test Number	18.1
Test Type	Normal Value Testing
Test Description	Two indexes were created with one document each. One index contained the Web page, http://www.cogs.susx.ac.uk/ and the other contained http://www.google.com/. The search query "COGS Google" was sent to both indexes.
Expected Result	The COGS homepage is listed in the results before the Google page, because the former contains references to both COGS and Google, but the latter only contains references to Google
Actual Result	As expected.
Test Status	Passed.
Test Failure Analysis	Not required.

9 Conclusions and Further Work

The two primary aims of this project have been successfully completed. The created application indexes Web pages visited by users and allows these distributed indices to be searched. Many of the use cases and requirements specification points were also successfully completed, but more work is needed before this project is released for use. The source code is insufficiently commented and the logic of the internal workings of the systems has not been tested (i.e. white-box testing). Another important consideration is that tests were performed on a very small local area network. Further tests are required to test the system in a more realistic environment, a wide area network such as the Internet.

There are also ethical issues yet to be discussed with the most important of these being privacy. Many users may not use this system as some may be worried about other files apart from the files in the Web cache directory being indexed. Also users may feel that such a system could associate an individual to the Web sites he/she visits. Releasing the source code may help to resolve this problem as users would be able to read the source to disprove their fears.

There is much more work that needs to be done to the system, code improvement and further testing to name a few. The system could also be extended to index Web documents other than HTML ones. At present the system only supports the English language, but this could be changed to support more languages. Finally none of the extensions given in the analysis have been implemented.

In conclusion, I feel this project has been a success, but much more work is yet to be done before such a system could rival traditional centralised search engines.

10 Bibliography

[1] Baezer-Yates, Ribeiro-Neto et al. Modern Information Retrieval. Addison-Wesley. 1999
[2] Witten, Moffat and Bell. Managing Gigabytes. Morgan Kaufmann Publishers. 1999
[3] Various Authors. Peer to Peer. O'Reilly. 2001
[4] Booch, The Unified Modelling Language User Guide. Addison-Wesley. 1999
[5] Lutz and Ascher. Learning Python.O'Reilly. 1999
[6] Lutz. Python Programming. O'Reilly. 1999
[7] Sian Baldwin. Agent Manager Database, BSc Artificial Intelligence with Management Studies Project. University of Sussex. 2001
[8] Peter Pimley. Jukebox: An arbitrary processor simulator. BSc Computer Science Project. University of Sussex. 2001

11 Appendices

11.1 Appendix A - Source Code

cache.py file

```
import re
import bsddb
import string
import document
from UserList import UserList

class MozillaCache(UserList):

    #returns a list of CachedDocument objects
    def __init__(self, cacheDirectory):
        UserList.__init__(self)
        if not cacheDirectory.endswith("/"):
            cacheDirectory = cacheDirectory + "/"
        self.data = []
        reHttp = re.compile("http", re.IGNORECASE)
        reDate = re.compile("date: (.+)", re.IGNORECASE)
        reType = re.compile("content-type: (\S+/\S+)$", re.IGNORECASE)
        reFile = re.compile(cacheDirectory+"(.+)$", re.IGNORECASE)
        allowedLetters = string.uppercase + string.lowercase \
                                    + string.digits + string.punctuation \
                                    + string.whitespace
        try:
            db = bsddb.hashopen(cacheDirectory+"cache.db", "r")
        except(bsddb.error):
            raise CacheNotFoundError(cacheDirectory+"cache.db")
        records = db.keys()
        for record in records:
            contentType = ""
            date = ""
            file = ""
            if reHttp.match(record):
                try:
                    fields = string.split(db[db[record]], "\r\n")
                except(KeyError):
                    continue
                for field in fields:
                    matchType = reType.search(field)
                    matchDate = reDate.search(field)
                    matchFile = reFile.search(field)
                    if matchType:
                        contentType = matchType.group(1)
                    elif matchDate:
                        date = matchDate.group(1)
                    elif matchFile:
                        file = ""
                        for c in matchFile.group(1):
                            if allowedLetters.find(c)>-1:
                                file = file + c
                            else:
                                continue
                self.data.append(document.CachedDocument(record, date, contentType,
cacheDirectory+file))
            else:
                continue

#standard error
class Error(Exception):
```

```
    def __init__(self, value):
        self.value = value

    def __str__(self):
        return `self.value`

#cache not found
class CacheNotFoundError(Error):

    pass
```

document.py file

```
class Document:

    def __init__(self, url, date, contentType, fileName):
        self.__url = url
        self.__date = date
        self.__contentType = contentType
        self.__fileName = fileName

    def getURL(self):
        return self.__url

    def setURL(self, url):
        self.__url = url

    def getDate(self):
        return self.__date

    def setDate(self):
        self.__date = date

    def getContentType(self):
        return self.__contentType

    def setContentType(self, contentType):
        self.__contentType = contentType

    def getLocalFilename(self):
        return self.__fileName

    def setLocalFilename(self, fileName):
        self.__fileName = fileName

    def __str__(self):
        return self.__url + "," + self.__date + "," + self.__contentType + "," +
self.__fileName

class CachedDocument(Document):

    def __init__(self, url, date, contentType, fileName):
        Document.__init__(self, url, date, contentType, fileName)

class ParsedDocument(Document):

    def __init__(self, url, date, contentType, fileName, title, descr, text):
        Document.__init__(self, url, date, contentType, fileName)
```

```
        self.__descr = descr
        self.__title = title
        self.__text = text

    def __str__(self):
        return Document.__str_(self) + \
             self.__title + self.__descr + self.__data

    def getDescription(self):
        return self.__descr

    def setDescription(self, descr):
        self.__descr = descr

    def getTitle(self):
        return self.__title

    def setTitle(self, title):
        self.__title = title

    def getText(self):
        return self.__text

    def setText(self, text):
        self.__text = text


class IndexableDocument(Document):

    def __init__(self, url, date, contentType, fileName, title, descr, data):
        Document.__init__(self, url, date, contentType, fileName)
        self.__descr = descr
        self.__title = title
        self.__data = data

    def __str__(self):
        return Document.__str__(self) + \
             self.__title + self.__descr + self.__data

    def getDescription(self):
        return self.__descr

    def setDescription(self, descr):
        self.__descr = descr

    def getTitle(self):
        return self.__title

    def setTitle(self, title):
        self.__title = title

    def getData(self):
        return self.__data

    def setData(self, data):
        self.__data = data

class ResultDocument(Document):
```

```
    def __init__(self, url, date, contentType, fileName, title, descr, rank):
        Document.__init__(self, url, date, contentType, fileName)
        self.__descr = descr
        self.__title = title
        self.__rank = rank

    def __str__(self):
        return Document.__str__(self) + "," + self.__title + self.__descr

    def getTitle(self):
        return self.__title

    def setTitle(self, title):
        self.__title = title

    def getDescription(self):
        return self.__descr

    def setDescription(self, descr):
        self.__descr = descr

    def setRank(self, rank):
        self.__rank = rank

    def getRank(self):
        return self.__rank
```

gdbmShelve.py file (modified from shelve)

```
try:
    from cPickle import Pickler, Unpickler
except ImportError:
    from pickle import Pickler, Unpickler

try:
    from cStringIO import StringIO
except ImportError:
    from StringIO import StringIO

class Shelf:
    """Base class for shelf implementations.

    This is initialized with a dictionary-like object.
    See the module's __doc__ string for an overview of the interface.
    """

    def __init__(self, dict):
        self.dict = dict

    def keys(self):
        return self.dict.keys()

    def __len__(self):
        return len(self.dict)

    def has_key(self, key):
        return self.dict.has_key(key)
```

```
    def get(self, key, default=None):
        if self.dict.has_key(key):
            return self[key]
        return default

    def __getitem__(self, key):
        f = StringIO(self.dict[key])
        return Unpickler(f).load()

    def __setitem__(self, key, value):
        f = StringIO()
        p = Pickler(f)
        p.dump(value)
        self.dict[key] = f.getvalue()

    def __delitem__(self, key):
        del self.dict[key]

    def close(self):
        try:
            self.dict.close()
        except:
            pass
        self.dict = 0

    def __del__(self):
        self.close()

    def sync(self):
        if hasattr(self.dict, 'sync'):
            self.dict.sync()

class DbfilenameShelf(Shelf):
    """Shelf implementation using the "anydbm" generic dbm interface.

    This is initialized with the filename for the dbm database.
    See the module's __doc__ string for an overview of the interface.
    """

    def __init__(self, filename, flag='c'):
        try:
            import gdbm
        except ImportError:
            import jgdbm as gdbm
        Shelf.__init__(self, gdbm.open(filename, flag))


def open(filename, flag='c'):
    """Open a persistent dictionary for reading and writing.

    Argument is the filename for the dbm database.
    See the module's __doc__ string for an overview of the interface.
    """

    return DbfilenameShelf(filename, flag)
```

httpd.py file

```
from java.lang import *
from java.io import *
from java.net import *
from java.util import *
from com.kenmccrary.jtella import *
from com.kenmccrary.jtella.SearchReplyMessage import *
from urlparse import urlparse
import jarray
import string
import cPickle
import time


class httpd(Runnable):

    def __init__(self, root, port, timeout, gnutella):
        rootFile = File(root)
        httpd.root = root

        if not rootFile.exists():
            raise httpderrror("Server root does not exist")

        httpd.port = port
        httpd.gnutella = gnutella


    def run(self):
        serverSocket = ServerSocket(httpd.port)

        while 1:
            client = serverSocket.accept()
            handler = Thread(HTTPRequestHandler(client, httpd.gnutella))
            handler.start()

#handle incoming requests
class HTTPRequestHandler(Runnable):

    def __init__(self, client, gnutella):
        self.client = client
        self.gnutella = gnutella
        self.buffer = jarray.zeros(2048, 'b')
        self.client.setTcpNoDelay(1);


    def run(self):
        self.handleClient()
        self.client.close()

    def handleClient(self):
        self.is = BufferedInputStream(self.client.getInputStream())
        self.ps = PrintStream(self.client.getOutputStream())

        b = self.is.read()

        count = 0
        while b>-1:
            if chr(b) == '\n' or chr(b) == '\r':
                break
            else:
```

```
        self.buffer[count] = b
        count = count + 1
    b = self.is.read()

isGet = 0

print self.buffer[0:4]

if self.buffer[0] == ord('G') and \
    self.buffer[1] == ord('E') and \
    self.buffer[2] == ord('T') and \
    self.buffer[3] == ord(' '):
        isGet = 1
        count = 4
elif self.buffer[0] == ord('H') and \
    self.buffer[1] == ord('E') and \
    self.buffer[2] == ord('A') and \
    self.buffer[4] == ord('D') and \
    self.buffer[5] == ord(' '):
        isGet = 0
        count = 5
else:
    self.ps.print("HTTP/1.0 " + "405" +
                " unsupported method type: ")
    self.ps.write(self.buffer, 0, 5)
    self.ps.print("\r\n")
    self.ps.flush()
    return

start = count

while count < len(self.buffer):
    if self.buffer[count] == ord(' '):
        break
    else:
        count = count + 1

filename = String(self.buffer, start, count - start)
filename = filename.replace('/', File.separatorChar)
if filename[0] == File.separator:
    filename = filename[1:]
if isGet:
    if filename[0:7] == "search?":
        q = "query="
        t = "time="
        index = filename.find(q) + len(q)
        sBuffer = StringBuffer()
        while index < len(filename):
            if filename[index] == '&':
                break
            else:
                sBuffer.append(filename[index])
            index  = index + 1
        query = sBuffer.toString().replace('+',' ')
        index = filename.find(t)
        time = string.atoi(filename[index+len(t):index+len(t)+1])
        self.doSearch(query, time)
        return
    else:
```

```
                    filepath = File(httpd.root, filename)
                    if filepath.isDirectory():
                        tmp = File(filepath, "index.html")
                        if tmp.exists():
                            filepath = tmp
                        else:
                            self.listDirectory(filepath)
                            return
            self.sendFile(filepath)


    def doSearch(self, query, mins):
        print query,"\t",mins
        s = RankedSearchReceiver()
        print "set up receiver"
        query = cPickle.dumps((0,query))
        print query
        search = self.gnutella.createSearchSession(query, 25, 0, s)
        print "creating search"
        time.sleep(30)
        search.close()
        terms = s.getRanks()
        print terms
        search = self.gnutella.createSearchSession(cPickle.dumps((1,terms)), 25, 1,
s)
        time.sleep(30)
        search.close()
        docs = s.getDocuments()
        self.printHeaders(None, 1)
        for doc in docs:
            self.ps.print("<p><a
href=\""+doc.getURL()+"\">"+doc.getTitle()+"</A><BR>")
            self.ps.print(doc.getDescription() + "<BR>")
            self.ps.print(doc.getURL()+"</p>")


    def listDirectory(self, path):
        found = self.printHeaders(path, 0)
        if found:
            self.ps.println("<html>")
            self.ps.println("<head>")
            self.ps.println("<title>Directory listing for " + httpd.root +
path.getName() +"</title>")
            self.ps.println("</head>")
            self.ps.println("<body>")
            self.ps.println("<A HREF=\"..\">Parent Directory</A><BR>\n")
            list = path.list()
            for i in list:
                f = File(path, i)
                if f.isDirectory:
                    self.ps.println("<A HREF=\""+ i +"/\">" + path.getName() + i
+"/</A><BR>")
                else:
                    self.ps.println("<A HREF=\""+ i +"\">" + path.getName() + i
+"</A><BR");

            self.ps.println("<P><HR><BR><I>" + Date().toString() + "</I>")
            self.ps.println("</body>")
            self.ps.println("</html>")
```

```
        else:
            self.error404(path)


    def sendFile(self, file):
        found = self.printHeaders(file, 0)
        if found:
            self.ps.print("\r\n")
            istream = FileInputStream(file.getAbsolutePath())
            n = istream.read(self.buffer)
            while n>0:
                self.ps.write(self.buffer, 0, n)
                n = istream.read(self.buffer)
        else:
            self.error404(file)


    def printHeaders(self, file, isSearch):
        ok = 0
        errCode = 0
        if isSearch or file.exists():
            rCode = 200
            self.ps.print("HTTP/1.0 200 OK")
            self.ps.print("\r\n")
            ok = 1
        elif not file.exists():
            errCode = 404
            self.ps.print("HTTP/1.0 404 not found")
            self.ps.print("\r\n")
            ok = 0
        self.ps.print("Server: zoekdienst web server")
        self.ps.print("\r\n")
        self.ps.print("Date: " +  Date().toString())
        self.ps.print("\r\n")
        return ok

    def error404(self, file):
        self.ps.print("\r\n")
        self.ps.print("\r\n")
        self.ps.println("<html>")
        self.ps.print("<head>")
        self.ps.print("<title>404 - Page Not Found</title>")
        self.ps.print("</head>")
        self.ps.print("<body>")
        self.ps.print("<h1>Error 404 - Page Not Found</h1>")
        self.ps.print("<p>The requested resource <code>" + file.getName() + " was
not found on this server.</p>")
        self.ps.print("</body>")
        self.ps.print("</html>")

# this class handle messages replies
class RankedSearchReceiver(MessageReceiverAdapter):

    def __init__(self):
        MessageReceiverAdapter.__init__(self)
        self.ranks = {}
        self.documents = {}

    def getRanks(self):
```

```
        return self.ranks

    def getDocuments(self):
        self.docs = self.documents.keys()
        self.docs.sort(lambda x,y: -cmp(x.getRank(), y.getRank()))
        return self.docs

    def receiveSearchReply(self, searchReply):
        for i in range(message.getFileCount()):
            fileRecord = searchReply.getFileRecord(i)
            type,  value = cPickle.loads(filerecord.getName())
            if type == 1:
                for term, ft in value:
                    try:
                        self.ranks[term] = self.ranks[term] + ft
                    except KeyError:
                        self.ranks[term]  = ft
            elif type == 3:
                for doc in value:
                    self.documents[doc] = None
# server exception
class HTTPdException(Exception):

    def __init__(self, value):
        self.value = value

    def __str__(self):
        return self.value
```

inversion.py file

```
import sys

class Indexer:

    def __init__(self):

        self.inversionStruct = {}
        self.inversionTemp = {}
        self.documentStruct = {}
        self.documentTemp = {}
        self.lexiconStruct = {}
        self.lexiconTemp = {}

    def index(self, cacheDir):

        import cache
        import parser
        import document
        import transformation
        import string
        import math

        # counters
        documentID = -1
        termID = -1
        currentTermID = -1
        # multi-type parser
        parse = parser.ParserProxy()
```

```
        # transformation manager
        transformationManager = transformation.TransformationManager(1,1,0)
        # cache
        docs = cache.MozillaCache(cacheDir)
        # for all the cached documents, index them - inversion procesa
        for doc in docs:
            try:
                parse.reset()
                # parse and transform document
                pd = parse.parse(doc)
                if not pd:
                    #print "none document"
                    continue
                else:
                    idoc = transformationManager.runIndexing(pd)
                # create document datbase entry
                documentID = documentID + 1
                self.documentTemp[str(documentID)] = [idoc.getURL(),
idoc.getTitle(),idoc.getDescription(),idoc.getDate(),idoc.getContentType(),idoc.get
LocalFilename()]

                # get the document terms
                documentTerms = idoc.getData()
                # for each term in the document
                for term in documentTerms:

                    fdt = documentTerms.count(term)

                    if fdt > 0:

                        try:
                            self.lexiconTemp[term] = self.lexiconTemp[term] + fdt
                        except KeyError:
                            self.lexiconTemp[term] = fdt

                        try:
                            self.inversionTemp[term].append((documentID, fdt))
                        except KeyError:
                            self.inversionTemp[term] = [(documentID,fdt)]
                    # deleting duplicates from list problem
                    documentTerms = [element for element in documentTerms if
element !=term]
                print idoc.getURL(), " indexed"
            except:
                continue

        self.accumulators = {}

        # create document weights
        for invList in values(self.inversionTemp):
            for doc, fdt in invList:
                if not self.accumulators.has_key(doc):
                    self.accumulators[doc] = 0
                wdt = 1+ math.log(fdt)
                self.accumulators[doc] = self.accumulators[doc] + wdt**2

        for docid in self.documentTemp.keys():
            try:
                c = math.sqrt(self.accumulators[string.atoi(docid)])
```

```
            except KeyError:
                c = 0
            self.documentTemp[docid].append(c)
            self.documentStruct[docid] = self.documentTemp[docid]

        for lexid in self.lexiconTemp.keys():
            self.lexiconStruct[lexid] = self.lexiconTemp[lexid]

        for invid in self.inversionTemp.keys():
            self.inversionStruct[invid] = self.inversionTemp[invid]

        # write document entries to disk

        #self.__close()
        print documentID+1, "indexed"

class PickleCompressedIndexer(Indexer):

    def __init__(self):
        Indexer.__init__(self)

    def close(self):
        import zlib
        import cPickle

        inversion = open("cpInversion.db", "wb")
        inversion.write(zlib.compress(cPickle.dumps(self.inversionStruct, 1)))
        inversion.close()

        document = open("cpDocument.db", "wb")
        document.write(zlib.compress(cPickle.dumps(self.documentStruct, 1)))
        document.close()

        lexicon = open("cpLexicon.db", "wb")
        lexicon.write(zlib.compress(cPickle.dumps(self.lexiconStruct, 1)))
        lexicon.close()

class ShelveIndexer(Indexer):

    def __init__(self):
        # remove the files if they exist
        import gdbmShelve
        Indexer.__init__(self)
        self.inversionStruct = gdbmShelve.open("inversion.db", "c")
        self.documentStruct = gdbmShelve.open("document.db", "c")
        self.lexiconStruct = gdbmShelve.open("lexicon.db", "c")

    def close(self):
        self.inversionStruct.close()
        self.documentStruct.close()
        self.lexiconStruct.close()

def values(db):
    try:
        return db.values()
    except AttributeError:
        values = []
        for key in db.keys():
            values.append(db[key])
```

```
        return values

if __name__ == "__main__":
    i = ShelveIndexer()
    i.index(sys.argv[1])
    i.close()
```

jgdbm.py file

```
import jdbm
import java.lang
import org.python.core

error = IOError                         # For anydbm

class Database:

    def __init__(self, file, flag):
        if flag == "c":
            self.database = jdbm.Database(file, jdbm.Database.O_CREATE)
        elif flag == "f":
            self.database = jdbm.Database(file, jdbm.Database.O_FAST)
        elif flag == "n":
            self.database = jdbm.Database(file, jdbm.Database.O_NEWDB)
        elif flag == "r":
            self.database = jdbm.Database(file, jdbm.Database.O_READER)
        elif flag == "w":
            self.database = jdbm.Database(file, jdbm.Database.O_WRITER)
        else:
            raise error("")

    def __getitem__(self, key):
        try:
            val = self.database.fetch(key)
        except jdbm.NoSuchKeyException:
            raise KeyError(key)
        return val

    def __setitem__(self, key, val):
        try:
            self.database.store(key, val, jdbm.Database.S_REPLACE)
        except jdbm.IllegalOperationError:
            raise error("Reader can't store")

    def __delitem__(self, key):
        key = java.lang.String(key)
        try:
            self.database.delete(key.getBytes())
        except jdbm.IllegalOperationError:
            raise error("Reader can't delete")

    def __len__(self):
        return len(self.keys())

    def has_key(self, key):
        return self.database.exists(key)

    def __contains__(self, key):
        return self.database.exists(key)
```

```
    def keys(self):
        keys = []
        currentKey = self.firstkey()
        while currentKey:
            keys.append(currentKey)
            currentKey = self.nextkey(currentKey)
        return keys

    def values(self):
        values = []
        currentKey = self.firstkey()
        while currentKey:
            values.append(self.__getitem__(currentKey))
            currentKey = self.nextkey(currentKey)
        return values

    def firstkey(self):
        try:
            valBytes = self.database.firstkey()
            return org.python.core.PyString(java.lang.String(valBytes))
        except java.lang.NullPointerException:
            return None

    def nextkey(self, key):
        key = java.lang.String(key)
        valBytes = self.database.nextkey(key.getBytes())
        try:
            return org.python.core.PyString(java.lang.String(valBytes))
        except java.lang.NullPointerException:
            return None

    def reorganize(self):
        try:
            self.database.reorganize()
        except jdbm.IllegalOperationError:
            raise error("Reader can't reorganize")

    def close(self):
        self.database.close()

    def __del__(self):
        self.close()

    def sync(self):
        self.database.sync()

class error(Exception):

    def __init__(self, value):
        self.__value = value

    def __str__(self):
        return str(self.__value)

def open(file, flag, mode = None):
    return Database(file, flag)
```

parser.py file

```
from sgmllib import SGMLParser
import document
import string
import robotparser
import urllib
import urlparse

class Parser:

    def __init__(self, verbose=0):
        pass

    def parse(self, cachedDocument):
        pass

    def reset(self):
        pass


class ParserProxy:

    HTML_CONTENT_TYPE = "text/html"

    def __init__(self, verbose=0):
        self.__html = HTMLParser(verbose)

    def parse(self, cachedDocument):
        if cachedDocument.getContentType() == ParserProxy.HTML_CONTENT_TYPE:
            self.__html.reset()
            parsedDocument = self.__html.parse(cachedDocument)
        else:
            raise UnknownContentTypeError(cachedDocument.getContentType())
        return parsedDocument

    def reset(self):
        self.__html.reset()

class HTMLParser(SGMLParser):

    from htmlentitydefs import entitydefs

    def __init__(self, verbose=0):
        SGMLParser.__init__(self, verbose)
        self.__robotParser = robotparser.RobotFileParser()

    def parse(self, cachedDocument):
        self.__cachedDocument = cachedDocument
        split = urlparse.urlparse(cachedDocument.getURL())
        #self.__robotParser.set_url(split[0]+"://"+split[1]+"/robots.txt")
        #self.__robotParser.read()
        #if not self.__robotParser.can_fetch("*",cachedDocument.getURL()):
            #self.__isIndexable = 0
            #print "not indexable"
        try:
            file = open(cachedDocument.getLocalFilename())
            #print "trying open local"
        except(IOError):
            try:
```

```
            file = urllib.urlopen(cachedDocument.getURL())
            #print "trying remote file"
        except(IOError):
            return
    try:
        self.feed(file.read())
    except(sgmllib.SGMLParseError):
        return
    return self.output()

# reset method called when class instantiated so no need for __init__
def reset(self):
    SGMLParser.reset(self)
    self.__cachedDocument = None
    self.__foundTitle = 0
    self.__foundScript = 0
    self.__isIndexable = 1
    self.__title = ""
    self.__text = []
    self.__description = ""

#2 go through and html tags at w3.org and make decisions
def handle_data(self, data):
    data = data.strip()
    if data == '':
        pass
    if self.__foundTitle:
        self.__title = data
        #print "got title"
    # we do not want to index any scripting languages
    elif self.__foundScript:
        pass
        #print "got script"
    else:
        self.__text.append(data)

def start_meta(self, attributes):
    # store description in list
    a,v=attributes[0]
    if a.lower()=="name" and v.lower()=="description":
            a,v = attributes[1]
            if a.lower()=="content":
                    self.__description=v
                    #print "got description"
    # check page can be indexed
    elif a.lower()=="name" and v.lower()=="robot":
            a,v = attributes [1]
            if a.lower() =="content":
                    for word in v:
                            if word.lower() =="noindex":
                                    self.__indexable=0
                                    #print "not indexable"

def end_meta(self):
    pass

def start_title(self, attributes):
    self.__foundTitle = 1
```

```
    def end_title(self):
        self.__foundTitle = 0

    def start_script(self, attributes):
        self.__foundScript = 1

    def end_script(self):
        self.__foundScript = 0

    # this method when called will return details re document
    def output(self):
        if self.__isIndexable:
            return document.ParsedDocument(self.__cachedDocument.getURL(),\

self.__cachedDocument.getDate(),\

self.__cachedDocument.getContentType(),\

self.__cachedDocument.getLocalFilename(),\
                                                        self.__title,\
                                                        self.__description,\

string.join(self.__text, " "))


class Error(Exception):

    def __init__(self, value):
        self.value = value

    def __str__(self):
        return `self.value`

class UnknownContentTypeError(Error):

    pass

class CacheNotFoundError(Error):

    pass
```

query.py file

```
import string
import transformation
import cPickle
import zlib
import gdbmShelve
import document

def intersect(*args):
    res = []
    for x in args[0]:                   # scan first sequence
        for other in args[1:]:          # for all other args
            if x not in other: break    # item in each one?
        else:                           # no:  break out of loop
            res.append(x)               # yes: add items to end
    return res
```

```
def union(*args):
    res = []

    for seq in args:                    # for all args
        for x in seq:                   # for all nodes
            if not x in res:
                res.append(x)           # add new items to result
    return res


class RankedQueryEngine:

    def handleQuery(self, queryFt):
      import math
      results = []
        # 1
        accumulators = {}
        #2 c-f
        for term in queryFt.keys():
            inv = self.inversion[term]
            for d, fdt in inv:
                if not accumulators.has_key(d):
                     accumulators[d] = 0
                accumulators[d] = accumulators[d] + math.log(1+ fdt) *
queryFt[term]
        #3
        for key in accumulators.keys():
            print accumulators[key]
            print self.document[str(key)][6]
            accumulators[key] = accumulators[key] / self.document[str(key)][6]
        #4
        for i in range(50):
            doc,max = 0,0
            for key in accumulators.keys():
                  if accumulators[key] > max:
                        doc = key
                        max = accumulators[key]
            try:
                del(accumulators[doc])
                args = self.document[str(doc)]
                del(doc)

results.append(document.ResultDocument(args[0],args[3],args[4],args[5],args[1],args
[2], max))
            except KeyError:
                pass
        return results

    def getTermFrequencies(self, query):
        #2 a-e
        queryTerms = self.tm.runQuerying(query)
        termFreq = []
        for term in queryTerms:
            termFreq.append((term, self.lexicon[term]))
        return termFreq

class ConjuctiveQuery:

    def handleQuery(self, query):
```

```
        candidates = []
        results = []
        queryFreq = []
        queryTerms = self.tm.runQuerying(query)
        for term in queryTerms:
            try:
                queryFreq.append((self.lexicon[term], term))
            except KeyError:
                return []
        queryFreq.sort()
        print "qf\t", queryFreq
        try:
            candidates = self.inversion[queryFreq[0][1]]
        except IndexError:
            return []
        print "c\t", candidates
        queryFreq = queryFreq[1:]
        for i in queryFreq:
            print "iteration"
            invEnt = self.inversion[i[1]]
            print "ie\t", invEnt
            #intersection of invEnt and self.candidates
            candidates = [(x,y) for x,y in candidates for v,z in invEnt if x==v]
            #if self.canddidates is empty, no answer matching query
            if not len(candidates):
                return []
        for docid, fdt in candidates:
            print docid
            doc = self.document[str(docid)]
            results.append(document.ResultDocument(doc[0], doc[3], doc[4], doc[5],
doc[1], doc[2]))
        return results


class DisjunctionQuery:

    def handleQuery(self, query):
        queryTerms = self.tm.runQuerying(query)
        docids = []
        results = []
        print queryTerms
        for term in queryTerms:
            try:
                for element in self.inversion[term]:
                    docids.append(element[0])
            except KeyError:
                continue
        print "1:\t", docids
        u = {}
        for x in docids:
            u[x] = None
        docids = u.keys()
        docids.sort()
        print u
        print "2:\t", docids
        for docid in docids:
            doc = self.document[str(docid)]
            results.append(document.ResultDocument(doc[0], doc[3], doc[4], doc[5],
doc[1], doc[2]))
```

```
        return results


class CPickleRankedQuery(RankedQueryEngine):

    def open(self):
        self.lexicon = cPickle.loads(zlib.decompress(open("cpLexicon.db",
"rb").read()))
        self.document = cPickle.loads(zlib.decompress(open("cpDocument.db",
"rb").read()))
        self.inversion = cPickle.loads(zlib.decompress(open("cpInversion.db",
"rb").read()))
        self.tm = transformation.TransformationManager(1,1,0)

    def close(self):
        pass

class ShelveRankedQuery(RankedQueryEngine):

    def open(self):
        self.lexicon = gdbmShelve.open("lexicon.db","r")
        self.document = gdbmShelve.open("document.db", "r")
        self.inversion = gdbmShelve.open("inversion.db", "r")
        self.tm = transformation.TransformationManager(1,1,0)

    def close(self):
        self.lexicon.close()
        self.document.close()
        self.inversion.close()

class CPickleConjuctiveQuery(ConjuctiveQuery):

    def open(self):
        self.lexicon = cPickle.loads(zlib.decompress(open("cpLexicon.db",
"rb").read()))
        self.document = cPickle.loads(zlib.decompress(open("cpDocument.db",
"rb").read()))
        self.inversion = cPickle.loads(zlib.decompress(open("cpInversion.db",
"rb").read()))
        self.tm = transformation.TransformationManager(1,1,0)

    def close(self):
        pass

class ShelveConjuctiveQuery(ConjuctiveQuery):

    def open(self):
        self.lexicon = gdbmShelve.open("lexicon.db","r")
        self.document = gdbmShelve.open("document.db", "r")
        self.inversion = gdbmShelve.open("inversion.db", "r")
        self.tm = transformation.TransformationManager(1,1,0)

    def close(self):
        self.lexicon.close()
        self.document.close()
        self.inversion.close()

class CPickleDisjunctionQuery(DisjunctionQuery):
```

```
    def open(self):
        self.lexicon = cPickle.loads(zlib.decompress(open("cpLexicon.db",
"rb").read()))
        self.document = cPickle.loads(zlib.decompress(open("cpDocument.db",
"rb").read()))
        self.inversion = cPickle.loads(zlib.decompress(open("cpInversion.db",
"rb").read()))
        self.tm = transformation.TransformationManager(1,1,0)

    def close(self):
        pass

class ShelveDisjunctionQuery(ConjuctiveQuery):

    def open(self):
        self.lexicon = gdbmShelve.open("lexicon.db","r")
        self.document = gdbmShelve.open("document.db", "r")
        self.inversion = gdbmShelve.open("inversion.db", "r")
        self.tm = transformation.TransformationManager(1,1,0)

    def close(self):
        self.lexicon.close()
        self.document.close()
        self.inversion.close()
```

application.py file

```
from java.awt import *
from java.awt.event import *
from javax.swing import *
from javax.swing.event import *
from javax.swing.border import *
from javax.swing.table import *
from java.util import *
#from java.util.prefs import *
from com.kenmccrary.jtella import *
from java.io import *
import jarray
import java.net
import java.lang
import string
import httpd
import cPickle
import query

class Application(JFrame):

    def __init__(self,ing,outg,port):
        print "start"
        JFrame.__init__(self, "zoekdienst - Indexing and Network Manager")
        print "varaible setup"

        browserEXEVal = "/opt/SUNWns6/netscape"
        browserCacheDirVal =
"/tsunx/home/ug/cs99/stephenn/.mozilla/stephenn/m23c8bj0.slt/Cache/"
        pythonEXEVal = "/tsunx/scratch/stephenn/bin/python"
        no0f0utgoingGnutVal = 5
        allowIncomingConnectionsVal = 1
        incomingPortVal = 6346
```

```
        incomingConnectionsVal = 3
        self.webServerPortVal = 8080
        allowWebServerInVal = 1
        autoConnectVal = 1

        print "done vars"

        # Tabbed Panel
        self.tabs = JTabbedPane()
        self.getContentPane().add(self.tabs)
        print ing,"\t",outg,"\t",port
        # set up as a gnutella node
        searchReceiver = Application.SearchReceiver()
        cd = ConnectionData()
        cd.setIncommingConnectionCount(string.atoi(ing))
        cd.setOutgoingConnectionCount(string.atoi(outg))
        cd.setIncomingPort(string.atoi(port))
        cd.setVendorCode("zoek")
        self.gnutella = GNUTellaConnection(cd)
        session = self.gnutella.createFileServerSession(searchReceiver)
        searchReceiver.setFileServerSession(session)
        self.gnutella.start()

        # start http server
        http = httpd.httpd(".", self.webServerPortVal, 0 , self.gnutella)
        java.lang.Thread(http).start()

        #Panels for each tab
        self.network = JPanel()
        self.index = JPanel()
        self.search = JPanel()
        self.prefs = JPanel()

        # create tabs
        self.tabs.addTab("Network", self.network);
        self.tabs.addTab("Indexing", self.index);
        self.tabs.addTab("Search", self.search);
        self.tabs.addTab("Preferences", self.prefs);

        # menu bar
        menuBar = JMenuBar()
        fileMenu = JMenu("File")
        navMenu = JMenu("Navigation")
        helpMenu = JMenu("Help")
        self.connectMenuItem = JMenuItem("Connect to Gnutella", actionPerformed =
self.gnutellaStart)
        self.disconnectMenuItem = JMenuItem("Disconnect from Gnutella",
actionPerformed=self.gnutellaStop)
        savePrefsMenuItem = JMenuItem("Save Preferences", actionPerformed =
self.savePrefs)
        exitMenuItem = JMenuItem("Exit", actionPerformed = self.windowClosed)
        networkMenuItem = JMenuItem("Network",
actionPerformed=self.navigateNetworkTab)
        indexMenuItem = JMenuItem("Indexing",
actionPerformed=self.navigateIndexingTab)
        searchMenuItem = JMenuItem("Search",
actionPerformed=self.navigateSearchTab)
        prefsMenuItem = JMenuItem("Preferences",
actionPerformed=self.navigatePrefsTab)
```

```
        aboutMenuItem = JMenuItem("About Zoekdienst")
        onlineHelpMenuItem = JMenuItem("Online Help")
        self.setJMenuBar(menuBar)
        menuBar.add(fileMenu)
        menuBar.add(navMenu)
        menuBar.add(Box.createHorizontalGlue())
        menuBar.add(helpMenu)
        fileMenu.add(self.connectMenuItem)
        fileMenu.add(self.disconnectMenuItem)
        self.disconnectMenuItem.setEnabled(0)
        fileMenu.addSeparator()
        fileMenu.add(savePrefsMenuItem)
        fileMenu.addSeparator()
        fileMenu.add(exitMenuItem)
        navMenu.add(networkMenuItem)
        navMenu.add(indexMenuItem)
        navMenu.add(searchMenuItem)
        navMenu.add(prefsMenuItem)
        helpMenu.add(onlineHelpMenuItem)
        helpMenu.addSeparator()
        helpMenu.add(aboutMenuItem)

        # network panel view components
        connectPanel = JPanel()
        addPanel = JPanel()
        routeCachePanel = JPanel()
        routersPanel = JPanel()
        routersPanelSub = JPanel()
        cachePanel = JPanel()
        cachePanelSub = JPanel()
        borderNetwork = TitledBorder("Network")
        borderConn = TitledBorder("Connections")
        borderAdd = TitledBorder("Add Connection")
        borderRouters = TitledBorder("Routers")
        borderCache = TitledBorder("Host Cache")
        connections = JTable(self.gnutella.getConnections())
        self.connectionsScroll = JScrollPane(connections)
        self.connection = JTextField("Enter
<host>:<port>",25,actionPerformed=self.gnutellaAdd)
        addConn = JButton("Add", actionPerformed=self.gnutellaAdd)
        routers = JList()
        routersScroll = JScrollPane(routers)
        connRouter = JButton("Connect")
        addRouter = JButton("Add")
        remRouter = JButton("Remove")
        hostcache = JList()
        hostCacheScroll = JScrollPane(hostcache)
        connectConn = JButton("Connect")
        clearCache = JButton("Clear")

        # create models
        hostmodel = self.gnutella.getHostCache()
        hostcache.setModel(hostmodel)
        routers.setModel(hostmodel)

        # network panel layout
        self.network.setLayout(BoxLayout(self.network, BoxLayout.Y_AXIS))
        self.network.setBorder(borderNetwork)
        self.network.add(connectPanel)
```

```
self.network.add(Box.createRigidArea(Dimension(10, 0)))
self.network.add(addPanel)
self.network.add(Box.createRigidArea(Dimension(10, 0)))
self.network.add(routeCachePanel)

# connect panel layout
connectPanel.setBorder(borderConn)
connections.setPreferredScrollableViewportSize(Dimension(500, 70))
connectPanel.add(self.connectionsScroll)

# add connection layout
addPanel.setLayout(FlowLayout())
addPanel.setBorder(borderAdd)
addPanel.add(self.connection)
addPanel.add(addConn)

# sub panel
routeCachePanel.setLayout(BoxLayout(routeCachePanel, BoxLayout.X_AXIS))
routeCachePanel.add(routersPanel)
routeCachePanel.add(Box.createRigidArea(Dimension(10, 0)))
routeCachePanel.add(cachePanel)

# routers panel
routersPanel.setBorder(borderRouters)
routersPanel.setLayout(BoxLayout(routersPanel, BoxLayout.Y_AXIS))
routersPanel.add(routersScroll)
routersPanel.add(Box.createRigidArea(Dimension(0, 10)))
routersPanel.add(routersPanelSub)
routersPanelSub.setLayout(BoxLayout(routersPanelSub, BoxLayout.X_AXIS))
routersPanelSub.add(connRouter)
routersPanelSub.add(Box.createRigidArea(Dimension(10, 0)))
routersPanelSub.add(addRouter)
routersPanelSub.add(Box.createRigidArea(Dimension(10, 0)))
routersPanelSub.add(remRouter)

# host cache panel
cachePanel.setBorder(borderCache)
cachePanel.setLayout(BoxLayout(cachePanel, BoxLayout.Y_AXIS))
cachePanel.add(hostCacheScroll)
cachePanel.add(Box.createRigidArea(Dimension(0, 10)))
cachePanel.add(cachePanelSub)
cachePanelSub.setLayout(BoxLayout(cachePanelSub, BoxLayout.X_AXIS))
cachePanelSub.add(connectConn)
cachePanelSub.add(Box.createRigidArea(Dimension(10, 0)))
cachePanelSub.add(clearCache)

# Indexing Panel
index1 = JPanel()
index2 = JPanel()
index3 = JPanel()
indexBorder = TitledBorder("Indexing")
schedulerBorder = TitledBorder("Scheduler")
schedulerLabel = JLabel("Scheduler On/Off")
everyL = JLabel("Every")
hourL = JLabel("Hour")
hrs = Vector()
for i in range(24):
    hrs.add(i+1)
self.hour = JComboBox(hrs)
```

```
allowScheduler = JCheckBox(actionPerformed = self.schedule)
indexButton = JButton("Index", actionPerformed=self.runIndexer)
self.feedback = JTextArea()
scrollFeedback = JScrollPane(self.feedback)

# Indexing View
self.index.setBorder(indexBorder)
self.index.setLayout(BoxLayout(self.index, BoxLayout.Y_AXIS))
self.index.add(index1)
self.index.add(index2)

index1.setBorder(schedulerBorder)
index1.add(schedulerLabel)
index1.add(allowScheduler)
index1.add(everyL)
index1.add(self.hour)
index1.add(hourL)

index2.setLayout(BorderLayout())
index2.add(scrollFeedback, BorderLayout.CENTER)
index2.add(indexButton, BorderLayout.SOUTH)

self.feedback.setBorder(TitledBorder("Indexer Feedback"))
self.feedback.setEditable(0)

# indexing listeners


# searching component
info = JTextArea("To search using zoekdienst simply click the button\n"+\
                 "below or enter the url, http://localhost:6348/\n"+\
                 "into the web browser of your choice.")

info.setBackground(everyL.getBackground())
info.setEditable(0)

searchButton = JButton("Search",actionPerformed=self.launchBrowser)

#searching view
self.search.setLayout(BorderLayout())
self.search.setBorder(TitledBorder("Search Zoekdienst"))
self.search.add(info,BorderLayout.CENTER)
#search.add(Box.createRigidArea(Dimension(10, 0)))
self.search.add(searchButton, BorderLayout.SOUTH)

# prefrences components
prefsBorder = TitledBorder("Preferences")
browserBorder = TitledBorder("Browser Settings")
gnutBorder = TitledBorder("Gnutella Settings")
httpBorder = TitledBorder("Web Server Settings")
pythBorder = TitledBorder("Python Settings")
prefs1 = JPanel()
prefs2 = JPanel()
prefs3 = JPanel()
prefs4 = JPanel()

browserPrefL = JLabel("Browser Preferences Folder")
self.browserPref = JTextField(browserCacheDirVal, 25)
browserL = JLabel("Path to Browser Executable")
```

```
self.browserPath = JTextField(browserEXEVal, 25)
self.browserButton = JButton("Browse", actionPerformed=self.fileChooser)
self.browserButton2 = JButton("Browse", actionPerformed=self.fileChooser)

autoConnectL = JLabel("Auto-Connect to Gnutella Network")
outgoingMaxL = JLabel("Number of Outgoing Connections")
incomingAllowedL = JLabel("Allow Incoming Connections")
incomingPortL = JLabel("Incoming Connection Port")
incomingMaxL = JLabel("Number of Incoming Connections")
self.autoConnect = JCheckBox()
self.outgoingMax = JTextField(str(no0f0utgoingGnutVal), 5)
self.incomingAllowedYes = JRadioButton("Yes")
self.incomingAllowedNo = JRadioButton("No")
incomingAllowedGroup = ButtonGroup()
incomingAllowedGroup.add(self.incomingAllowedYes)
incomingAllowedGroup.add(self.incomingAllowedNo)
self.incomingPort = JTextField(str(incomingPortVal), 5)
self.incomingMax = JTextField(str(incomingConnectionsVal),5)

serverPortL = JLabel("Web Server Port")
self.serverPort = JTextField(str(self.webServerPortVal), 5)
allowNonLocalL = JLabel("Allow Non-Local Connections to Web Server")
allowNonLocalGroup = ButtonGroup()
self.allowNonLocalYes = JRadioButton("Yes")
self.allowNonLocalNo = JRadioButton("No")
allowNonLocalGroup.add(self.allowNonLocalYes)
allowNonLocalGroup.add(self.allowNonLocalNo)

self.pyPath = JTextField(pythonEXEVal, 25)
pyPathL = JLabel("Python Binary/Exe Location")
self.pyPathBrowse = JButton("Browse", actionPerformed=self.fileChooser)

self.prefs.setLayout(BoxLayout(self.prefs, BoxLayout.Y_AXIS))
self.prefs.setBorder(prefsBorder)

prefs1.setBorder(browserBorder)
prefs1.setLayout(GridLayout(2,3))
prefs1sub1 = JPanel()
prefs1sub2 = JPanel()
prefs1.add(prefs1sub1)
prefs1.add(prefs1sub2)
prefs1sub1.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
prefs1sub1.add(browserL)
prefs1sub1.add(self.browserPath)
prefs1sub1.add(self.browserButton)
prefs1sub2.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
prefs1sub2.add(browserPrefL)
prefs1sub2.add(self.browserPref)
prefs1sub2.add(self.browserButton2)

prefs2.setBorder(gnutBorder)
prefs2.setLayout(GridLayout(3,2))
prefs2sub1 = JPanel()
prefs2sub1.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
prefs2sub1.add(autoConnectL)
prefs2sub1.add(self.autoConnect)
prefs2.add(prefs2sub1)
prefs2sub2 = JPanel()
prefs2sub2.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
```

```
        prefs2.add(prefs2sub2)
        prefs2sub2.add(outgoingMaxL)
        prefs2sub2.add(self.outgoingMax)
        prefs2sub3 = JPanel()
        prefs2sub3.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
        prefs2.add(prefs2sub3)
        prefs2sub3.add(incomingAllowedL)
        prefs2sub3.add(self.incomingAllowedYes)
        self.incomingAllowedYes.setSelected(allowIncomingConnectionsVal)
        self.incomingAllowedNo.setSelected(not allowIncomingConnectionsVal)
        prefs2sub3.add(self.incomingAllowedNo)
        prefs2sub4 = JPanel()
        prefs2sub4.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
        prefs2.add(prefs2sub4)
        prefs2sub4.add(incomingPortL)
        prefs2sub4.add(self.incomingPort)
        prefs2sub5 = JPanel()
        prefs2sub5.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
        prefs2.add(prefs2sub5)
        prefs2sub5.add(incomingMaxL)
        prefs2sub5.add(self.incomingMax)

        prefs3.setBorder(httpBorder)
        prefs3.setLayout(FlowLayout(FlowLayout.LEFT))
        prefs3sub1 = JPanel()
        prefs3.add(prefs3sub1)
        prefs3sub1.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
        prefs3sub1.add(serverPortL)
        prefs3sub1.add(self.serverPort)
        prefs3sub1.add(allowNonLocalL)
        prefs3sub1.add(self.allowNonLocalYes)
        self.allowNonLocalYes.setSelected(allowWebServerInVal)
        self.allowNonLocalNo.setSelected(not allowWebServerInVal)
        prefs3sub1.add(self.allowNonLocalNo)

        prefs4.setBorder(pythBorder)
        prefs4.setLayout(GridLayout(1,1))
        prefs4sub1 = JPanel()
        prefs4.add(prefs4sub1)
        prefs4sub1.setLayout(FlowLayout(FlowLayout.LEFT,10,10))
        prefs4sub1.add(pyPathL)
        prefs4sub1.add(self.pyPath)
        prefs4sub1.add(self.pyPathBrowse)

        self.prefs.add(prefs1)
        self.prefs.add(prefs2)
        self.prefs.add(prefs3)
        self.prefs.add(prefs4)


connections.getModel().addTableModelListener(Application.ConnectionModelListener(co
nnections))

    def gnutellaStart(self, event):
        self.gnutella = GNUTellaConnection()
        self.gnutella.start()
        self.connectMenuItem.setEnabled(0)
        self.disconnectMenuItem.setEnabled(1)
```

```
    def gnutellaStop(self, event):
        self.gnutella.stop()
        self.connectMenuItem.setEnabled(1)
        self.disconnectMenuItem.setEnabled(0)

    def gnutellaAdd(self, event):
        hostPort = self.connection.getText()
        n = hostPort.find(":")
        self.gnutella.addConnection(hostPort[0:n],string.atoi(hostPort[n+1:]))

    def savePrefs(self, event):
        #self.preferences.put("BROWSER_EXE", self.browserPath.getText())
        #self.preferences.put("BROWSER_PREF", self.browserPref.getText())
        #self.preferences.put("PYTHON_EXE", self.pyPath.getText())
        #self.preferences.putInt("OUTGOING_GNUTELLA_CONNECTIONS",
string.atoi(self.outgoingMax.getText()))
        #self.preferences.putBoolean("INCOMING_GNUTELLA_CONNECTIONS_ALLOWED",
self.incomingAllowedYes.isSelected())
        #self.preferences.putInt("INCOMING_GNUTELLA_PORT",
string.atoi(self.incomingPort.getText()))
        #self.preferences.putInt("NUMBER_INCOMING_CONNECTIONS",
string.atoi(self.incomingMax.getText()))
        #self.preferences.putInt("WEB_SERVER_PORT",
string.atoi(self.serverPort.getText()))
        #self.preferences.putBoolean("ALLOW_WEB_SERVER_INCOMING",
self.allowNonLocalYes.isSelected())
        #self.preferences.putBoolean("AUTO_CONNECT", self.autoConnect.isSelected())
        #self.preferences.flush()
        JOptionPane.showMessageDialog(self, "Preferences Saved", "Preferences
Saved", JOptionPane.INFORMATION_MESSAGE);

    def navigateNetworkTab(self, event):
        self.tabs.setSelectedComponent(self.network)

    def navigateIndexingTab(self, event):
        self.tabs.setSelectedComponent(self.index)

    def navigateSearchTab(self, event):
        self.tabs.setSelectedComponent(self.search)

    def navigatePrefsTab(self, event):
        self.tabs.setSelectedComponent(self.prefs)

    def runIndexer(self, event):
        runtime = java.lang.Runtime.getRuntime()
        print "before process"
        process = runtime.exec("/tsunx/scratch/stephenn/bin/python inversion.py
/tsunx/home/ug/cs99/stephenn/.mozilla/stephenn/m23c8bj0.slt/Cache/")
        print "after process"
        inputStream = BufferedReader(InputStreamReader(process.getInputStream()))
        line = inputStream.readLine()
        while line is not None:
            self.feedback.append(line+"\n")
            line = inputStream.readLine()
            print "sending stuff out"

    def schedule(self, event):
        if event.getSource().isSelected():
            self.timer = Timer()
```

```
            time = self.hour.getSelectedItem()*6000#00

self.timer.scheduleAtFixedRate(Application.ScheduledIndexer(self.feedback), time,
time)
        else:
            self.timer.cancel()

    def fileChooser(self, event):
        chooser = JFileChooser()
        chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES)
        returnVal = chooser.showOpenDialog(self)
        if returnVal == JFileChooser.APPROVE_OPTION:
            if event.getSource() is self.pyPathBrowse:
                self.pyPath.setText(chooser.getSelectedFile().getPath())
            elif event.getSource() is self.browserButton:
                self.browserPath.setText(chooser.getSelectedFile().getPath())
            elif event.getSource() is self.browserButton2:
                self.browserPref.setText(chooser.getSelectedFile().getPath())

    def launchBrowser(self, event):
        runtime = java.lang.Runtime.getRuntime()
        runtime.exec(self.browserPath.getText() + " http://localhost:" +
str(self.webServerPortVal))

    def windowClosed(self, event):
        returnVal = JOptionPane.showConfirmDialog(self,\
            "Are you sure you want to exit?", "Confirm Exit",
JOptionPane.YES_NO_OPTION)
        if returnVal ==  JOptionPane.YES_OPTION:
            returnVal = JOptionPane.showConfirmDialog(self, \
                "Would you like to save your preferences?", "Save Preferences", \
                JOptionPane.YES_NO_OPTION)
            if returnVal == JOptionPane.YES_OPTION:
                self.savePrefs(None)
            java.lang.System.exit(0)

    class ConnectionModelListener(TableModelListener):

        def __init__(self, table):
            self.table = table

        def tableChanged(self, event):
            self.table.revalidate()
            self.table.repaint()

    class ScheduledIndexer(TimerTask):

        def __init__(self, textArea):
            self.textArea = textArea

        def run(self):
            runtime = java.lang.Runtime.getRuntime()
            print "before process"
            process = runtime.exec("/tsunx/scratch/stephenn/bin/python inversion.py
/tsunx/home/ug/cs99/stephenn/.mozilla/stephenn/m23c8bj0.slt/Cache/")
            print "after process"
            inputStream =
BufferedReader(InputStreamReader(process.getInputStream()))
            line = inputStream.readLine()
```

```
            while line is not None:
                self.textArea.append(line+"\n")
                line = inputStream.readLine()
                print "out"


    class SearchReceiver(MessageReceiverAdapter):

        def __init__(self):
            self.queryEngine = query.ShelveRankedQuery()

        def setFileServerSession(self, fileServerSession):
            self.fileServerSession = fileServerSession

        def receiveSearch(self, searchMessage):
            #print "received"
            self.queryEngine.open()
            queryMessage = searchMessage.getSearchCriteria()
            try:
                # zoekdienst search
                query = cPickle.loads(queryMessage)
                #print "Query\t",query
                if query[0] == 0:
                    # node is asking for local term weight
                    r = self.queryEngine.getTermFrequencies(query[1])
                    #print "freq\t",r
                    reply = cPickle.dumps((1,r))
                    #print "reply\t", reply
                    replyMessage =
SearchReplyMessage(searchMessage,6666,"139.184.50.12",DownloadConstants.DOWNLOADSPE
ED_T3, "ZPEK")
                    #print "created message"
                    fileRecord = FileRecord(1,1,reply)
                    #print "file record"
                    replyMessage.addFileRecord(fileRecord)
                    #print "added"
                    self.fileServerSession.queryHit(searchMessage, replyMessage)
                    #print "query 0"
                elif query[0] == 2:
                    # node is providing us with global weights and query
                    reply =
cPickle.dumps((3,self.queryEngine.handleQuery(query[1])))
                    replyMessage =
SearchReplyMessage(searchMessage,6666,"139.184.50.12",DownloadConstants.DOWNLOADSPE
ED_T3, "ZPEK")
                    fileRecord = SearchReplyMessage.FileRecord(1,1,reply)
                    replyMessage.addFileRecord(fileRecord)
                    self.fileServerSession.queryHit(searchMessage, replyMessage)
                    #print "query 1"
            except cPickle.BadPickleGet:
                # normal gnutella file sharing search
                #print "normal"
                pass
            self.queryEngine.close()

def test(arg1,arg2,arg3):
    a = Application(arg1,arg2,arg3)
    a.pack()
    a.setVisible(1)
```

```
    a.setResizable(0)

if __name__ == "__main__":
    import sys
    test(sys.argv[1],sys.argv[2],sys.argv[3])
```