Search Engine Customization and Data Set Builder

Master Thesis developed at
Katholieke Universiteit Leuven, Belgium
evaluated at
Barcelona School of Informatics, Spain

by
Francisco Javier Arias Moreno

under the direction of the professor
Marie-Francine Moens

with the supervision of
Marie-Francine Moens
Christina Lioma

Belgium, 9th of January, 2009

Títol del Projecte:	Search Engine Customization and Data Set Builder
Volum:	1/1
Alumne:	Arias Moreno, Francisco Javier
Director:	Moens, Marie-Francine
Departament:	Department of Computer Science of the Katholieke Universiteit Leuven Belgium
Data:	09/01/09

DADES DEL PROJECTE

Títol del Projecte: Search Engine Customization and Data Set Builder

Nom de l'estudiant: Arias Moreno, Francisco Javier
Titulació: Master of Science in Computer Engineering
Crèdits: 37.5
Director: Moens, Marie-Francine
Department: Computer Science of the Katholieke Universiteit Leuven, Belgium

---

MEMBERS OF THE TRIBUNAL (name and signature)

President: TURMO BORRÁS, JORGE (LSI)

Speaker: RODRÍGUEZ HONTORIA, HORACIO (LSI)

Member: FUERTES ARMENGOL, JOSE MARIA (ESAII)

QUALIFICACIÓ

Numeric qualification:
Descriptive qualification:

Date:

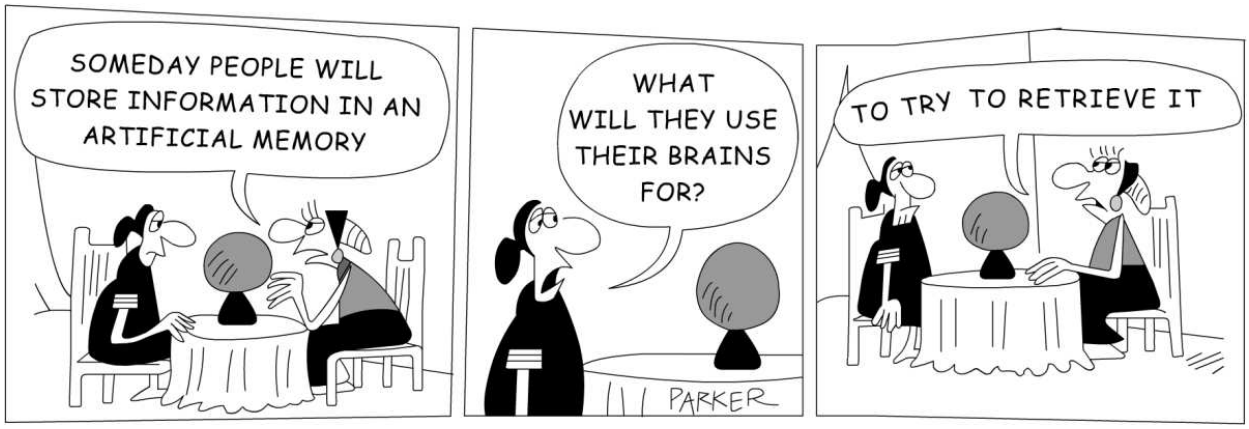

Figure 1: A prediction of the future

Dedicado a mi padre.

Contents

Preface

This project is the reflection of applying the knowledge acquired during my studies and work experience to a typical informatics situation. The project was developed at Katholieke Universiteit Leuven (Belgium) where I was hired as an international scholar. The director of the project is Professor Marie-Francine Moens who holds a M.Sc. and a Ph.D degree in Computer Science from K.U. Leuven. She currently leads the Language Intelligence and Information Retrieval (LIIR) research group. The core of the research conducted by LIIR regards problems of information retrieval, extraction, summarization and further language processing problems in the domains of news, business intelligence, bioinformatics, police and intelligence services, and electronic mail.

Under the supervision of Professor Moens this project is part of the Acknowledge project (Accessible & Open Knowledge Infrastructure for Flanders) which is sponsored by the European Commission, Belgian and Flemish governments, other academic research institutions (the United Nations University - Comparative Regional Integration Studies (UNU-CRIS) and the Interdisciplinary Institute for Broadband Technology (IBBT)). The Acknowledge project aims Flanders to become acquainted with a demonstration model of giving fast, simple and cheap access to an open and accessible knowledge infrastructure. This way, even people with low to no Information and Communications Technology (ICT) skills will have access to high quality personalized learning and knowledge content.

The number of Project credits is 30 ECTS (37,5 FIB credits). Thus, the workload required for completing the project is estimated as 15 weeks working full time (40 hours per week). In my case, I worked full time (40 hours per week) from the 1st of July 2008 until the 19th of December 2008, which means 25 weeks, that is over the 65% of the estimated time.

This project has two main parts: the first part addresses a data set builder; the second part addresses a search engine customization. Regarding the first part, the data set builder is a crawler and cleaner application. This application collects links in a Web page which contains news, for later visiting the links and extracts the main content from the noisy parts of the external Web site (menus, advertisement, etc.). Regarding the second part, a search engine is customized in order to facilitate the retrieval of high quality personalized information and hence the access of people without ICT skills to knowledge. The search engine used for customization is the Lemur Toolkit. Lemur is an open-source toolkit designed to facilitate research in language modeling and information retrieval developed by Carnegie Mellon University and the University of Massachusetts, Amherst.

Chapter 1

Introduction

The research was realised in the frame of the Acknowledge[1] project, where the aim was to crawl, clean, automatically classify and index Web pages. This thesis presents a data set builder and a search engine customization. The data set builder consists of a news crawler, which contains a content extractor, also called a "cleaner".

Data Set Builder: Crawler. A Web crawler (also known as a Web spider, Web robot and scutter) is a program that browses the World Wide Web in a methodical, automated manner. This process is called Web crawling or spidering. Many sites, in particular search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches. A Web crawler is one type of bot, or software agent. In general, it starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies.

Data Set Builder: Content Extractor. When building a system for searching or mining Web content, a first task is extracting the main content and removing extraneous data. Also when showing Web pages on small screens (e.g., of mobile phones) or sending text to screen readers that translate the text to a more appropriate format (e.g., text-to-speech for visually impaired people), the content extraction operation is very valuable. Content extraction (CE) is defined as the process of determining those parts of an HTML document which represent its main textual content [27]. Because different Web pages often have a different layout and a variety of configurations are possible, the task is not trivial. Recently a number of solutions have been proposed. The problem, however, is to find a solution that is generic (i.e., portable to many types of Web pages), accurate (i.e., find all important content in a precise way) and efficient (often a large number of Web pages are processed).

This project designed, implemented and evaluated a content extraction system that satisfies the above requirements. This method is a novel, simple, generic, language-independent, robust and an efficiently computable solution for extracting the main content of an HTML-formatted Web page and for removing additional contents such as navigation menus, functional and design elements, and commercial advertisements. This method creates a text density graph of a given Web page and then selects the region of the Web page with the highest density. The results are comparable or slightly better than state of the art methods

[1] www.ibbt.be/en/project/acknowledge

that are computationally more complex, when evaluated on a standard dataset.

Search Engine Customization. A Web search engine is a tool designed to search for information on a large repository of data, such as the World Wide Web. Information may consist of web pages, images and other types of files. This information is collected with the crawler, cleaned with the content extractor and indexed with the search engine. The search engine used in this project is customized to associate meta-information with each cached document.

The remainder of this thesis is organized as follows: Chapter 2 defines the specific objectives of this work. Chapter 3 presents related research on content extraction. Chapter 4 describes the technical development of the crawler, cleaner and search engine toolkit realized. Chapter 5 presents the experimental evaluation of this technology. Chapter 6 discusses the results of the experimental evaluation with respect to the objectives of this thesis. Chapter 7 presents an economic analysis of this work. Chapter 8 concludes this thesis with an overview of the work realized and its contributions.

Chapter 2

Definition of the objectives

There are two core objectives in this work: firstly, to build a data set, and secondly, to customize a search engine.

The first objective is to design and implement a data set builder. There are two steps required for this. The first step is to build a crawler. The second step is to include a cleaner. The crawler collects Web links. The cleaner extracts the main content and removes noise from the files crawled. The goal of this application is crawling Web news sites to find the different sources of the news and retrieve the original articles.

The second core objective is to customize a search engine. There are two steps required for this. The first step is to enhance the functionalities of the search engine. The second step is to integrate the enhanced search engine with a different knowledge management platform. In order to enhance the search engine, meta-information is added to its index, and the retrieval process is modified so that the selection of documents to be retrieved takes into account this new meta-information. The integration of the search engine to a different knowledge platform is a requirement of this project, so that pre-existing repositories of knowledge can interact with the search engine in an efficient and effective way. This integration also includes the development of a front-end that will allow users to utilize the search engine. The goal of this application is to provide the users with a convenient environment that allows retrieving information depending on meta-information.

To recapitulate, the main goals of this project are:

- **Objective 1.** Data set builder

Step 1 Crawl URLs

Step 2 Extract content (or 'clean') the URLs

- **Objective 2.** Search engine customization

Step 1 Add meta-information to the index and retrieval of documents by the search engine

Step 2 Integrate the search engine to the Acknowledge platform and create a CGI front-end

Figure 2.1 describes an overview of the entire system designed in this project.

The data set builder and search engine customization described in this work are realised in the news domain. In particular, the focus was to crawl, clean and retrieve information

from Google News[1]. Google News searches news related about a topic in a huge number of sources, and classifies the articles related to different topics. Google News classifies as well these topics depending on various categories.

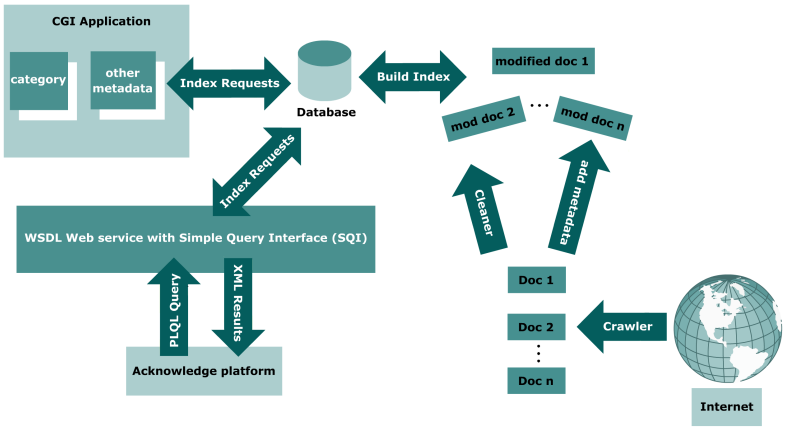

Figure 2.1: An overview of the System

Because Google News is available in different languages, the program for crawling information has to consider the character map of the language that it is retrieving. After crawling the Web, the primary content has to be chosen from the other items (navigation sidebars, advertisements, copyright notices, etc.). Current Web mining techniques approach this goal by assuming that the Web source is well designed or that they can create a generic pattern retrieving different news from the same source. These are really strong assumptions. Because the content has not a predefined location in every Web page, the goal of this work was to develop a generic cleaning algorithm to select the content of the news assuming that the Web page can be badly designed and the content can be located at any point. Creating patterns for retrieving different news from the same source was avoided because that needs some knowledge about the site that is retrieved. The assumption was simple: the largest almost continuous text in the Web page should be the content.

2.1 News crawler

Web Robots (also known as Crawlers, Web Wanderers, or Spiders) are programs that traverse the Web automatically. Search engines use them to index the Web content, spammers use them to scan for email addresses, and they have many other uses.

The first aim of this project is to crawl Web sites and build a collection of URL links that can be later used for retrieving. The Web sites crawled will be cleaned if possible. In addition, meta-information that describes the document will be added. In the last step, this collection of documents will be indexed and used for information retrieval.

Typically, crawlers are mainly used to generate a local copy of the visited pages. Often, crawlers are used with additional cleaners, which remove noisy content from the crawled documents in order to facilitate the indexing process of the search engine. The cached copy of the external Web page is the cleaned text, which does not introduce noise in the index, and which saves space in the hard disk.

[1]http://news.google.com

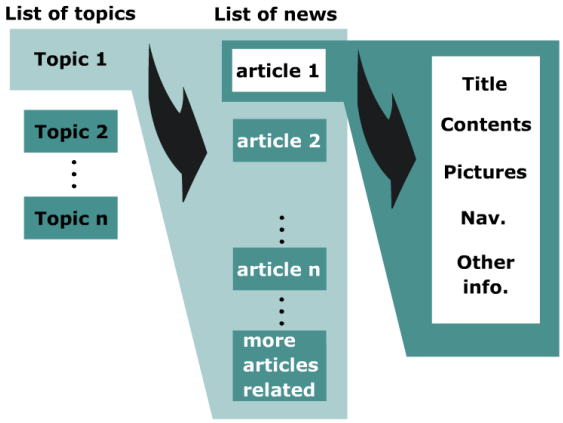

Figure 2.2: An overview of the structure of a news list Web site

Crawling URL links is realized as follows:

- At the source code of the Web page which lists the news, we detect a token (a combination of tag names and attribute values) which identifies the URLs that redirect the browser to the list of articles related to a news item. At the source code of the Web page which lists the articles related to a topic, we detect another token which identifies the URL that redirects the browser to the external link where the news are located.

As it can be appreciated in Figure 2.2, this structure can be handled like a tree structure of three levels, where the root is the category and the nodes of the first level are the different topics that the category lists. The children of the nodes of this first level are the articles related to this concrete topic. Finally, the children of this second level are the nodes which contain the article itself.

The methodolgy to crawl a Web page that lists news is based in the following straightforward algorithm:

1 Create a queue of URLs to be searched beginning with the known list of news passed by parameter to the executable file. The URL is detected using the token that we described above.

2 Pull a URL that contains a news item out of the queue and create a second queue of external URLs to be fetched. These external URLs contain articles from different sources related with the news of the previous queue. If the page contains a URL with more articles related, follow the URL which lists them and continue queuing the URL-articles to the second queue. Continue Step 2 until the URL-news queue gets empty.

3 Pull a URL-article out of the second queue and fetch the HTML page to clean the page from noisy contents. Create a local copy of the cleaned part and attach some meta-information (language, source, document id, etc.). Continue Step 3 until the URL article queue gets empty.

When discussing crawlers, there are two aspects to consider. The first one is the *robot exclusion protocol*, and the second one is the *crawling depth*. These two issues are discussed separately next.

Robot Exclusion Protocol. The robot exclusion standard, also known as the Robots Exclusion Protocol or robots.txt protocol, is a convention to prevent cooperating Web crawlers from accessing all or part of a Website which is otherwise publicly viewable. The standard complements Site maps, a robot inclusion standard for Websites. A robots.txt file on a Website will function as a request that specified robots ignore specified files or directories in their search. If the robot tries to access a Web page that is excluded for robots in the robots.txt file, the server will not show the accessed file.

The robot exclusion standard is both a security measure and a polite measure. This is a polite measure because the user-agent can fake his identity and the server cannot distinguish if it is a fake or not. When a crawler is designed, there is a field to be completed, which is the name of the user-agent that will negotiate the HTTP protocol, that it should be the name of the crawler. The name of the user-agent will be the field that the robots.txt will compare to allow or not the submission of the Web page. If it is necessary to download the Web page, it is possible to set the crawler with the same value as the value that a Web browser will send.

In addition, servers have further measures for denying access. These measures look at the times where the user-agent requests HTML files from an IP address. If a user-agent (that says that is a browser) retrieves often Web pages, the server can determine that it is not a browser. When the Web server determines that it is a robot that says that is a browser, the server can automatically ban this IP address for a period of time (see Figure 2.3 for an example).

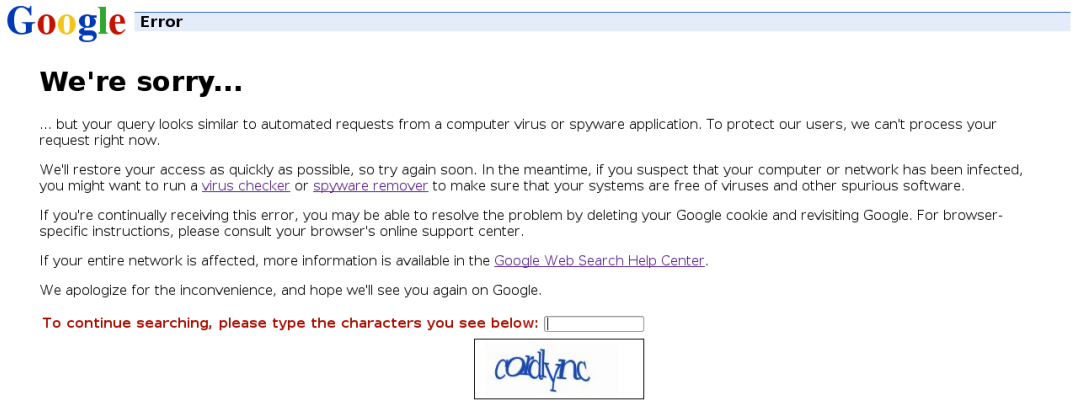

Figure 2.3: An advice that crawlers are not allowed

If one wishes to crawl this information even if it is advised not to, one can do it. The idea to skip this time protection is based on the different ways that the robot can crawl the Web server.

Crawling Depth. From the point of view of a crawler, there is a limit of how deep it crawls. If the levels of depth are considered like the levels of a tree structure, the robot can visit this virtual tree with different algorithms. The two main algorithms in browsing tree structures are: Breadth-first search and Depth-first search. Note that there are further algorithms, for instance Symmetric traversal [32], but in this work we focus on the main ones only. With Depth-first search, if the Web server lists external Web pages in a level higher than for example three, the Web server will not analyze what the robot does when it is in an external URL. It will be more difficult for a server to determine if the robot is a human or

not. With Breath-first search, if the robot uses Breath-first search, the server will detect the application easily even if there are delays in the execution of the code, because the robot will be interacting often with the same server from the same IP address. Figure 2.4 illustrates graphically the Breath-first search and the Depth-first search algorithms.

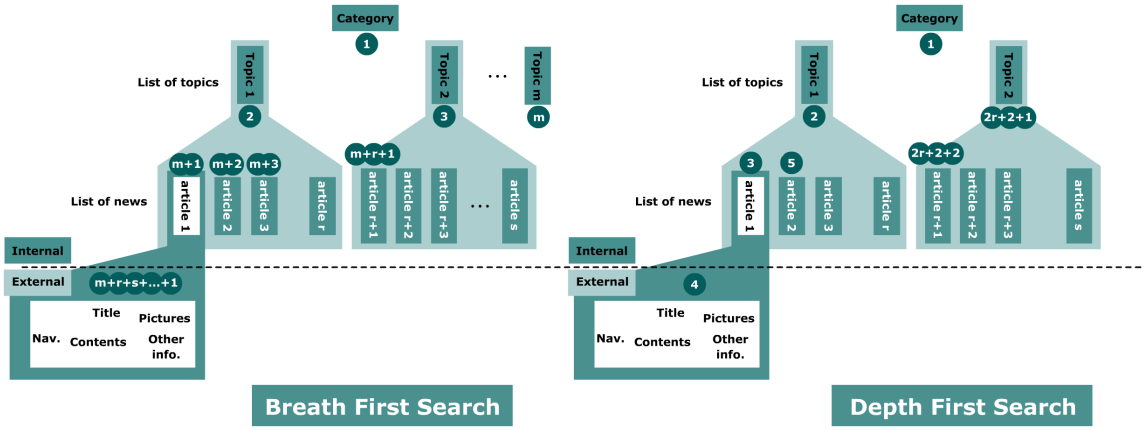

Figure 2.4: Breath-first search and Depth-first search

2.2 Content extraction

After crawling a Web page, the next step is to 'clean' it, or extract content from it. This process looks at the source file of the Web page. An HTML source file contains HTML markup-tags and text. This text consists of some relevant content ("main content") but also of a lot of content that is not relevant outside the context of this particular Web page, such as navigational menus, comments, links to related articles and others. The goal of the content extraction approach presented in this work is to develop a method that classifies every character in the source file as being relevant or not relevant, and creates an output file that contains only the main content, cleaned from any markup-tags.

The first approach that was thought to clean the Web page from the noisy parts, was to remove the non-important parts using a parser, but this is not possible, because all the Web pages are not designed in the same way, and one cannot assume in what part of an unknown design the contents of the news are located. On the other hand, one cannot assume neither that the Webpage is designed respecting the standards of W3C.

The solution that was found after analyzing hundreds of pages was to generalize a method that splits the Web page in an array of strings depending on the location of these strings on the layout of the Webpage. This solution uses the HTML tags that usually define the structure that the page is rendered in order to determine the location of the strings. It would be interesting to determine which tags can give us a semantic meaning, but there is not a relation of semantic values and HTML tags. In some cases, H1 determines the title of the news, H2 the section, and H3 a subsection.

If these HTML tags were to determine the semantic value of text in a consistent way, then they could be used when indexing the Web pages. But these tags define solely how the page is rendered, hence they are of no use to indexing.

Hence, in order to select the main content, the generic methodology followed processes different tags in different ways to determine where the content is located.

Few weak assumptions were made:

- We can ignore the content inside JavaScript tags, Style contents and HTML comments. This content does not provide any information about the news. This content just introduces noise to our analysis.
- If a part of the content of the Webpage is embedded in a function of JavaScript, we will not consider that part of the content, because we cannot guess how it is managed by the function.

The text of the Webpage is split depending on few tags that determine the layout of the Webpage. All non-graphical characters (excluding the white space) are removed from the text. For this reason it is important to set the character map for the text that is retrieved.

The specific methodology adjusts the settings and parameters to improve the results. The tags that often define the layout of the Web page are the following:

- The closing tags (`</table>`, `</div>`, `</dd>`, `</h1>`, `</l1>`, `</tr>`, `</td>`, `</option>`, `</p>` and `<br>`) the tag `<a>` refers to a link. When this tag is found, a position in the array is moved forward, otherwise sometimes the words get concatenated. The rest of the tags are ignored. With this first step an array of strings is obtained.
- The second step is to analyze this array of strings depending on the length of the different positions.

A graphical representation of the array is built, where the x-axis represents the position of the array and the y-axis represents the length of the strings at the different positions. With this array, an assumption can be made. It can be assumed that in a Web page that contains an article, the contents of the article must be the strings that have a longer length and are near each other, which is the interval of the array with highest density. To find a solution, the interval in the x-axis that includes the contents of the article has to be defined.

Notice that the data points do not fall exactly on a straight shape of a function's representation. The fastest approach which finds this interval is an algorithm that looks for the positions with few high values of the array, and choose the interval around these points with the highest density, minding about short intervals with low values between the high density intervals.

Another solution can be to use interpolation and approximation by functions; such solutions can have certain disadvantages, for instance, the algorithmic cost is much higher and the algorithm has to be smart enough to choose the correct approximation (least squares polynomials [35], B splines and bezier splines [5]). On the contrary, the computational cost of our algorithm is much lower because the computation has a linear cost.

2.3 Search engine customization

2.3.1 Information Retrieval Systems and the Lemur Toolkit

'Search Engine' is the name commonly used to describe Information Retrieval (IR) systems. Given a collection of documents and a user with an information need (or query), the task

of an IR system is to retrieve from the collection documents which are relevant to the user query. Typically the retrieved documents are ranked according to their relevance to the query. In order for retrieval to take place, the document collection must be indexed by the information retrieval system. The index is a set of data structures, containing statistics about the occurrence of words (or terms) in the collection. An integral part of the index is the inverted file, which records information on which terms appear in which documents and the term frequency statistics of these terms. Typically, terms are associated with individual weights, which capture the importance of the terms to the content of each document. A matching function then estimates the likely relevance of a document to a query, on the basis of these term weights, and the most relevant documents are identified and retrieved.

In this project, the information retrieval system used was the widely popular and established open-source Lemur Toolkit. The Lemur Toolkit is designed to facilitate research in language modeling and information retrieval.

Lemur is a complete IR platform, with additional functionalities, such a distributed information retrieval, structured queries, automatic synopsis of documents, and content filters. It is a flexible platform that allows to develop own customizations. Lemur has an API to develop our extensions in C++. Lemur is a multi platform toolkit; it works over Windows, UNIX, GNU Linux and MacOSX.

The basic tools from the toolkit are the following:

- BuildIndex: Generates an index from a collection of documents.
- ParseQuery: Processes the queries submitted by users.
- Retrieval: Evaluates the queries and returns a ranking of the documents.

The Lemur toolkit uses the parser antlr and xpdf. This parser is used to handle the files that must be indexed. In the Microsoft Windows version, the Lemur toolkit uses the Microsoft Office libraries to handle the Microsoft Word and Power Point files.

Within this work, Lemur was customized with respect to the following:

- Specify the documents that belong to a collection: the document collection to be indexed was the output of the data set builder described previously (crawled and cleaned news documents).
- Specify the operations that will be done over the documents' text: Meta-information was added to the documents indexed. This meta-information constitutes new retrieval elements that could be specified by the users.

2.3.2 Meta-information

The United Nations University required a solution to retrieve files depending on meta-information. This is one of the reasons why the Lemur Toolkit was chosen. The initial aim was to associate two types of meta-information that describe the documents. These details are the entity which provides the document, and the category of the document.

- **Entity.** The entity which provides the document can be for example East African Community (EAC)[2] or Group of Three (G3)[3]. The categories are obtained from a predefined list. This list of categories can be adapted to specific needs.

[2]Kenya, Tanzania, Uganda
[3]Colombia, Venezuela, Mexico

- **Category.** Part of the Acknowledge project regards the categorization of texts. Subject categories are automatically assigned to a document. For example the classification can be Agricultural and Foreign Policy. For the classification of documents into categories a supervised machine learning technique was used with a focus on finding the best feature selection method. Supervised learning methods train from examples that are manually classified by experts, and learn a classification model (e.g., function) that classifies new, previously unseen examples. Here training and testing examples are the full documents that need to be classified.

It is assumed than in the future, the United Nations University will provide more meta-information which describes the documents in more depth.

2.3.3 The Acknowledge platform interaction

One of the aims of this project was to develop a proof of concept demonstrator for an easy and accessible service in the field of learning and content management. This content should be accessed, exchanged and disseminated in an open, standards-based, service-oriented architecture. Hence, a platform interaction architecture was required in order to integrate the existing, often topical, and sometimes too fragmented research in this domain and embed it into a comprehensive service platform. This comprehensive service platform has repositories. These repositories are based in a set of standards specified under demand.

To allow this Acknowledge networked knowledge infrastructure interact with the Lemur index installed in the United Nations University, a Web service was developed. Figure 4.3 describes an overview of this system interaction.

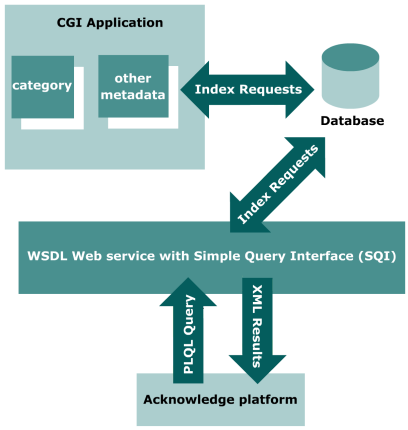

Figure 2.5: Overview of the system interaction between the Acknowledge platform and the Lemur search engine

The Acknowledge platform uses a standard Web Service Definition Language (WSDL). This standard is the Simple Query Interface (SQI). SQI is an Application Programming Interface (API) for querying. This is an open, collaborative effort, under the auspices of the Commité Européen de Normalisation/Information Society Standardization System (CEN/ISSS) Learning Technologies Workshop [15], to achieve interoperability between learning object repositories. The objective of these initiatives is to build a global Learning Network of learning object repositories. The SQI does not define a query language, it just defines an API for

querying as mentioned above. This interface defines the format to execute synchronous/asynchronous queries, create sessions for querying and so on.

One of the currently researched objectives of the Acknowledge Platform is to define a common query language. For the time being, this platform uses "ProLearn Query Language" (PLQL). PLQL is primarily a query interchange format defined over the query language that Apache Lucene uses internally. PLQL defines few levels of queries and few ways for representing results. PLQL is used by source applications for querying repositories. The specific name of the search engine used in the Lemur Toolkit is Indri. The Indri query language, based on the Inquery query language, is the query language which the Lemur Toolkit uses to query the index.

The Inquery language used by Lemur is different to PLQL. Hence, if the Lemur Index has to interact with the Acknowledge platform, there should exist a Web service with the Simple Query Interface and the queries should comply with the PLQL syntax. Such a Web service needs a compiler to translate the PLQL queries to the Indri queries and a translator that parses the result in the way that the specification of PLQL defines.

Chapter 3

Related research

3.1 Content extraction

This section presents a literature overview of the state-of-art in content extraction, also called Web Content Mining. Given a Web page, Web content Mining aims to identify and extract its important content from its less important or peripheral content. This may be referred to as "cleaning a Web page".

The most simple way to clean Web pages is to remove meta-data and tags from the source data. The derivation is a fast, single-pass process. However, most often a deeper processing is needed in order to extract the main content, because Web data are infiltrated with advertisements and interaction menus. Early approaches to the content extraction problem heavily relied on a priori knowledge of the Web site's layout and formatting [22, 36], knowledge which could eventually automatically be learned. However, such approaches suggest that only a limited amount of formating templates for Web pages are used, which is an unrealistic assumption.

Gradually the interest was awaken to build generic content extraction systems that operate on all types of Web pages. Usually the main text of a Web page is long and homogeneously formatted, while additional content is usually highly formatted and contains little and short texts. These and other signaling cues for relevant and irrelevant content were exploited in various ways. Han et al. (2001) start from the HTML tree and wrap its relevant content as a subtree that contains a large number of visible text elements, and which fans out into many children [29]. Gupta et al. (2003) use the high ratio of link content to detect navigational menus and similar structures [28]. Mantratzis et al. (2005) operate on the DOM (Document Object Model) tree, which defines the logical structure of well-formed HTML or XML documents, and identify hyper-linked clutter as text advertisements and long lists as syndicated references to other structures [38]. Pinto et al. (2002) detect a continuous part of the document which contains text based on the analysis of so called document slope curves [44]. A document is represented as a binary vector. HTML tags except for the ones that indicate content (e.g., font changes) are given a weight one, all other tokens are given a weight zero. From this vector a document slope curve is generated. The entries in the document slope curve graph correspond to the total of the binary vector entries up to and including each token. Long, low sloping regions of this graph represent content (text without tags). Weninger and Hsu (2008) use the text-to-tag ratio of lines of a document to find clusters of content in a Web page [48].

The closest to the approach showed in this project is the Content Code Blurring (CCB) method of Gottron (2008) that implements several methods to identify those parts of a Web page which contain a lot of text and few or no tags [27]. A document is represented as a sequence of text and tag (code) characters or tokens stored as a binary content code vector. The code vector is blurred by using a Gaussian blurring filter (by iteratively spreading the values of a character or token to its neighbors until the values stabilize), after which the areas with high content bearing values are extracted. A variant of the method, Adapted Content Code Blurring (ACCB), is better suited to wiki style documents and ignores anchor tags. Gottron makes a comparison with the methods described in [28] and [44], where he shows that the content code blurring method that ignores the anchor tags outperforms the former methods.

More sophisticated approaches extract the information from the visually rendered output of a Web page or other HTML content. Such an approach was followed by Krupl et al. (2005), who extracted tabular data from rendered pages [33], and Bergholz et al. (2008) who classified emails based on the content that is rendered in the email browser [17]. Although very valuable and generic, especially in an adversary setting where certain content might be present in the source, but hidden for the user of the browser, these approaches are computationally much more expensive than the method for Web page cleaning that we propose.

The method for Web page cleaning proposed in this thesis has a linear cost and performs better on average than the state of the art.

3.2 Search Engines (IR tools)

Regarding the state of the art in search engines, one can distinguish between two types of search engines. The first type of search engines are those that are freely used by Web users daily, the commercial search engines such as Google [7], Yahoo! [14], MSN-live [10], A9[1], ASK [3]. Note that there are further language-specific search engines, such as Baidu for Chinese [4], IN [8] for Greek, and so on. The second type of search engines are those that are freely available for research and development by the scientific community. They are often referred to as IR platforms. The main such experimental search engines are Lemur/Indri[40], Terrier [39], MG [49], Wumpus [41] and Apache/Lucene [2].

Chapter 4

Technical development

This chapter describes the technical development of the data set builder and search engine customization realized in this work. Section 4.1 presents the crawler developed. Section 4.2 describes the content extraction application. Section 4.3 details the search engine customization.

4.1 The Crawlers

There are two versions of the crawler developed in this work. The C version with the W3C Library, and the Java version with a HTML parser[1]. The C version is much faster than the Java version, but the Java version is more reusable; the evaluation of the noise reducer (described in Chapter 5) was made with the Java version for that issue. The main problem of the C version was that the libraries from W3C contain bugs. These bugs make the main application crash with a segmentation fault. On September 2003, W3C stopped work on the library; on January 2004, W3C passed over development of libwww to the Open Source community. Nevertheless since June 2002, there is not any official new release.

4.1.1 Structure of the C crawler application

The source code (ANSI-C) is divided into 6 principal files (.c) and 4 associated header files (.h).

1 The file main.c contains the topmost procedure main(). It calls the routines that initialize the W3C Protocol Library, and start the retrieving process with the associated functions defined in the other files.

2 The files getNewsList.c and getNewsList.h, implement the functions that the main program associates with the events related to the tag names. When the program detects a URL that contains a news list, it queues to the news queue.

3 The files getArticleList.c and getArticleList.h implement the functions associated with the second step of the main algorithm, that is, the main program associates functions with the events of tag names and only text, to tackle the news of the external Web

[1]The Hot-Java parser, available at HTTP://java.sun.com/products/archive/hotjava/

pages. A structure of the article is filled with the related information (id, title, file to save the article locally, external URL, content).

4 The files getArticle.c and getArticle.h provide a method to retrieve the external URL and clean it from noisy content. In a first step, functions are associated to events of retrieving tags and text only. Depending of the tag names that define the layout, the content get split in the positions of an array. At that point, the array is analyzed to determine the interval of the array that contains the non-noisy parts.

5 The file language.c, determines the language character map, depending of the URL that we are retrieving.

Technical Details

An introduction into crawlers and the main algorithm for crawling URLs was presented in Chapter 2.

The crawling program is developed in C using the W3C Protocol Library provided by the W3C (World Wide Web consortium). The libwww library is a general-purpose client side Web API. It is well suited for small applications, like the crawler that was developed. Before starting retrieving URLs, the libwww library has to be set up to define the way that the information has to be retrieved (robot, client, preemptive, non-preemptive and delays).

Typically crawlers include a time threshold, which corresponds to the maximum time that a crawler can wait for a Web page to respond. This is often referred to as "delay". In this work, we set this delay to 9 seconds. Depending on the step of the crawling algorithm described earlier in Chapter 2, the application associates different functions to the events related to the Web retrieving process, this is for example, if it finds a tag name with different attribute names and values, it can queue the URL that it contains.

After working few weeks with it and getting used to the library, it was found that a bug breaks the execution of the application. The W3C library contained a breaking debug point. This breaking debug point contained a bug, which was found during the development of the crawler. The source code of the library was modified to avoid this breaking point; an error code was returned instead of the execution's break. More specifically, this bug is located in HTTChunk.c. In the procedure HTChunkDecode_header(), there is a user breakpoint for the case the chunk length is not present in a line where it is expected.

In order to recover from such a request, the function should return an error code instead of a breakpoint.

Listing 4.1: Original code HTTChunk.c from the W3C Protocol Library

```
if (errstr == line) {
HTDEBUGBREAK("Chunk␣decoder␣received␣illigal␣chunk␣size:␣‘%s\’\n" _ line);
return NO;
}
```

The original version just continues, and returns a zero, causing the chunk decoder to break. It was just necessary to add the "return NO" code.

Listing 4.2: Modified code HTTChunk.c from the W3C Protocol Library

```
PRIVATE BOOL HTChunkDecode_header (HTStream * me) {
    char * line; _ASSERT( me );
   //linn line = HTChunk_data(me->buf);
   if (line) {
      char *errstr = NULL; me->left = strtol(line, &errstr, 16);
      /* hex! */
      HTTRACE(STREAM_TRACE, "Chunked.....␣‘%s\’␣chunk␣size:␣%X\n" _ line _me->left);
      if (errstr == line) {
        // HTDEBUGBREAK("Chunk decoder received illigal chunk size: ‘%s\’\n" _ line);
        printf("Chunk␣decoder␣received␣illegal␣chunk␣size:␣%.70s\n",line);
       return NO;
      }
```

After the bug that was found, and the ones that were not found, the decision was taken to migrate the application to Java.

The crawler application in C also contains a noise reducer. The cost of noise reducer in the C version has a linear complexity. It parses once the content of the file to clean, and at the same time the contents of the file are located in different positions of an array. The array gets a linear mathematical analysis to select the interval with the highest density.

Sometimes, the Hot Java parser has problems detecting few tags. For this reason, the program uses first regular expressions to remove the parts of the HTML source code that have not any value for the content extraction. These parts are styles, forms and HTML comments. Java uses a backtracking implementation for the regular expressions. In fact, the java.util.regex interface requires a backtracking implementation, because arbitrary Java code can be substituted into the matching path. This means that the previous task to do before the ported java algorithm costs more than the full C version. However, it does not matter a lot, because these times are really low if they are compared with the times that the Web pages need to be retrieved.

The HTML Parser from the Hot Java package is working in a similar way to the libwww. There are different parsers for HTML in Java. This is the one that uses less memory.

The port of the crawler and the noise reducer took around 20 hours.

4.1.2 Structure of the Java application

As we mentioned in the first section of this chapter, the application was ported from C to Java. This means that the structure of both applications maintains several similarities. Many principal files in C have corresponding classes in Java.

The class diagram in Figure 4.1 shows the main classes of the Java port. These classes are the following:

run This is the class which contains the main method of the entire application. The iteration diagram of Figure 4.3 shows the interaction between the different classes. On a first step, the main method retrieves the Web page which lists the different topics which list news. These topics are extracted with the ParseTopics class and are queued to a queue which contains Topics. Each Topic contains a URL which points to a Web page that contains a list of news. If the news article does not fit in a single page, there will be a link that contains another Web page with more news and so on. On a second step, each topic is handled to retrieve the Web page that has the lists of news. For each news item in the list, there is the title, a brief summary and the external URL which contain

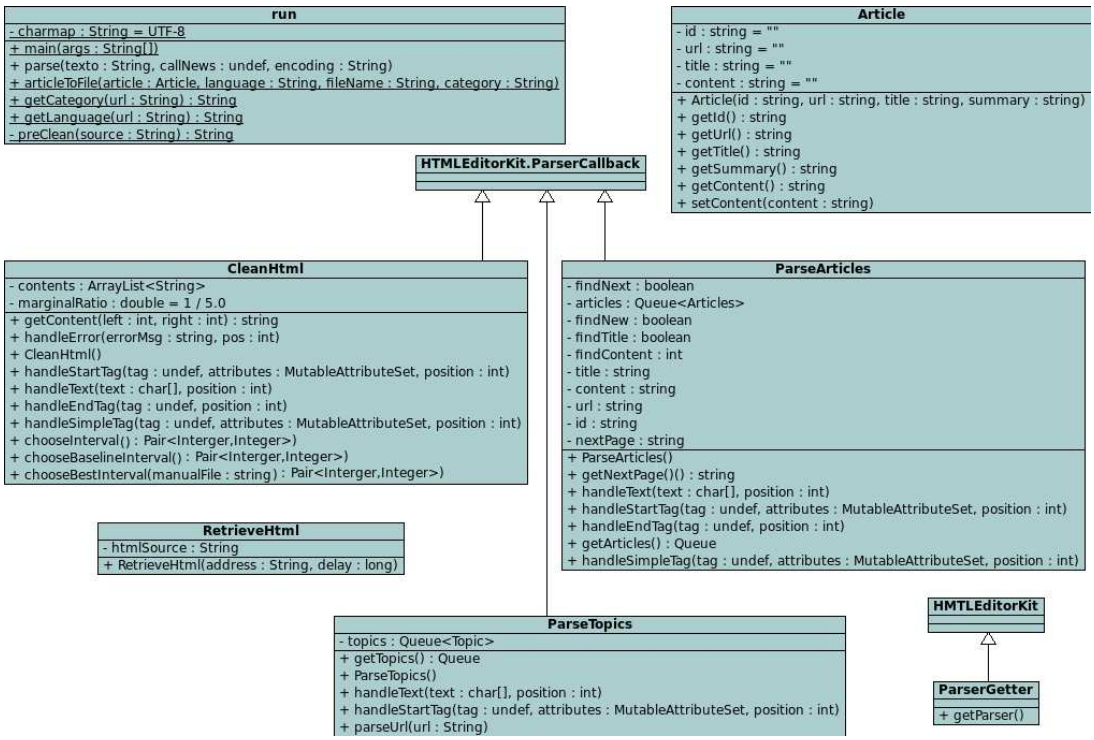

Figure 4.1: Class Diagram of the Java application.

the news. This information is used for creating instances of the class Article. On a third step, for every instance of the class Article, the content of the external URL is retrieved and cleaned using the methods of the class CleanHtml. Steps two and three are interspersed to visit the nodes of the tree in Depth-first search as we described in the definition of the objectives.

RetrieveHtml This is an auxiliary class. This class contains a method that given a URL and a time threshold, the HTML source is retrieved if it is possible in less than the establised time. This method shows the user-agent as "Firefox/2.0.0.11" to avoid the robot exclusion standard.

ParseTopics This class extracts the topics from the Web page that lists topics.

Topic This class contains attributes that describe the different topics and methods that allow the interaction with the attributes.

ParseArticles This class extracts the news text from the Web page that lists news. If there is a link with a link that contains the rest of the news. This URL will be added to a queue of Articles.

Article This class contains attributes that describe the news and methods that allow the interaction with the attributes.

CleanHtml This class implements the cleaner process. In each position of the private attribute Array List there is a string of text. These strings of text are the different parts of

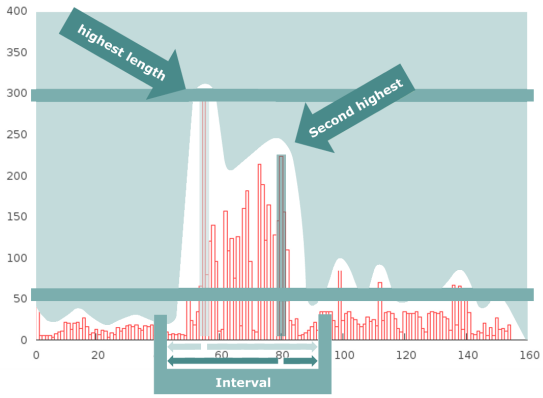

Figure 4.2: A graphical explanation of the cleaning algorithm

text of the Web page without the tag elements and in sequence. These strings have different lengths. The criteria for splitting the text in different positions of the Array List depend on the different tags that indicate the browser how to render the Web page. The method chooseInterval returns the interval in the array that probably will contain the content of the news. This interval has a high density of content. This interval is chosen with a linear mathematical approximation which chooses the two positions of the array with the highest length and finds the two intervals which contain these points using a marginal ratio parameter and a marginal gap parameter. The marginal ratio is 1/5 of the length of the position with the longest string in the array. When the number of consecutive positions with a length shorter than the marginal ratio is higher than the marginal gap parameter, the end of the interval is found. Figure 4.2 shows a graphical representation of this explanation. The Y axis represents the length of the strings and the X axis represents the different positions. It could be possible that these two intervals are the same interval. In the case that they are not the same, the one with the highest value of the length summation will be chosen. This method will be more fully discussed in the next section of this project.

ParseGetter This class is an extension of the HTMLEditorKit. The HTMLEditorKit is a robust HTML parser[2].

4.2 Content extraction

A method was developed for extracting the main content from Web pages. The difficulty of this task differs largely in different settings. First, there is the setting where the Web page is known at development time. Here, the structure of the page can easily be exploited to accurately extract the main content based on simple regular expressions. The second setting, where the Web page is unknown at development time, requires that a method is developed that can extract the main content from *any* Web page. This task is extremely challenging, since fully successful extraction methods need to perform a semantic analysis of both the text

[2]the Hot-Java parser, available at HTTP://java.sun.com/products/archive/hotjava/

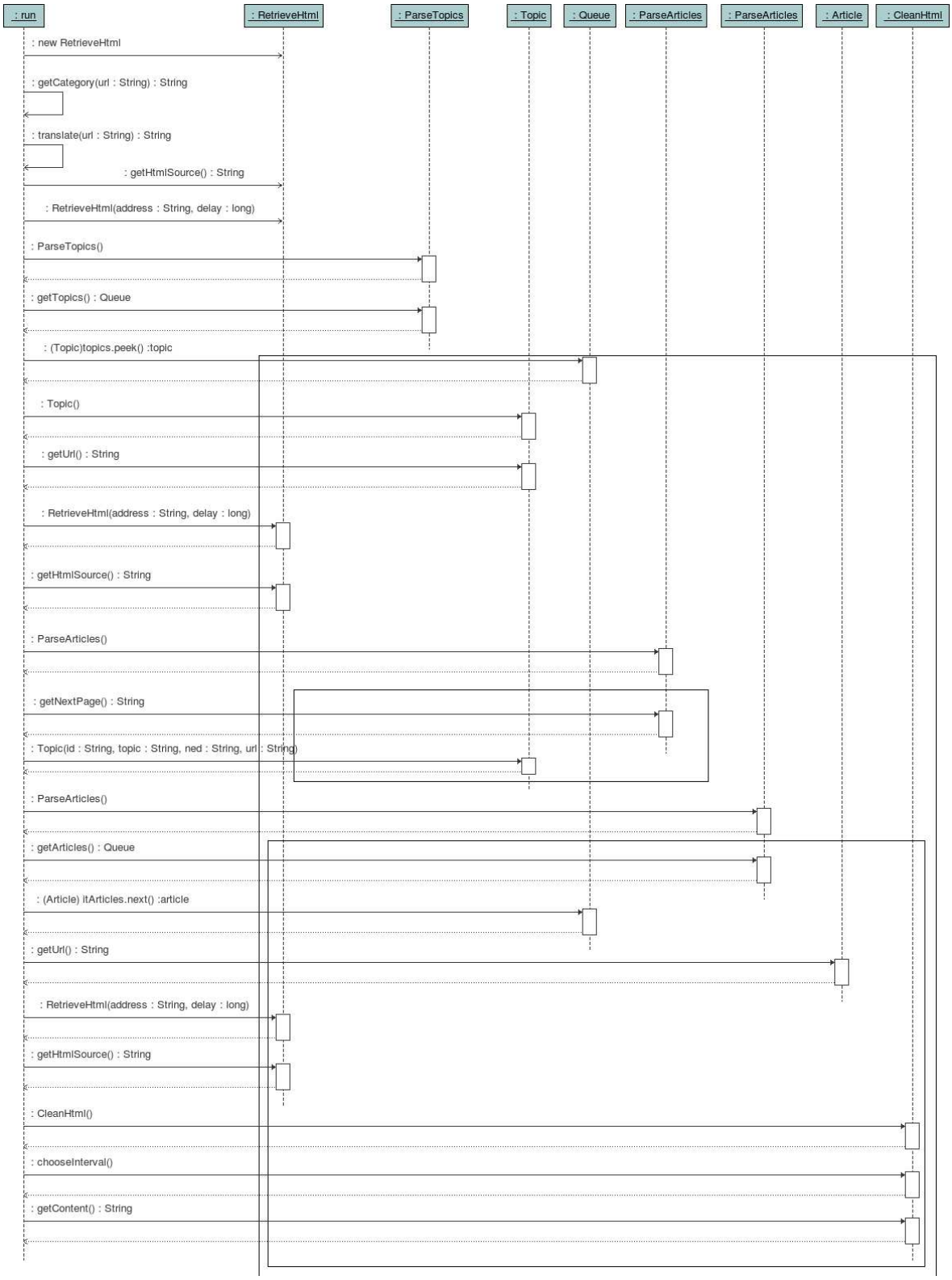

Figure 4.3: Iteration diagram of the Java application.

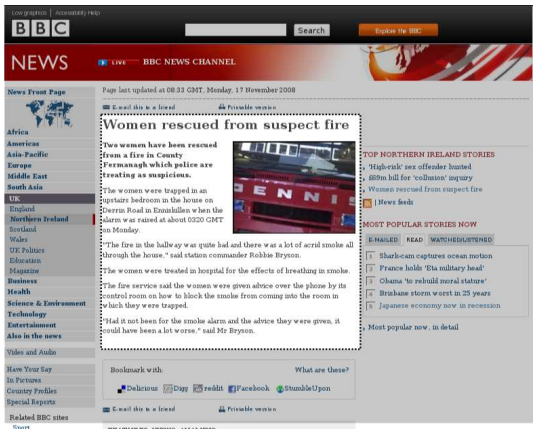

Figure 4.4: Example Web document with the main content marked.

and structure of the Web page. Furthermore, the task is even more complicated because of common mistakes in spelling, miswritten markup-tags and an ill-defined HTML-structure.

In this project the method proposed performs only a very shallow analysis of the Web page. This method does not depend on strong assumptions on the structure or content of the Web page and is fully language independent. The main idea behind our method is that a Web page has both content text (the news item, blog entry, etc.) and garbage text (navigational menus, links to other articles, adverts, comments, etc.); but that the content texts tend to be continuous, long text with little structural markup, and that the garbage text tends to be short texts with a lot of structural markup. We make the following weak assumptions: The first assumption states that the text representing the content is separated from the garbage text with one or more markup-tags. The second assumption states that no garbage text occurs in the main content, e.g., that the main content text is continuous (not taking into account the markup-tags). The third and most important assumption states that the main content of the text contains less structural markup-tags (see later) than the garbage text.

An informal inspection of some targeted Websites reveals that both assumption 1 and 2 are always satisfied, and these assumptions are also satisfied in our test set (see section 5.1.1). The third assumption, although intuitively correct, was violated in some cases. In section 5.1.3 we will discuss in detail when this occurred and the influence of this violation on the content extraction method.

We first locate a subset of markup-tags that modify the structure of the Web page. These tags include, but are not limited to `<p>`,`<table>`,`<br>`,`<div>`,`<h1>`,`<h2>`,.. and `<li>`. We ignore the tags that do not modify the structure of the Web page, such as `<b>`, `<a href=..>` and `<font ...>` and we also ignore data that is not content-related, such as Java Scripts, style definitions and HTML comments. We then transform the structured HTML page to a linear list of text strings $L = \{s_1, ..., s_n\}$. We parse the structure of the Web page using a robust HTML parser[3], that will, when presented with a not well-structured HTML page perform a best-effort parse. This parser visits every node in the HTML structure. If a node containing text is encountered, this text is added to the last text string in L. If a markup-tag that modifies the structure of the Web page is encountered, L is extended with one empty string. We continue this process until the entire Web page is parsed.

[3]the Hot-Java parser, available at HTTP://java.sun.com/products/archive/hotjava/

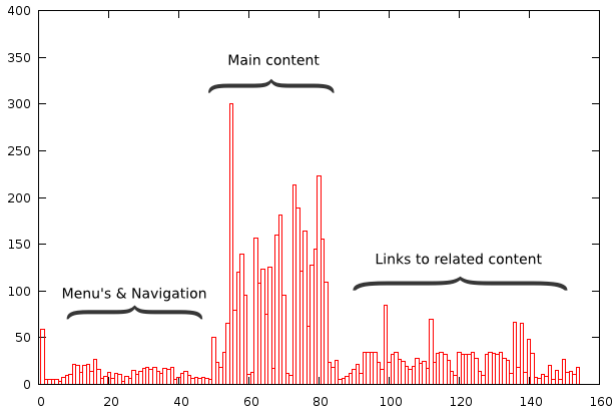

Figure 4.5: Example plot of the document density

We build a graphical representation of the array L in Figure 4.5 where the x-axis represents the position of the array and the y-axis represents the length of the strings at the different positions. In a second step we analyze this graph to find the main content in the Web page. Typically, the main content for a Web page containing news articles is located in the region of L that has the highest density. We therefore convert the problem of extracting the main content of a Web page in the problem of selecting the highest density region of L, for which we have designed a simple algorithm. We first locate the string s_{max} in L with maximum length $maxL$. Then a cutoff length $cutoffL$ is computed as $cutoffL = maxL * c_1$, where c_1 is a constant. We initialize the high density region R as $R = \{s_{max}\}$. We then incrementally add strings s_i to R until no more strings can be added. A string s_i is added to R iff $length(s_i) > cutoffL$ and there is a string $s_j \in R$ such that $|i - j| < 4$. The algorithm terminates when no more strings can be added to R.

To create the final text T containing the main content of the Web page, we find the leftmost string s_l in R and the rightmost string s_r in R. We then create T by concatenating all strings s_i, where i ranges from l to r (inclusive).

Although this algorithm is very simple, it incorporates several interesting ideas. First of all, it does not depend on the structure of any particular Website, but uses a notion of document density which can be expected to be universal for most Website containing news articles. Secondly, it does not depend in any way on the text and is thus fully language independent. Thirdly, it relies only to a limited extend on the HTML-markup, thus allowing for dirty and non-well structured Web pages.

4.3 The Lemur Toolkit

The remainder of this chapter is organized as follows: In the next section the installation of this platform under Unix operating systems will be described in detail. Section 4.3.2 presents the setup of this information retrieval platform that allows managing meta-information. The last section of this chapter describes the technical development that integrates this information retrieval system with the Acknowledge platform.

4.3.1 Installation on Linux

This section will cover the installation of the Lemur Toolkit and the modified Common Gateway Interface application (CGI). CGI is a standard protocol for interfacing external application software with an information server, commonly a Web server, in this case Apache [23].

The Lemur Toolkit is programmed in C, C++, including small scripts in Perl and Java. The installation of Lemur under a Linux system requires a C compiler, C++ compiler and the following libraries:

- z (Suse: zlibdevel, Debian: zlib1g-dev)
- iberty (Suse: binutils-devel, Debian: included in the package binutils-dev)

These libraries are not referenced in the documentation of the Lemur Web site [40] but are compulsory for an installation in a Unix system from the source code.

Typically, Unix distributions provide package managers. These package managers give an interface to install or remove software, but it is possible as well to install software manually from the source code of the application. The Lemur Toolkit provides the platform in several manners:

- The Indri search engine separately from the platform
  - A Windows 32-bit (i386 platform dependent) installer for the binaries
  - The source code of the search engine
- The Lemur Toolkit
  - A Windows 32-bit (i386 platform dependent) installer for the binaries
  - A MacOSX 32-bit (i386 platform dependent) package for the binaries
  - The source code of the Lemur Toolkit (Platform-Independent)

The next steps describes how to install the Lemur Toolkit in a generic Unix distribution from the sorce code:

1 Download the Lemur Toolkit [4] from the Source Forge repository [12]

2 Unpack the source. On the command line, type the following commands to unpack the package. This should create a directory named Lemur-4.7:

> gunzip Lemur-4.7.tar.gz

> tar -xvf Lemur-4.7.tar

3 Configure the makefiles. To configure Lemur with the default libraries type:

$Lemur - 4.7 >$ ./configure

If one wants to compile and install the java wrappers, one should run:

[4] Lemur-4.7.tar.gz

$Lemur - 4.7 >$ ./configure –enable-java

This option is disabled by default, and it is not necessary for the goal of this project, but the java wrappers can be useful to deal with Lemur Toolkit. If one wants to define the installation directory, one have to set it explicitly:

$Lemur - 4.7 >$ ./configure –prefix=/opt/Lemur-4.7

4 Compile Lemur. Type "make" from the Lemur-4.7 working directory. This will compile the whole Lemur Toolkit.

$Lemur - 4.7 >$ make

5 Install the Lemur Toolkit. After compiling Lemur, type "make install". This will install the Lemur applications, libraries and include files according to the directory specified by the prefix option of the configure script.

One can check if the installation of the application succeeded correctly, running the Shell scripts located in the "data" folder under the Lemur installation directory.

4.3.2 Meta-information

The Lemur Toolkit handles different classes of documents, by using internally the ANTLR Parser Generator. These known classes are:

html – Web page data

trecWeb – TREC[5] Web format, e.g., terabyte track

trectext – TREC format, e.g., TREC-3 onward

trecalt – TREC format, e.g., TREC-3 onward, with only the TEXT field included

doc – Microsoft Word format (windows platform only)

ppt – Microsoft Powerpoint format (windows platform only)

pdf – Adobe PDF format

txt – Plain text format

These parsing utilities have been designed with flexibility and extendability in mind. This means that if a new functionality is required and it is not currently implemented by the toolkit, it should be easy to add the functionality and plug it into the parser framework.

As mentioned earlier, the Lemur Toolkit allows indexing different kinds of files (pdf, txt, TrecText, XML, trecWeb), but if the pdf file is indexed directly, the meta-information associated to this file cannot be handled by the search engine. The format that suits more properly for our purposes is the TrecText, i.e., files in the TREC format, which is the file format accepted by Lemur. This format allows defining fields.

[5]Text Retrieval Evaluation Conference (TREC) is a standard evaluation framework for IR systems, organised by the U.S. National Institute of Standards and Technology. http://trec.nist.gov/

The TrecParser and the TrecWebParser remove contractions and possessives, have a simple acronym recognizer, and convert words to lowercase. This means that it is not necessary to specify an explicit acronym list, even if it is possible. These parsers recognize text in the TEXT, HL, HEAD, HEADLINE, TTL, and LP fields. It is possible to define more fields to be indexed. In this approach, it is useful because these fields will be the categories, the entities, and the fields that will be needed in the future.

These documents that depend on meta-infomation can be handled in different manners:

- Through the command-line
- Through the C++ API
- Through the Java API
- Through the C Sharp API

The code from the listing 4.3, shows how to change the current index used by the CGI application described in the next section. This Shell script uses the "IndriBuildIndex" commandline to generate the index. One can specify various parameter values directly on the command line. One of these parameter values can be a parameter file. Parameter files for IndriBuildIndex are well-formed XML documents that must be wrapped in $< parameter >< /parameter >$ tags. To specify the use of a parameter file on the command line, use:

$>$ IndriBuildIndex $< parameter\_file >$ [$< parameter\_file\_2 >$ ... $< parameter\_file\_n >$]

Note that one can specify more than one parameter file (say, if one has a standard set of stopwords, one wishes to use for all the indexes one builds). The Shell script 4.3 uses one parameter file. The interaction between the Lemur Toolkit and the application which categorizes documents mentioned in the objectives is discussed below.

The categorization application was designed with this interaction in mind. This application generates TrecTex files from the original documents, and adds meta-information which describes the document. The location of the original document is specified in the URL field. The application sets categories according to the contents of the documents analyzed according to a trained data set. This information is specified in the CATEGORY field.

Listing 4.3: Shell script that renew the index

```
\#!/bin/sh
cd /tmp
rm -Rf /tmp/data
cp ‘‘theClassifiedDocuments’’ data
rm -Rf indri_index
mkdir indri_index
IndriBuildIndex bld_par && rm -Rf /var/www/Lemur/indri_index && mv indri_index /var/www/Lemur
```

The format of these well-formed XML parameter files for IndriBuildIndex is specified below.

- Specifying Source Data

corpus: a complex element containing parameters related to a corpus. This element can be specified multiple times. For each corpus parameter, you can specify the following items:

path: The pathname of the file or directory containing documents to index. Specified as $< corpus >< path > /path/to/file\_or\_directory < /path >< /corpus >$ in the parameter file and as $-corpus.path = /path/to/file\_or\_directory$ on the command line.

class: The FileClassEnviroment of the file or directory containing documents to index. Specified as $< corpus >< class > trecWeb < /class >< /corpus >$ in the parameter file and as $-corpus.class = trecWeb$ on the command line. For a list of default known classes, see the Indexer File Formats.

annotations: The pathname of the file containing offset annotations for the documents specified in path. Specified as $< corpus >< annotations > /path/to/file < /annotations >< /corpus >$ in the parameter file and as $-corpus.annotations = /path/to/file$ on the command line. For a full description of how to use offset annotations, see Inline and Offset Annotations.

metadata: The pathname of the file or directory containing offset metadata for the documents specified in path. Specified as $< corpus >< metadata > /path/to/file < /metadata >< /corpus >$ in the parameter file and as $-corpus.metadata = /path/to/file$ on the command line.

- Index Parameters

index: path to the Indri Repository to create or to add to. Specified as $< index > /path/to/repository < /index >$ in the parameter file and as -index=/path/to/repository on the command line.

- Memory and Optimizations

memory: an integer value specifying the number of bytes to use for the indexing process. The value can include a scaling factor by adding a suffix. Valid values are (case insensitive) K=1000, M=1000000, G=1000000000. So 100M would be equivalent to 100000000. The value should contain only decimal digits and the optional suffix. Specified as $< memory > 100M < /memory >$ in the parameter file and as -memory 100M on the command line.

offsetannotationhint: An optional parameter to provide a hint to the indexer to speed up indexing of offset annotations when using offset annotation files as specified in the $< corpus >$ parameter. Valid values here are "unordered" and "ordered". An "unordered" hint (the default) will inform the indexer that the document IDs of the annotations are not necessarily in the same order as the documents in the corpus. The indexer will adjust its internal memory allocations appropriately to pre-allocate enough memory before reading in the annotations file. If you are absolutely certain that the annotations in the offset annotation file are in the exact same order as the documents, then you can use the "ordered" hint. This will tell the indexer to not read in the entire file at once, but rather read in the offset annotations file as needed for only the annotations that are specified for the currently indexing document ID.

- Stopwords and Stemming

stopper: a complex element containing one or more subelements named word, specifying the stopword list to use. Specified as $< stopper >< word > stopword < /word >< /stopper >$ and as $-stopper.word = stopword$ on the command line. This is an optional parameter with the default of no stopping.

stemmer: a complex element specifying the stemming algorithm to use in the subelement name. Default valid options are 'Porter' or 'Krovetz' (case insensitive). Specified as $< stemmer >< name > stemmername < /name >< /stemmer >$ and as $-stemmer.name = stemmername$ on the command line. This is an optional parameter with the default of no stemming.

- Specifying Metadata and Fields

metadata: a complex element containing one or more entries specifying the metadata fields to index, e.g., title, headline. There are three options:

field: Make the named field available for retrieval as metadata. Specified as $< metadata >< field > fieldname < /field >< /metadata >$ in the parameter file and as $metadata.field = fieldname$ on the command line.

forward: Make the named field available for retrieval as metadata and build a lookup table to make retrieving the value more efficient. Specified as $< metadata >< forward > fieldname < /forward >< /metadata >$ in the parameter file and as metadata.forward=fieldname on the command line. The external document id field "docno" is automatically added as a forward metadata field.

backward: Make the named field available for retrieval as metadata and build a lookup table for inverse lookup of documents based on the value of the field. Specified as $< metadata >< backward > fieldname < /backward >< /metadata >$ in the parameter file and as $metadata.backward = fieldname$ on the command line. The external document id field "docno" is automatically added as a backward metadata field.

field: a complex element specifying the fields to index as data, e.g., TITLE. This parameter can appear multiple times in a parameter file. If provided on the command line, only the first field specified will be indexed. The subelements are:

name: a required field specifying the field name, specified as $< field >< name > fieldname < /name >< /field >$ in the parameter file and as $-field.name = fieldname$ on the command line.

numeric: a optional parameter that specifies if the field contains numeric data (by specifying "true"), otherwise the symbol false, specified as $< field >< numeric > true < /numeric >< /field >$ in the parameter file and as $-field.numeric = true$ on the command line. This is an optional parameter, defaulting to false. Note that 0 can be used for false and 1 can be used for true.

parserName: an optional parameter that contains the name of the parser to use to convert a numeric field to an unsigned integer value. The default is NumericFieldAnnotator. If numeric field data is provided via offset annotations,

you should use the value OffsetAnnotationAnnotator. If the field contains a formatted date (see Numeric and Date Fields in Indri) you should use the value DateFieldAnnotator.

The parameter file 4.4 will create (or add to) an index at *./indri_index/*. The indexer will use a soft-limit of 500M of RAM before flushing out its internal indexing buffers to disk. The source data for the example comes from a corpora located at *./data/*. The parameter file also specifies that stemming is to be performed using the Krovetz method, and that the fields (language, title, headline, date, category and url) should be made available for searching on.

Listing 4.4: The file build_param describes how to generate the index with the specific meta-information that was required

```
<parameters>
   <index>indri_index/</index>
   <corpus>
      <path>./data</path>
           <class>trectext</class>
   </corpus>
   <memory>500m</memory>
   <stemmer>
     <name>krovetz</name>
   </stemmer>
   <field>
     <name>LANGUAGE</name>
   </field>
   <field>
        <name>TITLE</name>
   </field>
   <field>
        <name>HEADLINE</name>
   </field>
   <field>
        <name>DATE</name>
        <numeric>true</numeric>
        <parserName>DateFieldAnnotator</parserName>
   </field>
   <field>
        <name>CATEGORY</name>
   </field>
   <field>
        <name>URL</name>
   </field>
</parameters>
```

4.3.3 CGI application

The United Nation University requires a Web page which interacts with the generated index. Inside the source code of the Lemur Toolkit, there is a CGI application implemented which uses the C++ API to handle the index. This is referenced in the documentation of the Lemur Toolkit [9] as the "LemurCGI package". This is the most efficient and the fastest solution to interact with the index through a Web interface. This CGI was adapted to the needs of the requirements.

The current version of the LemurCGI package is the Version 5.0, updated the 4th of March, 2006. The documentation that describes the LemurCGI package available is just a reference of functions and types for each file of the source code. In the next section there is a description of the structure of the LemurCGI package with the changes made for this solution.

Structure of the CGI application

This is a description of the application based on the structure of the folders.

- LemurCGI.cpp This file contains the main function, that is the point by where the CGI application starts the execution. The CGI application returns a Web page which contains the search interface, the results, the query's help, the index information and the error messages. The most important function included in this file is *void processRequest(CGIOutput *output)*, this function makes different actions depending on the parameters that are processed. These actions are returning the HTML code of the search box, returning the HTML source code of a Web page which lists the help, collecting all the parameter for committing the query and so on. The modification that this project includes on this file is the modification of this function, to allow collecting the new parameters (category, language and date interval) and constructing the proper query to be committed.

- CGIConfiguration.cpp/.hpp This class configures the setup of the CGI application and uses a constructor to initialize a default configuration that describes where is the template path, the indexes location and the root path of the application. This configuration can be defined explicitly from a file using the function *bool CGIConfiguration::readConfigFile(char *filePath)*. This class has an public attribute that is a dictionary hash. This dictionary hash is used to store temporally a set of pairs that is formed as key/value. The class includes functions that interact like an interface over the functions of the DictionaryHash class that sets and gets the values of the pairs. The modification that this project includes on this file is a function that reads a configuration from a well-formed XML file and returns a HTML drop-down box with the options that the XML file defines. The Lemur Toolkit provides an interface to load XML hierarchies. The structure of this XML file is described after this listing.

- DictionaryHash.cpp/.hpp This class creates and manages a dictionary hash. A dictionary hash is a set of pairs of key-value.

- SingleResultItem.cpp This class provides functions to show the attributes for each item of the results of the query. The function *string SingleResultItem::toString()* returns a HTML string that shows the results of a single item according of the structure defined in the template "SingleResult.html".

- CGIOutput.cpp This class provides functions to load templates and replace the values of these templates according to the values of the results of the queries. These templates define patterns that describe the structure of the Web pages. The modification that this project includes on this file is a modification of the search box. This modification adds two Java Script pop up calendars for selecting the interval of the dates of the search, and two drop-down box menus to choose the language and the category of the

contents. These drop-down boxes are generated using the *readConfigFile* defined in the CGIConfiguration class. It means that the program does not need to be recompiled if the categories or languages are changed, it will be necessary to change just the related files ("bin/templates/category.xml" and "bin/templates/language.xml").

- IndriSearchInterface.cpp This class provides an interface to interact easily with the index. This interface provides functions that allow actions like removing duplicate results, normalizing the URLs that locate the cached documents, displaying the instances of the results and setting the environment scoring rules of the search. The modification that this project includes on this file is the function *getFieldValue* that reads a specified XML node that describes an attribute in the TrecTex cached document and returns the value. The Class Indri::QueryEnvironment of the Lemur Toolkit provides the function *StringVector documentMetadata (ScoredExtentResultVector documentIDs, string attributeName)* which retrieves the value of the specified attribute of the specified document, but this functions does not work with the attributes that are different than "title".

- DBInterface.cpp This class represents a single result item for the results. This class provides a function for retrieving the passage of the document that includes the term of the search to generate a summary for each item of the results. This class also provides functions for handling the indexes and the object that defines the word stemmers.

- bin (folder)
  - Lemur.config This file describes a configuration in a well-formed XML structure. This structure defines the location of the templates and the indexes. This file can describe more than one index. For each index, the required field is the location, a description of the index can be defined in a optional field "< *description* >".
  - help-qry.html A template HTML file which lists how to formulate queries.
  - help-db.html A template HTML file which describes briefly the contents of the document set being searched by the Lemur search engine.
  - Lemur100.gif A Lemur logo of 100x79 pixels which is instanced by the templates when the cached document is displayed.
  - Lemur150.gif A Lemur logo of 150x119 pixels which is instanced by the templates when the search box or the results are displayed.
  - js (folder)
    - Calendar.js This file is a calendar popup [34]. This calendar is instantiated from the Lemur search box to show the user a friendly interface for selecting the date interval.
  - Lemur.cgi The generated executable of the CGI application.
  - templates (folder) This folder contains the templates which the executable uses for rendering the Web interface and loading the categories and the languages.
    - GenericPage.html Template which contains the template for loading the Lemur search box.
    - SearchPage.html Template which contains the template for loading the Lemur search box and the links to the help pages.

* ErrorPage.html Template which contains the template for loading the error's report of the failures from the search.
* ResultsPage.html Template which contains the template for loading results of the query.
* SingleResult.html Template which contains the template for loading the different attributes of each result item. These attributes are the title, the original URL, the summary, the score and the URL of the cached document.
* category.xml The XML file which describes the different categories.
* language.xml The XML file which describes the different languages.

This (DTD) 4.5 describes the well-formed XML structure of the file that describes the values to load in the HTML drop-down boxes by the $readConfigFile$ function.

Listing 4.5: This file contains a set of options

```
<!ELEMENT options (value*)>
<!ELEMENT value (name, short)>
<!ELEMENT name (#PCDATA) >
<!ELEMENT short (#PCDATA)>
```

Perusing the description 4.5 line by line:

- 1 < *options* > is a valid object name. The asterisk means that it can be possible 0 or more values.
- 2 < *value* > is a valid object name. This tag contains compulsory the name tag and short tag.
- 3 < *name* > is a valid object name. This tag contains characters which gives a long description of the value. This description allows white spaces.
- 4 < *short* > is a valid object name. This tag contains characters which can gives a short description of the value.

The "language.xml" file 4.12 is an example of a XML file which uses this DTD.

Listing 4.6: This file contains an example of the different languages available

```
<options>
  <value>
    <name>English</name>
    <short>English</short>
  </value>
  <value>
    <name>Dutch</name>
    <short>Dutch</short>
  </value>
  <value>
    <name>Spanish</name>
    <short>Spanish</short>
  </value>
</options>
```

"LemurCGI.cpp" contains the function $voidprocessRequest(CGIOutput * output)$ which processes the parameters. The original version of the CGI application takes the text from the field of free text and commits the query to the information retrieval platform. This modified version for this project collects the free text and all the meta-information from the parameters that the form generates, compounds the query and commits the query. The code 4.7 lists the most relevant parts from this modification. The original version of the code realizes the query inside the bucle because the query is composed with the value of one parameter. The modified version of the function commits the query outside of the bucle, when all the meta-information and the free text are processed.

Listing 4.7: The modification of the LemurCGI.cpp

```
std::vector<t_kvPair*>::iterator vIter=queryParameters.begin();
while (vIter!=queryParameters.end()) {
  string thisKey=(*vIter)->key;
  string thisVal=(*vIter)->value;
  ...
    } else if (thisKey=="query") {
      if (thisVal=="") {
        output->displayDefaultSearchPage();
      } else {
        //db.search(datasourceToUse, thisVal, maxDocsToRetrievePerPage,
        // startRank, currentQueryType);
        compQuery+=thisVal;
        hasQuery=true;
      }
      hasOutput=true;

    } else if (thisKey=="until") {
      if (thisVal!="MM/DD/YYYY") {
        compQuery ="#band(#datebefore("+thisVal+")␣" + compQuery + ")";
      }

    } else if (thisKey=="from") {
      if (thisVal!="MM/DD/YYYY") {
        compQuery ="#band(#dateafter("+thisVal+")␣" + compQuery + ")";
      }

    } else if (thisKey=="language") {
      if (thisVal!="") {
        compQuery ="#band("+thisVal+".language␣" + compQuery + ")";
      }

    } else if (thisKey=="category") {
      if (thisVal!="") {
        compQuery ="#band(#1("+thisVal+").category␣" + compQuery + ")";
      }
    }
  vIter++;
} // end while (vIter!=queryParameters.end())

if(hasQuery){
  db.search(datasourceToUse, compQuery, maxDocsToRetrievePerPage,
            startRank, currentQueryType);
}
```

```
if (!hasOutput) {
  output->displayDefaultSearchPage();
}
```

Installation of the CGI application

The default configuration of the Apache[23] HTTP Server does not run CGI applications. The reason of this default configuration is because of safety measures. Typically, Web developers do not design Web applications with effectiveness in mind, they use interpreted languages instead of compiled languages. The main benefit of using an interpreted language is that it does not need to be compiled. It is not recommended to provide dedicated Web Servers with compilers, because it reduces the security of the system. The code listing 4.8 needs to be added in the configuration file $/etc/apache2/sites-enabled/000-default$. These configuration changes will allow the HTTP Server run a CGI application with the extension ".cgi". Once that this modification is done, the HTTP Server has to be restarted.

Listing 4.8: Setup configuration to allow CGI applations in the HTTP Server

```
<Directory /var/www/Lemur>
        Options +ExecCGI
        AddHandler cgi-script .cgi
</Directory>
```

Typically a compiled program, an executable, depends on the versions of libraries of the system that is compiled. The application can be statically compiled, that is the libraries are included inside the executable. This solution has more disadvantages than benefits, because the size of the executable increases drastically, and when the libraries that the system provides get updated, the executable points to the previous version.

This project found a solution that gives benefits without the disadvantages exposed here. The solution consists in a replication of the entire HTTP Server with the compilers and the same versions of the libraries. The CGI application gets compiled in the replica of the server, and the compiled applications are hosted in the real HTTP Server. The replica of this server can be installed as a virtual machine in the computer of the developer. This does not impose any extra cost: the Vmware player [13] can run virtual machines on Linux or Windows PCs for free, and the Operative System used as a HTTP Server is available for free. The Web server installed at the United Nations University is a Linux box, specifically a Debian GNU/Linux 4.0r5 [16].

There are two versions of the CGI application installed in the HTTP Server of the United Nations University. These two versions were demonstrators. They were developed for the Event celebrated in Gent by IBBT. They can be tested from these addesses:

- http://www.cris.unu.edu/Lemur/Lemur.cgi
- http://www.cris.unu.edu/Lemur/category/Lemur.cgi

The first demonstrator was an application that handles an index generated from crawling Google News during one week. The categories are assigned by Google. The meta-information used in this application is the language of the news item, the category and the date when the news were retrieved. There is a field for introducing free text. When the user introduces free

text, selects the meta-information from the menus and presses "Search", the internal query is displayed to the user and the list of documents that match with this query. This list contains a list of items. Each item is a match of the query realized. The items are sorted depending on the ranking value returned by the information retrieval platform. Each item contains:

- the title of the news
- the internal identifier of the news
- a brief summary of the news
- the score of how much the news item matches the query
- the url of the external link which contains the original news
- the url of the internal link which contains the cached news

The second demonstrator was an application that handles an index generated from the documents classified using the automatic classifier. The meta-information used in this application is just the category. The application works as the first demonstrator.

The source code of this CGI application is available in the Acknowledge intranet and in the cd attached to this project.

To compile the CGI application which uses the Lemur API on Unix, one will need both the Lemur library file and all the header files. The installation script of Lemur puts the library file in "prefix/lib/libLemur.a" and all the header files in "prefix/include/". Header files in C have the extension of .h, while a C++ header file has an extension of .hpp.

The application level Makefile that was used in this project is the Makefile.app from the top level directory of the Lemur installation. The following values were edited:

OBJS – list of each of the object files needed to build the application (LemurCGI.o CGIConfiguration.o CGIOutput.o DBInterface.o IndriSearchInterface.o DictionaryHash.o SingleResultItem.o)

PROG – name for the application (bin/Lemur.cgi)

The Lemur library can be used exactly in the same way as any other C++ library. These are the steps which describe how to use the library:

- In the C++ code, include the relevant Lemur header files.
- When compiling the code, use -Iprefix/include as an option so that the compiler can find the included files. (where prefix is as specified when running configure.)
- When linking the code, use -Lprefix/lib as an option so that the linker can find the Lemur library. Also, one needs to specify -lLemur as a linking option to indicate that the Lemur library is needed to be linked with the code. See Makefile.app for the list of other libraries that are required to link with -lLemur. Be careful about the order of the libraries you specified. The order reflects the assumed dependency among the libraries.

The next steps describe how to install the Common Gateway Interface application in a generic Linux distribution:

1 Download the modified version of the Lemur CGI front-end [6] from the deliverables hosted at the Acknowledge intranet

2 Install the modified cgi. Copy the file cgi.tar.gz to the Lemur directory and unzip it.

> cp yourdownloadfolder/cgi.tar.gz Lemur-4.7/

> cd Lemur-4.7

> tar zxf cgi.tar.gz

3 Compile the modified cgi.

> make

4 Copy the modified cgi, and the dependent files to a folder where the server application can run executables cgi files

> $cp - Rf\ Lemurdirectory/site - search/cgi/bin/ *\ serverdirectory/htdocs/cgi - Lemur/$

5 Edit the index information at the "Lemur.config" file (which is located in the same directory as Lemur.cgi)

The "Lemur.config" configuration file is a well-formed XML file with the opening tag <Lemurconfig>. There are two required elements within the configuration file:

- <templatepath> This tag should reflect the path (either relative or absolute) to the template files
- <indexes> This section contains information about what indexes are available, and can contain as many indexes as needed. For each <index> item, there should be two elements:
  - <path> This element must be set pointing at where the index is located
  - <description> This element is optional. This tag can be set to be a description of the pointed index.

4.4 The Acknowledge platform integration

As it was mentioned in the chapter of the objectives, the Acknowledge platform needs to be integrated to the customized search engine, hence an extra application is required. This application is a Web service which receives a synchronous query in PLQL (the query language used by the Acknowledge platform) and provides the results in the format established by PLQL. In the meantime, the processing of the search engine is done using Indri's query language, Inquery.

The Web service is developed using Java Development Kit from Sun Microsystems [46] and a Web Services Description Language Binding for SQI [19] hosted at Source Forge [11]. This Web Service provides a method that realizes a synchronous query in a concrete query language and returns the results in a predefined XML format. The query language that the

[6] cgi.tar.gz

method accepts and the format of the results are not the same that the information retrieval platform uses internally (Inquery). To approach this goal, the Web Service uses a method that calls a compiler which translates the accepted query language to the Inquery, and another application to interact with the index and format the results in this predefined XML format.

4.4.1 The PLQL to Indri compiler

This application translates queries in PLQL format to queries in the Indri query format. The Indri query format is the format of the queries that the Lemur Toolkit uses inside the search engine.

This application is based on the compiler that the Barcelona School of Informatics (FIB) uses to teach the course about compilers in the Degree in Informatics Engineering. This compiler was used to translate the queries because it is methodical and well structured. In the case of the PLQL translator, it is a subset of the compiler which was taken by reference.

The PLQL query language does not have procedures/functions, but just one scope of visibility. In addition, PLQL does not have a definition of variables, but only has meta-data structures.

A compiler was needed because the syntax of the two query languages has different structure. The PLQL language uses a query syntax which is infix, and the Indri search engine uses a prefix syntax. This means that if the semantic meaning of the query should be respected, the easier way to develop the required application is developing a compiler.

The PLQL language

The specification of the language is needed to specify the grammar of the input language. This specification is getting updated often. The reference that was taken to develop the compiler is [47], specifically the Level 1.

The Indri query language

The Indri query language [45] was designed to be robust. It can handle both simple keyword queries and extremely complex queries. Such a query language sets Indri apart from many other available search engines. It allows complex phrase matching, synonyms, weighted expressions, Boolean filtering, numeric (and dated) fields, and the extensive use of document structure (fields), among others. The document [45] provides a broad overview of the language, including the grammar of the query language. One of the most different aspects between the PLQL language and the Indri query language is the order of operators, PLQL is infix and Indri is prefix.

The structure of the compiler

PLQL [47] handles the following structured documents which describe meta-data schemes:

- The IEEE Learning Object Metadata (LOM) [20] is a hierarchical meta-data standard usually encoded in XML, published by the IEEE in 2002. Its purpose is to enable the description of learning objects through attributes that include the type of object, author, owner, terms of distribution, and format, as well as pedagogical attributes, such as typical learning time or interaction style.

- Dublin Core (DC) [6] is a standard for generic resource descriptions. The simple DC meta-data element set consists of 15 elements, including title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage, and rights.

- MPEG-7 is an ISO/IEC standard for describing multimedia content. MPEG-7 Multimedia Description Schemes (DSs) are meta-data structures in XML that facilitate searching, indexing, filtering, and access [31].

PLQL handles the meta-structures in a tweaky way. It does not check with the grammar if the structure is correct or not. The structure is checked during the processing of the query. This means that it is not checked, it just fails and it does not give results.

When there is a XML data structure which has different levels, the way to browse the different levels of the structure is using a DOT to go to the next level. In the Acknowledge platform, the way to "check" the structure during the running time is to label directly the fields of the search engine with the name of the full deployed structure. For example, a name of a field to be indexed could be "<lom.general.title.langstring.string>".

Another important point of PLQL is that it is defined over the language that Apache Lucene uses internally.

The conclusion of these last two paragraphs is that it is not possible to reuse the grammar defined in the implementation of PLQL because it is not implemented completely; it is just implemented partially and the queries are forwarded to the Lucene search engine. On one hand, the approach of reusing the grammar of the Lucene search engine and integrate changes that are defined in PLQL, is more complicated than defining a new grammar. On the other hand, the lexical analysis of the standards used here, and the generation of code of these standards cannot be totally completed during the time dedicated for this approach. The LOM lexical analysis is implemented following the definition of the standard [20]. In the case of this standard code is generated just for the meta-information that matches with the information that was provided in the solution for the United Nation University.

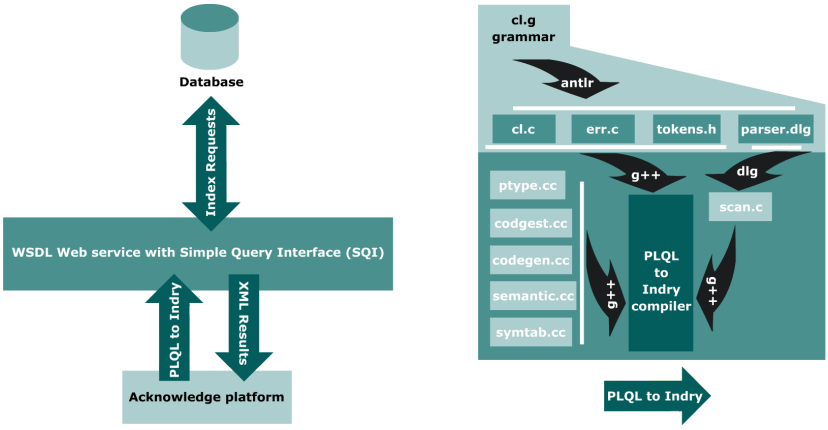

Figure 4.6: Structure of the compiler which translates PLQL to Indri

The overview of the structure of the compiler, which contains several modules, is:

- cl.g (with the grammar definitions and AST construction).

- semantic.cc and semantic.hh (with the recursive function that performs the semantic analysis of the AST).
- Makefile (the makefile for compilation). Usage: "make" (built cl) or "make clean" (remove all the auxiliary files for compilation).
- codegen.cc and codegen.hh (with the recursive function that generates the Indri code).

This compiler is developed using the Purdue Compiler-Compiler Tool Set (PCCTS) [37]. If one is not familiar with this application for language translation, it is recommended to have a glance to the reference guide [43].

The file "cl.g" contains the grammar definitions and AST construction. The AST is the abstract syntax tree. The AST is defined to later analyze the structure semantically, and if everything is correct, generate the output code (Indri query language).

The file "semantic.cc" contains the semantic analysis over the AST tree constructed during the lexical analysis.

The file "codegen.cc" contains the code generation definitions. The main function of this file is the recursive function, which browses the constructed tree recursively and generates the Indri code recursively as well. A piece of code which illustrates how it works is showed here:

Listing 4.9: A section of the recursive code generator function

```
string CodeGenInstruction(AST * a, string info = "") {
  string code = "";
  string c1 = "";
  string c2 = "";

  string c;
  offsetauxspace = 0;
  if (!a) {return c; }
    ...
  } else if (a->kind == "and") {
     c1 = CodeGenInstruction(a->down);
     c2 = CodeGenInstruction(a->down->right);
     code += "#band(" + c1 + c2 + ")";
  }
```

This code structure allows modifications under demand. This means that if the specification of PLQL is modified again, it will be easy to readapt these changes.

Operators priority and hierarchy

The priority order of the operators is depicted as follows. In case of the same priority left-associativity is assumed.

- Minimum priority:
  - AND
  - =,<,>
  - *,/
  - NOT,–(unary)

- Maximum priority:
  - (), [], .

Listing 4.10: The section of the grammar which defines priorities and hierarchy

```
l_expression: expression (POINT! expression)*;
expression: expr1 ( ( AND^ | OR^ ) expr1)* ;
expr1: expr2 ( ( PT^ | LT^ | EQ^ )  expr2 )* ;
expr2: expr3 ( ( PLUS^ | MINUS^ ) expr3 )* ;
expr3: expsimple ( ( MUL^ | DIV^ ) expsimple )* ;
```

As was mentioned in the section above, the syntax of the XML structured expressions is checked during the lexical analysis. It was decided to be checked here because these structures are not variable and easy to check at this point. If the structure is not valid, it will return a syntax error.

Listing 4.11: The section of the grammar which defines priorities and hierarchy

```
lom:
        GENERAL DOT^ lgeneral     |     //1 General
        LIFECYCLE DOT^ llifecycle |     //2 Life Cycle
 ...
lgeneral:
        IDENTIFIER DOT^ lgid      |     //1.1 Identifier
        TITLE  (DOT^ langstring)* |     //1.2 Title = "LangString"
 ..
lgid:
        CATALOG |       //1.1.1 3.1.1 7.2.1.1 Title = "LangString"
 ..
```

4.4.2 DB interaction with PLQL results

The command line application provided by the Lemur Toolkit "IndriRunQuery" is not able to give the information required by the specification defined by the Acknowledge platform. It was necessary to develop an application which interacts with the index and retrieves information more specifically.

The reference [47] provides the specification of the results. There are different levels of specification. For the development of this application, the Level 3 specification was selected.

The four levels are defined incrementally as follows:

- Level 0: the number of results must be returned.
- Level 1: the results must also contain the URLs for the learning objects, which can then be retrieved by the client. If ranking is available server-side, the results are returned in an ordered way, so that the"best" results are presented first.
- Level 2: meta-data information must be added to the result set. We do not express limits on what is to be returned. However, we expect at least the learning object title, author and language to be included.
- Level 3: a numeric ranking value is added for each result, as well as an identifier for the ranking method that has been used if it is available.

Level 3 was chosen because it was considered that if the search engine returns results depending on a ranking value and position, it has no sense to throw out this information.

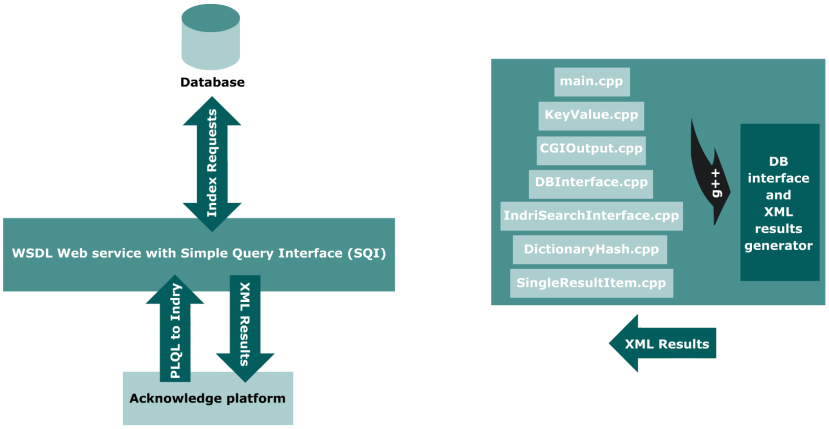

Figure 4.7: Structure of application which interacts with the index and parses the results

The structure of this application is similar to the structure of the CGI. The explanation of this similarity is because both applications interact with the index and retrieve the same information. Another explanation is because normally reusing software components enhances the speed to develop an application.

The CGI returns a Web page which contains the results. This application returns the required XML structure instead of the Web page.

The information per item that it shows is the same:

- The ranking position
- The ranking score
- The document identifier
- The document language
- The document title
- The document URL
- The document timestamp when it was retrieved

4.4.3 The Simple Query Interface

The Simple Query Interface does not define just the headers of the functions that are available to interact with the server. SQI defines an explicit WSDL file (WSDL [21]). The explicit file is provided to avoid using WSDL files than can be generated automatically by programming languages.

WSDL provides Web services over the Internet. A client program connecting to a Web service can read the WSDL to determine what functions are available on the server. Any special data-types used are embedded in the WSDL file in the form of an XML schema. The client can then use SOAP to actually call one of the functions listed in the WSDL.

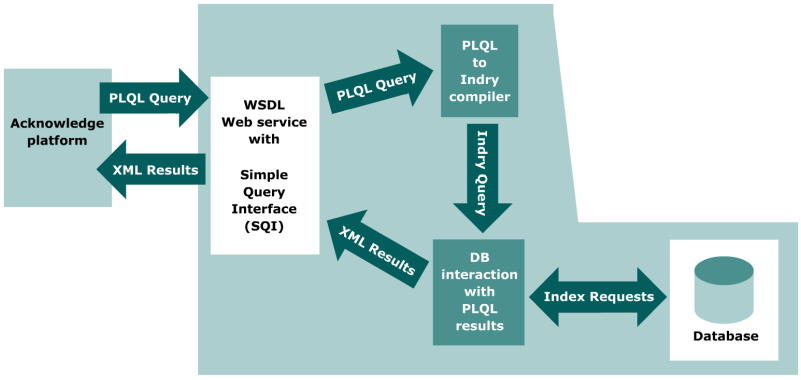

Figure 4.8: An overview of the Web service with the auxiliary applications

The Java Development Kit (JDK) version 1.6 can generate, if it is necessary, WSDL, but this automation adds legacy dependency with Java. This means that if the server uses a generated WSDL by Java, the client should be implemented in Java as well.

The implementation of SQI is a Web service. This means that is necessary to install a Web Services server to allow the interaction. The server installed for this approach is the Apache Tomcat server 5.5.x [24]. It was necessary to ask for authorization for the installation of this server at the United Nation University. To allow the Apache Tomcat server to run, it is necessary to install the Java Development Kit in the server, and re-define the system variables (PATH, CLASS PATH and so on).

WSDL development

From the point of view of a developer, the classic steps [25] for debugging a Web service are:

1 Code the implementation class

2 Compile the implementation class

3 Use wsgen to generate the artifacts required to deploy the service

4 Package the files into a WAR file

5 Deploy the WAR file. The tie classes (which are used to communicate with clients) are generated by the Application Server during deployment

6 Code the client class

7 Use wsimport to generate and compile the stub files

8 Compile the client class

9 Run the client

The figure 4.9 describes the interaction of a generic client with a JAX Web Service. The endpoint implementation class and the wsgen tool is used to generate the Web service artifacts

Figure 4.9: illustrates how JAX-WS technology manages communication between a Web service and client.

and the stubs that connect a Web service client to the JAX-WS runtime. The steps described above are tedious for a developer. The Java JDK version 1.6 allows a faster prototyping which makes the development easer. Nevertheless, once the application is finished the previous steps are necessary for generating the package to install the Web Service in the Apache Tomcat server, but the code inside the Web methods is the same.

The reference [42] describes the fast prototyping that JDK 1.6 allows. This method has a simplified deployment using different technologies that the JDK 1.6 provides:

- Endpoint API and light-weight HTTP Server.
- The Java API for XML-Based Web Services (JAXWS) Tools wsimport and wsgen are used for the WSDL deploy.
- Java Architecture for XML Binding (JAXB 2.0 also part of JDK 1.6) allows Java developers to map Java classes to XML representations such as WSDL.
- Streaming API for XML (StAX) is an application programming interface (API) to read and write XML documents in the Java programming language.
- The SOAP with Attachments API for Java (SAAJ 1.3) provides a standard way to send XML documents over the Internet from the Java platform.

The Web Services Description Language file is generated automatically from the @WebService annotation. This annotation tells JAXWS that it is a Web Service endpoint.

Listing 4.12: Plqlws.java represents an example of the WebMethod annotation

```
package server;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.xml.ws.Endpoint;
import java.io.*;
import java.util.StringTokenizer;

@WebService
public class Plqlws {

    @WebMethod
    public String synchronousQuery(String query) {
      //code
    }
}
```

These two steps build and publish this endpoint:

- Run apt to compile and generate required wrapper classes

  > apt -d sample server/Plqlws.java

- Publish

  > java -cp sample server.Plqlws

The command line listed for publishing will deploy the WSDL. This command line publishes the WSDL with the light-weight HTTP server available in the Java JDK 1.6. The published WSDL can be accessed at http://localhost:8080/Plqlws?wsdl. There is no need to provide a deployment descriptor, starting a container and so on. JAXWAS does all the steps internally. This is an extremely fast prototyping.

> wsimport -p client -keep http://localhost:8080/plqlws?wsdl

A client based on proxy is needed to test the Web Services. wsimport generates and compiles automatically some auxiliary classes.

- server/Plqlws.java – Service Endpoint Interface or SEI

- PlqlwsService – Generated Service, instantiate it to get the proxy

The client invokes the endpoint Web methods to test the Web Service.

Listing 4.13: PlqlwsApp.java is the source code of a test client

```
package client;
class PlqlwsApp {

  public static void main(String args[]){
      /**
       * Instantiate the generated Service
       */
      PlqlwsService service = new PlqlwsService();
      /**
        * Get the port using port getter method generated in CaculatorService
        */
      Plqlws plqlwsProxy = service.getPlqlwsPort();
      /**
        * Invoke the remote method
        */
      String answer = plqlwsProxy.synchronousQuery(args[0]);
      System.out.println(answer);
  }
}
```

These steps will compile and run the client code 4.13 with an instance of a synchronous query.

- javac -cp . PlqlwsApp.java
- mv PlqlwsApp.class client/
- java -cp . client.PlqlwsApp “elections and lom.general.title = ’Sarah Palin’ and lom.general.language = ’English’ ”

The Web service will return the result of this query. This result will be shown on the standard output by the client.

Chapter 5

Experimental evaluation

The search engine customization cannot be evaluated, because it is an adaptation of a software. The crawler cannot be evaluated neither, because it is a specific crawler that handles lists of topics and news, with a predefined format. The application that will be evaluated and compared with the state of art, is the content extractor.

5.1 Content extraction experiments

5.1.1 Data set used

The proposed method was evaluated on a data set previously used in state of the art content extraction [27]. 14 different datasets (see table 5.1) were gathered from the Web. Then a golden standard was created for every HTML page by manually selecting the main content of every Web page. Most Web pages contain news items, although some also contain encyclopedia articles (wiki) or help-pages (manual). The Websites are written in different languages: English (en), Italian (it) and German (de).

5.1.2 Evaluation

As described in section 4.2 the aim was to build a method that can successfully label text in a Web page as "main content" or "garbage". This section describes how the method proposed in this project was evaluated.

Although the task can be conceptually described as labeling text in a Web page, in reality most systems that perform this extraction task take as input the HTML source code of the Web page and return a file containing the cleaned text. The ground truth data (described above) is also stored in this cleaned text format. To evaluate the developed method, a metric is needed that compares how "similar" the automatic output is compared to the manually generated output. More formally, let t_{manual} be the main content text that was manually created, and let $t_{automatic}$ be the content text that was automatically created. Then, different metrics have been proposed that measure the similarity between the two files. In this section two evaluation metrics are used: longest common substring and longest common subsequence.

The longest common substring metric (LCString) [18] finds the longest continuous string that appears both in the automatic output and the manual output. For example, the LCString of the strings "the dog jumps over the brown fox" and "the fox jumps over the brown dog" is "jumps over the brown" (of length 20). This metric is usefull since it focuses on the longest

Website	URL	# of pages	Language
bbc	news.bbc.co.uk	1000	en
chip	www.chip.de	361	de
economist	www.economist.com	250	en
espresso	espresso.replubblica.it	139	it
golem	golem.de	1000	de
heise	www.heise.de	1000	de
manual	different	65	en,de
repubblica	www.replubbica.it	1000	it
slashdot	slashdot.org	364	en
spiegel	www.spiegel.de	1000	de
telepolis	www.telepolis.de	1000	de
wiki	de.wikipedia.org	1000	en
yahoo	news.yahoo.com	1000	en
zdf	www.heute.de	422	de

Table 5.1: Datasets used for evaluation, showing the name of the dataset, the url, the number of pages used in this evaluation and the language of the pages.

continuous string, thus highly penalizing any discarded words (or punctuation marks) in the center of the text, which could possibly carry high semantic value (e.g., imagine that at some point in the text the word "not" would not be extracted by the automatic method, thus possibly changing the entire meaning of a sentence). On the other hand, a major disadvantage of this method is that it treats all symbols identically, e.g., that discarding a space in the center of the text could possibly half the LCString, thus halving the score on a certain document.

The longest common subsequence metric (LCSequence) [30] finds the longest sequence of characters that appear in that order in both the automatic output and the manual output. For example, the LCSequence of the strings "the dog jumps over the brown fox" and "the fox jumps over the brown dog" is "the jumps over the brown " (of length 23). Notice how LCString is always a substring of LCSequence. This metric is less strict in that it assigns only a modest penalty to missing characters.

Two character based algorithms were used (in contrast to for instance [26]), because word based algorithms can be harder to implement (because the characters on a Web page need to be correctly split into words and this word splitting is not relevant for this task).

For both metrics, given the length of the longest common string or sequence s_{max}, the familiar information retrieval metrics of precision, recall and F1-measure, were calculated as follows:

$$precision = \frac{length(s_{max})}{length(t_{manual})}$$

$$recall = \frac{length(s_{max})}{length(t_{automatic})}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

	Baseline		Content extraction	
Website	LCString	LCSequence	LCString	LCSequence
bbc	60.16	61.52	96.32	97.17
chip	6.09	19.25	26.33	78.09
economist	30.91	66.85	45.48	91.88
espresso	69.04	77.32	82.10	89.25
golem	8.28	50.92	15.78	92.17
heise	46.57	61.47	72.28	96.82
manual	11.72	40.72	20.64	53.94
repubblica	14.77	71.95	21.57	90.74
slashdot	10.96	11.61	29.93	53.85
spiegel	8.86	55.86	13.39	86.84
telepolis	5.25	83.14	5.83	89.15
wiki	70.49	81.87	71.96	78.67
yahoo	34.73	65.75	52.36	94.58
zdf	14.39	67.50	25.13	82.93

Table 5.2: Average results for the baseline and automatic extraction method for the LCString and LCSequence evaluation metrics, given in % of F1-measure.

5.1.3 Results

A first set of experiments was performed where the raw output of the proposed method was compared with the manual extracted texts. This method yielded results that were lower than expected, caused by superfluous spaces in the manual extracted texts. These spaces do not play any role in the HTML source code (apart from splitting words) and it was felt that they can be ignored. All results reported ignore any spaces in both automatic and manual extracted texts.

This approach was compared with a baseline method that extracts *all* texts from a given Web page, removing markup information, but does not perform any content selection. Table 5.2 shows that the method proposed here results in a significant increase over the baseline. This can be explained by the fact that although the baseline achieves near perfect recall (since the main content will certainly be part of the extracted text), it suffers from a low precision (since it extracted all texts). This method generally also achieves a high recall, but also a high precision because of the dynamic content selection methods.

Table 5.2 shows that for some datasets (bbc, yahoo, heise) our method achieves near perfect F1-measure. For many other datasets (spiegel, economist, repubblica, espresso, telepolis, golem) this method achieves an F1-measure of 85% or more, which is certainly satisfying given the complexity of the task.

The proposed method achieves disappointing results on only two datasets, slashdot and manual. The slashdot dataset achieved a LCSequence F1-measure of 53.85%. The reason for this result can easily be explained by Figure 5.1.

In Figure 5.1, the string representing the main content (as indicated) is only a small subset of the entire text on the Web page. Closer inspection reveals that the remaining text appearing after this content is comments. Indeed, a page on the slashdot Website typically consists of a small text describing some newsworthy facts and many comments. Often the

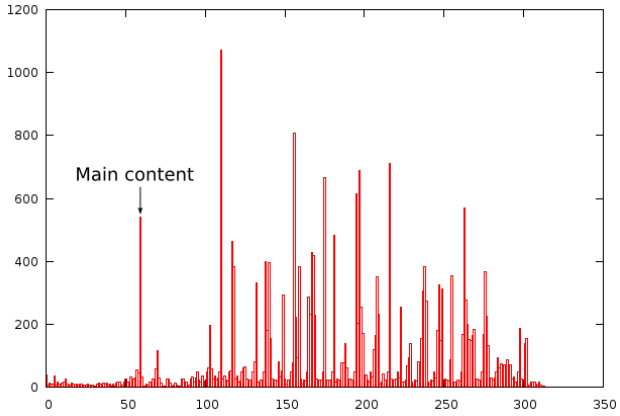

Figure 5.1: Document density graph for a sample of the slashdot corpus.

length of an individual comment is larger than the length of the news item. It is thus easy to understand how our method, relying heavily on the text density, fails for this kind of documents. Although one could also argue that the comments should also be considered part of the content.

The reason which was found for the low performance on the manual dataset is not due to the automatic extraction method, but due to the golden standard for that dataset. For many files, the golden standard is not correct and misses large parts or all of the main content for a particular Website. The method used was not aware of the reason for these errors, but note that the low result on this dataset in [27] can probably also be attributed to this incorrect golden standard.

It is hard to compare our results with existing work on this task. Only one work that also employs this dataset [27] was found. The authors evaluate their methods using the LCSequence metric, using a word based algorithm, and not a character based algorithm as reported here. Although this might lead to slightly different results, a comparison is still feasible.

Table 5.3 compares this content extraction method to the currently best method (Adapted Content Code Blurring) reported in [27]. The baseline results reported here are also compared with the baseline result of this method. Surprisingly, it was noticed that for some datasets (e.g., espresso, respectively 62.4% and 77.32%) there is a large difference between the two baseline systems, although for other systems this difference is only very modest (e.g., golem, with respectively 50.2% and 50.92%). This difference can only partially be explained by different evaluation metrics, but must also be caused by a weaker baseline in [27], possibly caused by a less robust HTML parser or by accidentally adding non-textual content (e.g., JavaScripts) to the baseline result.

When comparing this content extraction method to the ACCB method, the conclusion is that it achieves significantly ($> 1\%$) higher results for 9 datasets (bbc, chip, economist, espresso, heise, manual, slashdot, wiki and yahoo), comparable results for 1 dataset (spiegel) and significantly ($> 1\%$) lower results for 4 datasets (golem, repubblica, telepolis and zdf). These results might indicate that this method performs better on average. This belief is further strengthened by the fact that the method exposed seems more robust, where this method achieves less then 75% F1-measure on only two (slashdot and manual) datasets,

	Baseline		Content extraction	
Website	Here	In [27]	Density	ACCB
bbc	61.52	59.5	97.17	92.4
chip	19.25	17.3	78.09	70.3
economist	66.85	61.3	91.88	89.0
espresso	77.32	62.4	89.25	87.5
golem	50.92	50.2	92.17	95.9
heise	61.47	57.5	96.82	91.6
manual	40.72	37.1	53.94	41.9
repubblica	71.95	70.4	90.74	96.8
slashdot	11.61	10.6	53.85	17.7
spiegel	55.86	54.9	86.84	86.1
telepolis	83.14	85.8	89.15	90.8
wiki	81.87	82.3	78.67	68.2
yahoo	65.75	58.2	94.58	73.2
zdf	67.50	51.4	82.93	92.9

Table 5.3: Average LCSequence F1-measure (in %) results of the density method reported here and the ACCB method reported in [27].

compared to 5 datasets (chip, manual, slashdot, wiki and yahoo) in [27].

The slightly better results can be largely due to the method of creating the density vector employed here. The choice to use only structural tags, and ignore other mark-up tags means that this vector reflects closer the real structure of the page, and that text elements that are structurally closer together are also closer together in this vector.

Table 5.4 shows the processing times needed for the different dataset on a 1.66Ghz Intel cpu. These processing times are also compared with the times reported in [27]. This comparison might not be very accurate since we are not aware of the speed of the computer that generated these results, but still indicates that the proposed method performs at least comparable to this state of the art method.

Website	ACCB	density
bbc	1.0	0.361
chip	8.0	0.314
economist	15.0	0.294
espresso	16.0	0.317
golem	9.0	0.337
heise	12.0	0.341
manual	20.0	0.353
repubblica	14.0	0.355
slashdot	13.0	0.353
spiegel	15.0	0.351
telepolis	52.0	0.377
wiki	28.0	0.377
yahoo	13.0	0.315
zdf	1.0	0.318

Table 5.4: Average processing time (in s/Mb) for our density extraction method and the ACCB method reported in [27].

Chapter 6

Agreement between results and objectives

6.1 Content extraction

This thesis presented a novel method for content extraction from Web pages, sometimes also referred to as Web page cleaning. This method relies on a single heuristic that the main content of a HTML page has a high density of text characters and low density of structural code. It has been shown that this method performs comparably to, or better than state of the art methods. Furthermore, it has the following valuable properties :

1 it is simple, and easy to implement

2 it is fast, processing up to 3.4Mb of HTML code per second

3 it runs robustly on dirty or not well-formed HTML code and

4 it does not use the content of the text itself and is thus language-independent

Furthermore, this work proposed to make a distinction between structural and non-structural markup-tags. A comparison with another state of the art method has shown that making this distinction improves results and allows for more robust methods.

The method as described here works on the raw density values. It might be advantageous however to create a more abstract representation which potentially allows more powerfull algorithms. For instance, one could approximate a smooth function such as a polynomial function to the density values using a least squares method. Then the maxima, minima, first and second order derivatives could be used to decide on the high density region.

Although this project has shown that text density is an important heuristic when extracting content from Web pages, it is naive to expect that all Web pages can be successfully cleaned using this heuristic alone. Therefore, in the future more powerfull methods will have to be developed. It can be guessed that two research directions could prove to be promising: Firstly, methods that perform an analysis of an entire Website (as compared to a single Web page) could discover the common structure and texts of all pages of a certain Website. It can be expected that this common structure and texts do not belong to the main content for a particular Website. Secondly, one could perform an analysis of the text in a Web page, and

learn that certain words do (e.g., “written by”, “author”) or do not (e.g., “close this window”, “comments”) belong to the main content.

Chapter 7

Economic analysis and comparison with alternative analyses

7.1 Temporal estimation

The temporal estimation is defined over the goals of the project defined in the second chapter.

- Training: On this stage, the basic knowledge of search engines and information retrieval should be acquired and the methodology of the other groups which interact in the same platform.
- Research: The research at this stage should investigate technologies that could adapt to the development of the project and select the most appropriate for the goals defined.
- There are different goals which have common temporal stages. These are the crawler, the noise reducer, the Lemur customization and the different interfaces (the Web service, and the cgi). These are common stages:
  - Application design: The application design should be based on the features that need to be implemented, the performance of the selected technologies and the user interface.
  - Development: The application should be developed following the design decided in the previous stage.
  - Quality control: During the development and after it, several tests of performance and accuracy should be made.
  - Documentation: Finally, a documentation should be done which describes the functional prototype.

The initial planning of the project defined two important goals that were based on development. The first was the delivery of the report. The delivery of the report was scheduled before the end of 2008. Therefore, at this date the solution should be designed, tested and the feasibility of the project should be demonstrated. As it was expected, the date of delivery was as expected.

Without the second goal, the first goal could not be possible. This second goal was to complete the solution that was required, that is, the data set builder and the search engine

customization (it includes the Web service, the CGI, and the configuration files integrated with the categorization utilities). This set of applications were planned to be demonstrated on the Acknowledge final event, the 24th of November of 2008 in Gent. We succeeded in reaching this deadline and the application was exposed and demostrated as expected.

One could conclude that the development of the project was set largely according to the planning.

In the next section, the tasks are broken down at the same time that the weight of every part of the project has had in actual fact.

7.2 Timeline

The time needed for the entire project was 6 months (from 01/07/2009 until 19/12/2007). This time was divided in three stages: training, development and documentation. Figure 7.1 shows a Gantt chart which defines an overview of the different stages based on the objectives of the project during the working period.

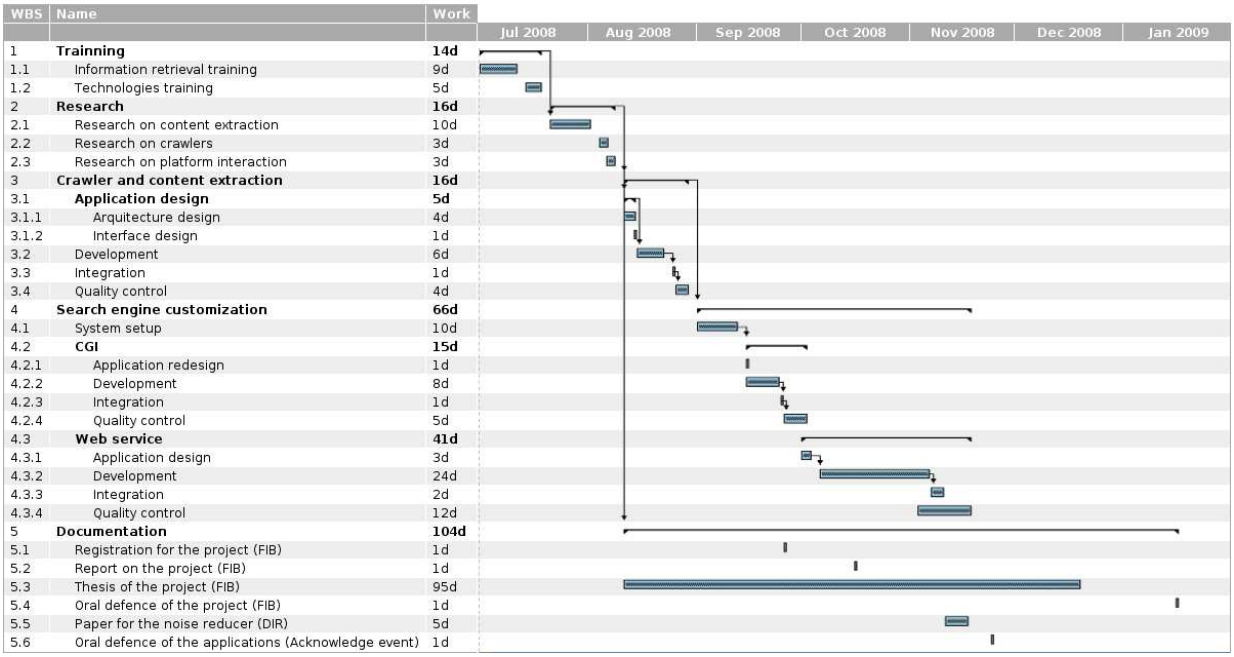

Figure 7.1: Gantt Chart of the project timeline

Given the variety of technologies that interact in the demonstrators, the development has had to be subdivided into multiple stages.

A monitoring of the development of the project has been conducted weekly, through meetings, presentations and demonstrations. The project has been developed with the interaction of other developers to allow the integration in the different platforms. During the last month of the project, an extra monitoring has been conducted weekly for supervising the writings of this document.

As it was mentioned in the section above, all the goals were approached on time. Furthermore, an unexpected goal was approached. This goal is the acceptation of a paper related

WBS	Name	Start	Finish	Work	Complete
1	Trainning	Jul 1	Jul 18	14d	
1.1	Information retrieval training	Jul 1	Jul 11	9d	100%
1.2	Technologies training	Jul 14	Jul 18	5d	100%
2	Research	Jul 21	Aug 8	16d	
2.1	Research on content extraction	Jul 21	Aug 1	10d	100%
2.2	Research on crawlers	Aug 4	Aug 6	3d	100%
2.3	Research on platform interaction	Aug 6	Aug 8	3d	100%
3	Crawler and content extraction	Aug 11	Aug 29	16d	
3.1	Application design	Aug 11	Aug 14	5d	
3.1.1	Arquitecture design	Aug 11	Aug 14	4d	100%
3.1.2	Interface design	Aug 14	Aug 14	1d	100%
3.2	Development	Aug 15	Aug 22	6d	100%
3.3	Integration	Aug 25	Aug 25	1d	100%
3.4	Quality control	Aug 26	Aug 29	4d	100%
4	Search engine customization	Sep 1	Nov 18	66d	
4.1	System setup	Sep 1	Sep 12	10d	100%
4.2	CGI	Sep 15	Oct 2	15d	
4.2.1	Application redesign	Sep 15	Sep 15	1d	100%
4.2.2	Development	Sep 15	Sep 24	8d	100%
4.2.3	Integration	Sep 25	Sep 25	1d	100%
4.2.4	Quality control	Sep 26	Oct 2	5d	100%
4.3	Web service	Oct 1	Nov 18	41d	
4.3.1	Application design	Oct 1	Oct 3	3d	100%
4.3.2	Development	Oct 6	Nov 6	24d	100%
4.3.3	Integration	Nov 7	Nov 10	2d	100%
4.3.4	Quality control	Nov 3	Nov 18	12d	100%
5	Documentation	Aug 11	Jan 16	104d	
5.1	Registration for the project (FIB)	Sep 26	Sep 26	1d	100%
5.2	Report on the project (FIB)	Oct 16	Oct 16	1d	100%
5.3	Thesis of the project (FIB)	Aug 11	Dec 19	95d	100%
5.4	Oral defence of the project (FIB)	Jan 16	Jan 16	1d	0%
5.5	Paper for the noise reducer (DIR)	Nov 11	Nov 17	5d	100%
5.6	Oral defence of the applications (Acknowledge event)	Nov 24	Nov 24	1d	100%

Table 7.1: Project schedule

Economic cost	
Concept	Cost (euros)
Human resources	9600.00
Computer equipment	133.60
Systems and programs	0.0
Furniture and office	15.00
Traveling expenses	30.40
TOTAL	9779.00

Table 7.2: Description of the economic cost

with this project in the DIR 2009 Workshop. This paper is the compilation of all the section of this project related with the noise reducer.

DIR 2009 is the 9th Dutch-Belgian Information Retrieval Workshop. DIR 2009 will be celebrated during the second and third of February, 2009 at Vrijhof, University of Twente, Enschede, The Netherlands. The primary aim of the DIR workshops is to provide an international meeting place where researchers from the domain of information retrieval and related disciplines, can exchange information and present new research developments. A copy of this paper is included in the appendix.

7.3 Economic analysis

The different elements involved in this development are described in detail for calculating the approximate cost of this project, from human resources to the used furniture. The following table represents the cost of each item individually:

The calculations of this table are described with more detail in the followoing listing:

Human resources In this project a student was hired with a scholarship and an agreement between the Department of Computer Science of the Katholieke Universiteit Leuven and the Barcelona School of Informatics. The scholarship was 1600 euros per month, during a period of 6 months.

Computer equipment The computer used for this project was a Optiplex gx620. This computer was acquired the 6th of October, 2006. The price was 950 euros. Moreover, bearing in mind that the computer has an age of two years, the table 7.3 rates the devaluation. These calculations are not precise, because the devaluation coefficient is calculated according to the Spanish regulation and the project was developed in Belgium. The Spanish regulation defines a depreciation of equipment over the corporation tax. This regulation establishes a maximum lineal coefficient of 25% in a maximum period of 8 years, over the equipment for processing information. The devaluation was calculated for the third year, because this is the closest interval of time according to the usage of the computer.

Software This project was developed using Free Software. The operative system used for development was Debian GNU/Linux, the free desktop virtualization software application was VMware.

Economic cost		
Year	Recovery (euros)	Value (euros)
1st	0.25 * 950 = 237.5	712.5
2nd	0.25 * 712.5 = 178.12	524.37
3rd	0.25 * 524.37 = 133.60	390.77
	TOTAL	133.60

Table 7.3: Description of cost of the computer equipment

Furniture The furniture is a desk and a swivel chair. The regulation establishes a maximum lineal coefficient of 10% in a maximum period of 20 years, over the office furniture. The desk and the swivel chair have a value of 300 euros. The usage value is 15 euros (300*0.05).

Traveling expenses The traveling expenses between Spain and Belgium were covered by the employee. During the Workshops and meetings, the employee was traveling as companion in a car that was covered under another budget. The traveling expense that was covered by this project was one between Leuven and Brugge by train with a value of 30.40 euros.

Chapter 8

Conclusion and contributions

This thesis presented a data set builder and a search engine customization. This section discusses the contributions and conclusions of this thesis. This Master Thesis was executed at the university with a budget from private enterprises. This project was a set of tools that gives a solution for specified needs, in the research area of Web Mining.

8.1 Contributions

The main contributions of this thesis are the following:

- Web crawler that browses the World Wide Web in a specific methodical, automated manner.
- Fast, accurate and language independent context extraction.
- Information indexed by the use of the meta-information.
- User web interface to interact with the search engine using the meta-information.
- Integration of Lemur with the Acknowledge platform.

8.2 Conclusions

The achievements and conclusions of this work are:

The crawler browses the World Wide Web in a specific manner. There are more generic crawlers that browse Internet in a more generic manner depending on some parameters that can be set by the user, as for example Heritrix[1]. Nevertheless, the solution required was more specific.

The local copy of the Web pages crawled cannot be indexed directly by the information retrieval platform because this information contains noise. The algorithms which remove this noise are not fast enough for the solution required. This research reaches state of art of Web cleaning, the evaluations showed the same accuracy of the current research at least four times faster.

[1] http://crawler.archive.org/

The Lemur Toolkit is the most extended information retrieval platform available in open source. It means that it is the solution that provides more usability and extensibility for this work.

The query language interface provides the interaction with the Acknowledge platform. The Acknowledge platform provides a public specification but does not provide an implementation. The query language translation required for this integration defines a grammar of the language to be accepted following the provided specification.

Typically, when there are goals to approach, the final result is to approach the goals or to do not approach all of them. This project was unusual, it approached a non-defined goal as a side effect from a defined goal. This new goal was the acceptance of a paper for a workshop. The paper that was submitted to the workshop describes the research of cleaning made in this project.

8.3 Future Research

This section discusses several directions of future work stemmed from this thesis. This document is the documentation of the Master Thesis and the documentation of the application developed for a future adaptation. All the code developed in this project and the documentation itself will be uploaded to an internal intranet of the K.U. Leuven.

The applications developed in this project can be improved and adapted to new approaches. The aim of this documentation is making these approaches easier. These approaches can be the following:

- The crawler can be adapted to browse other lists of news, in different languages since the cleaner is language-independent.
- The grammar used by the Acknowledge platform integration can be improved and adapted to other applications of the Acknowledge platform.
- The meta-information plugged into the customized Lemur could be extended and incorporate information from large ontologies.
- The Web application that provides an easy interface to query the index depending on the meta-information can be adapted to meta-information defined under demand.

Appendix A

Copy of the developed software

Appendix B

Paper

Bibliography

[1] A9 innovations in search technologies. `http://www.a9.com/`.

[2] The apache software foundation.

[3] Ask.com search engine - better web search. `http://www.ask.com/`.

[4] Baidu. `http://www.baidu.com/`.

[5] Bézier spline from wikipedia, the free encyclopedia.

[6] The dublin core metadata initiativa. http://dublincore.org/.

[7] Google. `http://www.google.com`.

[8] in.gr. `http://www.in.gr/`.

[9] Lemur toolkit and indri search engine documentation.

[10] Msn/live search. `http://www.live.com/`.

[11] Sourceforge open source software repository.

[12] Sourceforge.net: Open source software.

[13] Vmware (nyse: Vmw) is the global leader in virtualization solutions from the desktop to the data center.

[14] Yahoo! `http://www.yahoo.com`.

[15] 2nd lor interoperability / sqi workshop. Lugano, Switzerland, June 21st, 2004.

[16] Debian gnu/linux 4.0r5 aka “etch” is the current stable version. October 23rd, 2008.

[17] Andre Bergholz, Gerhard Paass, Frank Reichartz, Siehyun Strobel, Marie-Francine Moens, and Brian Witten. Detecting known and new salting tricks in unwanted emails. In *CEAS 2008: Proceedings of the Fifth Conference on Email and Anti-Spam*, 2008.

[18] Paul E. Black. ”longest common substring”. In *Dictionary of Algorithms and Data Structures [online]*. http://www.nist.gov/dads/HTML/longestCommonSubstring.html, 2004, retrieved 17/11/2008.

[19] Erik Duval. The common sqi wsdl binding (cswb).

[20] Erik Duval. Ieee standard for learning object metadata. http://ltsc.ieee.org/wg12/, 2004.

[21] Microsoft Erik Christensen, IBM Research Francisco Curbera, Microsoft Greg Meredith, and IBM Research Sanjiva Weerawarana. Web services description language (wsdl) 1.1. 2001.

[22] Aidan Finn, Nicholas Kushmerick, and Barry Smyth. Fact or fiction: Content classification for digital libraries. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.

[23] The Apache Software Foundation. Apache http server project. http://httpd.apache.org/.

[24] The Apache Software Foundation. Apache tomcat. http://tomcat.apache.org/.

[25] The Apache Software Foundation. Creating a simple web service and client with jax-ws. In *The Java Web Services Tutorial*. http://java.sun.com/webservices/docs/2.0/tutorial/doc/JAXWS3.html.

[26] T. Gottron. Evaluating content extraction on HTML documents. In *ITA07: Proceedings of the 2nd International Conference on Internet Technologies and Applications*, pages 123–132, 2007.

[27] Thomas Gottron. Content code blurring: A new approach to content extraction. *International Workshop on Database and Expert Systems Applications*, 0:29–33, 2008.

[28] Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. Dom-based content extraction of html documents. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 207–214, New York, NY, USA, 2003. ACM.

[29] Wei Han, David Buttler, and Calton Pu. Wrapping web data into xml. *SIGMOD Record*, 30(3):33–38, 2001.

[30] Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675, 1977.

[31] ES) Jos M. Martnez (UAM-EPS-GTI. MPEG-7 Overview (version 10). Palma de Mallorca, October 2004.

[32] Donald Knuth. Section 2.3. In *The art of computer programming vol 1. Fundamental Algorithms*, page 318348. Addison-Wesley, 1997.

[33] Bernhard Krüpl, Marcus Herzog, and Wolfgang Gatterbauer. Using visual cues for extraction of tabular data from arbitrary html documents. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1000–1001, New York, NY, USA, 2005. ACM.

[34] Matt Kruse. Calendar popup javascript.

[35] Fred H. Lesh. Multi-dimensional least-square polynomial curve fitting. Communications of ACM.

[36] Shian-Hua Lin and Jan-Ming Ho. Discovering informative content blocks from web documents. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 588–593, New York, NY, USA, 2002. ACM.

[37] Inc. Maintained by Tom Moog, Polhode. Pccts resources and "notes for new users of pccts".

[38] Constantine Mantratzis, Mehmet Orgun, and Steve Cassidy. Separating xhtml content from navigation clutter using dom-structure block analysis. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 145–147, New York, NY, USA, 2005. ACM.

[39] The University of Glasgow. Terabyte retrirver.

[40] The University of Massachusetts Amherst. The lemur toolkit for language modeling and information retrieval.

[41] The University of Waterloo. Wumpus search engine.

[42] Vivek Pandey. Vivek pandey's blog at sun microsystems. In *"Webservices in JDK 6"*. http://weblogs.java.net/blog/vivekp/archive/2006/12/webservices_in.html, 2006.

[43] Terence John Parr. In *"Language Translation Using PCCTS and C++"*. Automata Publishing Company, San Jose, CA 95129 .

[44] David Pinto, Michael Branstein, Ryan Coleman, W. Bruce Croft, Matthew King, Wei Li, and Xing Wei. Quasm: a system for question answering using semi-structured data. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 46–55, New York, NY, USA, 2002. ACM.

[45] The Lemur Project. Indri query language quick reference. http://www.Lemurproject.org/Lemur/IndriQueryLanguage.php.

[46] Inc. Sun Microsystems. Java se technologies at a glance.

[47] Stefaan Ternier, David Massart, Alessandro Campi, Sam Guinea, Stefano Ceri, and Erik Duval. Interoperability for searching learning object repositories, the prolearn query language.

[48] Tim Weninger and William H. Hsu. Text extraction from the web via text-to-tag ratio. *Database and Expert Systems Applications, International Workshop on*, 0:23–28, 2008.

[49] Zobel and Moffat. Managing gigabytes fast search engine.