FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Search in Glintt HS Solutions

João Vitor Barros Monteiro Dias

Integrated Masters in Informatics and Computing Engineering

Advisor: Prof. Dr. Francisco José de Oliveira Restivo

June, 2010

© João Vitor Barros Monteiro Dias, 2010

Search in Glintt HS Solutions

João Vitor Barros Monteiro Dias

Integrated Masters in Informatics and Computing Engineering

Approved in public trial by the jury:

President: Prof. Ademar Manuel Teixeira de Aguiar

External Board Member: Prof. Dr. Álvaro Manuel Reis da Rocha

Advisor: Prof. Dr. Francisco José de Oliveira Restivo

Abstract

With the incredibly fast increase in hard drive capacity, the amount of information contained in a single personal computer, in databases or independent networks has equally increased a great deal. These increases led to huge amounts of information, in the form of documents, multimedia files, digital photos, etc, gathered in a single hard drive or database, and consequentially in a network, thus increasing the difficulty in the task of finding information, and even harder to find a specific piece of information.

It became very time consuming to search for a specific file, document or record in the huge sea of information available. This led to the creation of several search engines to help in the location of files and information, significantly reducing the time spent in that task. These software applications have several areas of use, but only desktop and database environments will be addressed, more specifically, only desktop search engines, also known as local search engines, and database search engines, will be studied.

This report presents a description of what are search engines, more specifically, database and desktop search engines.

It also presents a collection of information regarding nowadays database and desktop search engines. This information addresses the features that each one grants to the users and administrators, the compatibility and compliancy of each one to the problem at hand and its established standards, as well the technologies used by said company and even the financial aspects presented by each one of them. This collection of information is then used to determine the best course of action regarding a problem of a specific company, in this case Glintt Healthcare Solutions, SA.

After the decision is made, the solution consequently developed is presented and described.

Special Thanks

Francisco José de Oliveira Restivo for accepting the advisor task and for the guidance in an academic environment;

Nuno Ribeiro for the given opportunity and guidance in an business environment;

Faculdade de Engenharia da Universidade do Porto for the opportunity to get to this point;

Nuno Cruz for enduring with me during all the time in the company, for all the bouncing ideas times and for providing helpful resources;

Rui Pinto for all the bouncing ideas times and for providing helpful resources;

Daniel Alves for all the bouncing ideas times, for providing helpful resources and for being the main agent in my company's integration process.

Contents

Figures List

Tables List

Acronyms and Symbols

1. ODBC – Oracle Database Client
2. MSSQL – Microsoft SQL
3. WPF – Windows Presentation Foundation
4. WCF –Windows Communication Foundation
5. SE – Search Engine
6. DSE – Desktop Search Engine
7. DBSE – Database Search Engine
8. HSE – Hybrid Search Engine
9. PDF – Portable Document Format
10. OS – Operative System
11. GDS – Google Desktop Search
12. CTRL - Control
13. IDE – Integrated Development Environment
14. CDS – Copernic Desktop Search
15. YDS – Yahoo Desktop Search
16. WDS – Windows Desktop Search
17. AQS – Advanced Query Syntax
18. JDBC – Java Database Connectivity
19. OSES – Oracle Secure Enterprise Search
20. ACL – Access Control List
21. GSA – Google Search Appliance
22. CMS – Content Management System
23. API – Application Programming Interface
24. SDK – Software Development Kit
25. USD – United States Dollar
26. XML – eXtensible Markup Language
27. UI – User Interface
28. SQL – Structured Query Language
29. DBMS – Database Management System
30. VB – Visual Basic

31. DDL – Data Definition Language
32. DML – Data Manipulation Language
33. XSD – XML Schema Definition
34. SOAP – Simple Object Access Protocol
35. WSDL – Web Service Definition Language
36. HTTP – Hypertext Transfer Protocol

1. Introduction

Informatics Science is a fast growing and quick evolving area. With the constant evolutions and developments revolving this subject, the appearance of new data types and new data structures became pretty common. To back up this constant evolution system, the hardware grows in power, capacity and availability, allowing for a continuous growth in the area.

By making the best out of this evolution, the general society embraces the technological advance and, with each passing day, spreads a little bit more the need for more and more evolution, more and more capacity. The demands are based in the incredible amount of information that flows from channel to channel. Enterprises, hospitals, schools, newspapers, television networks, public services and much more, create every day huge amounts of data that has to be dealt with [1][2]. Also, with each passing day, the volume in available information increases at the point where sometimes it becomes a hard task for a single person to find the little piece of it that he or she is searching for.

When looking for information on a particular subject it's not feasible to read the entire collection of files, documents or records available to find out which ones suit the given search necessities. Scroll through all the available data would be too much time consuming and would not be a bearable way of commanding production and evolution. By scrolling through only a small subset of possibilities it becomes an easier and doable task.

This situation led to the necessity for tools and mechanisms that are able to filter the available sources of information in order to provide a meaningful yet relatively short set of information data that matches the searcher needs without too much trouble [3].

1.1- The Problem

The ability to search for information is a very useful and important feature in nowadays applications. With the increase in the amount of information and the higher capacities of hard drives and databases it becomes harder and harder to find what one's looking for. Also, with a great number of different types of documents and files and with all the different structures created to store all the information, the access to it becomes even more complex.

To deal with this issue, a company, Glintt Healthcare Solutions, SA, needs a tool capable of organizing all of its systems' information in order provide a functional and quick way to find the specific information without spending too much time doing so. This tool needs to be capable of dealing with all the information structures of Glintt Healthcare Solutions, SA and it must be developed so it can be possible to apply it to a wide range of applications. It must be able to access all the given hard drives and databases and successfully extract information from them and provide an easy and transparent means of search to the end user.

1.1-1. The Company

Glintt is a big enterprise with several sectors of activity, in which it has knowledge and experience through work already developed.

It has its very own way of approaching the market and methodologies of implementing its projects. This led it to earn the loyalty of several clients.

By continuously investing in the formation of its employees and with all the strategic partnerships that it's able to establish, it strives for the innovation that allows it to add value to the projects that it implements with the partnership of its clients. It's able to effectively respond to the increasingly challenging projects that are proposed to it.

Nowadays, the business offers that Glintt presents to the market allow it to respond to its clients necessities, not only in the Vertical sector but also in the Horizontal one.

Glintt Healthcare Solutions, SA is one of the several sectors of Glintt. It already developed several applications that cover a great number of hospitals and health clinics, both private and public [4].

1.1-2. Requirements

There are several requirements that the tool must fulfill. Glintt Healthcare Solutions, SA has two database systems, Oracle Database Client (ODBC) and Microsoft SQL Server (MSSQL). The tool must be able to transparently crawl through both types of databases and successfully extract information from them.

There is also a great deal of file types, such as Word documents, PDF files, etc. The tool must be able to index all the file types existent in the enterprise and must be built in a way that it allows for the addition of new file types.

As for the indexed information, all of it must be associated to the entities managed by each application, so that it is possible to search for only a range of selected entities and it should be possible to specify the data sources for the tool to crawl and which entities to index.

Finally it must be possible to provide or get a good support for such a tool so it must be taken into account if such support exists.

1.1-3. Constraints

The application to be developed or acquired must be Microsoft compliant. Since the company works mainly over Microsoft software and operates in a Microsoft environment, the application must be fully functional in such an environment. There's no restriction to whether the application must function under a different environment set.

If an application is to be in-house developed the development must also meet some technological demands. The connection between the tool and the applications from the enterprise must be made via Web Services. The technologies to be used fall in the .NET family. These are ASP.NET, WPF, WCF, and Silverlight and C#.NET. Also, in-house modules or frameworks already developed should be incorporated in the application's development, in order to meet the company's development policy

Finally, whether a solution is developed or acquired, it must be affordable for the company. The financial overhead of the tool must be taken into account.

1.1-4. Conclusions

In order to solve the company's problem an efficient search application must be either obtained or created. A search engine that is able to index not only databases but also desktop folders serves the company's needs, provided it complies with all the restrictions and constraints presented.

If a solution is to be obtained it must fulfill some requisites:

- Be able to handle ODBC and MSSQL databases;
- Be able to index the different types of files the company requires;
- Work in a Microsoft Windows Environment;
- Be fully Microsoft compliant;
- Be able to communicate with other applications;
- Be affordable to the company's budget;
- Provide a partial search;
- Provide ways to define that data sources to be indexed;
- Have a reliable support.

If a solution is to be developed, besides all the above requirements, the application must be:

- Developed using the company's standard technologies;
- Re-utilize company's previously created modules and frameworks;
- Inherit validation and permission protocols from the environment where it is inserted.

1.2- Document Structure

Besides this introduction and all its content, which complete the first chapter of this report, there are still five more chapters in it.

The second chapter describes the state of the art related to the subject being handled in the report, search engines. Besides said description, a comparative analysis between all search engines gathered is made in order to know what course to take regarding the resolution of the company's problem.

The third chapter has a full description of an application's full conception and architecture, always bearing in mind the requirements and constraints presented.

The fourth chapter follows the conception and architecture created and explains in detail the development of a prototype based in said architecture, after defining the necessary components for a ratable prototype.

The fifth chapter describes a simulation scenario and a full execution run of the prototype previously developed.

The sixth chapter finalizes the report by presenting some future evolution for the architected application and the final conclusion of all the developing process.

2. Search Engines

There are several SE applications available in the actual software panorama. The possibility that one of them is the solution to the problem at hand is present. Nevertheless there's no way of knowing it for sure without studying those applications. In order to get a grasp of what the actual SEs allow a user to do and to match the results with all the constraints and restrictions presented by the problem that needs solving a study over the available SE must be done.

2.1- What is a Search Engine?

Search Engines are special applications that have the ability to provide its users with links to files, documents or pages related to what they want to know in order to help them find the information they're searching for. Although there may be several differences in the way that each Search Engine works, all of them must perform three basic tasks:

- Search the system, in which the Search Engine is installed, or a part of it, based on input given by the user. This input takes the form of important words called keywords;
- Keep an index of the words they find and also where to find these words;
- Allow users to search for words or combinations of words, found in that same index.

Search Engines build an index database and, maintain it with regular update, in order to achieve acceptable performance results when performing searches over great amounts of data. The indexing process usually takes place when the computer is idle. There's also, usually, the possibility that most of these search applications be set to suspend when installed on a portable computer that is running on batteries, thus saving power.

There are, at least, three types of information about the files that a desktop search tool gathers when indexing:

- File and directory names;
- Content of supported documents;
- Metadata, such as titles or authors or even comments in several file types, like MP3 or PDF.

As for database search tools the common indexed information is:

- The database and table names;
- A primary key to the specific record.

If the search engine has the ability, or access to applications that provide the ability of parsing different types of documents, it also becomes possible to search inside documents. This is due to the use of filters that interpret the selected file formats [5].

2.2- How does a Search Engine Works?

Although there are several search engines already developed, and that their search methods may have several differences, the working method of every search engine must follow three simple steps in order to fulfill its purpose:

- Crawling;
- Indexing;
- Searching.

Each of these steps is a key part in the whole process that allows users to search for information [5].

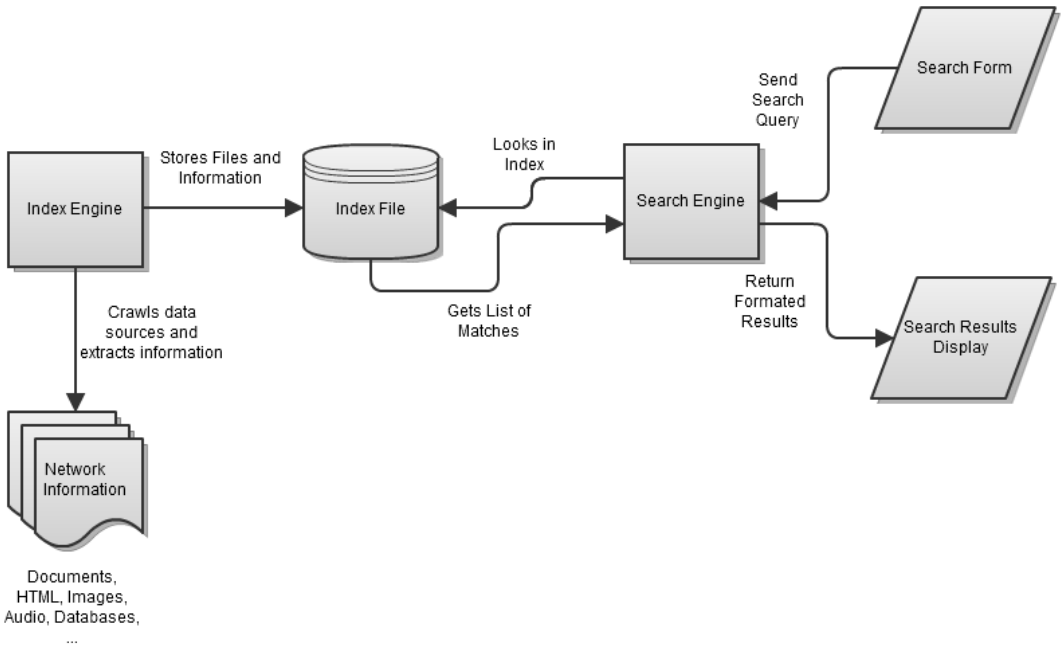

Figure 1 – Search Engine Working Schematics

2.3- The Crawling Process

Before a Search Engine can tell you where a file or document is, it must first be found by the Search Engine itself. To find the information contained in the vast number of files and documents of the hard drive, a Search Engine employs special software robots called spiders to build lists of words found spread across that hard drive. A search engine can have several spiders. To build and maintain a useful list of words a great number of files must be looked at by a Search Engine's spider. This process of lists building is called Crawling. It's also known as spidering.

The spider needs to have a starting point in order to begin the Crawling process. The usual starting points for this are very popular directories or list of heavily used files and documents. The spider will begin with a popular directory, indexing the words on its files and proceeding to subsequent directories or to other directories not related to the first. In this way, the spidering system quickly begins to travel, spreading out across the most widely used files in the hard drive.

Keeping the full system updated and running smoothly means building a component to feed the necessary information to the spiders. This component can be a tool that records new file and document entries or changes made to existent files and documents.

By crawling each file or document, a spider takes note, if possible, of the words within it and where those words were found. Furthermore, words occurring in titles, subtitles, meta-tags and other relatively important positions could be noted in order to take part in the subsequent user search. There can be intentionally ignored words, considered irrelevant to the search, or the system may take note of every single word in the page. Each Search Engine has its spidering method [5].

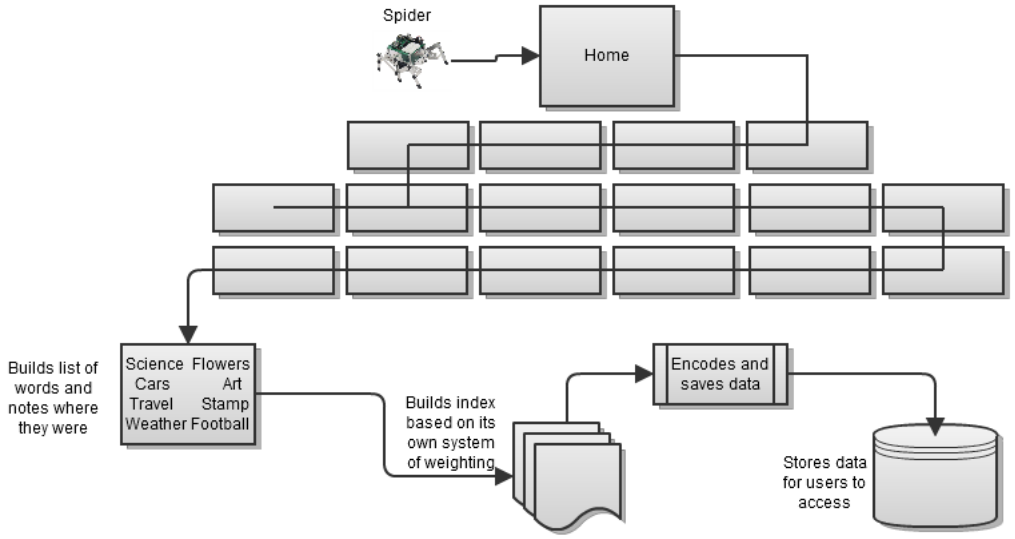

Figure 2 – Crawling Process

2.4- Meta-Tags

During the spidering process the spider will find several words. But there may be words that are equally spelled but have different meaning. This is where the Meta Tags come in handy. Meta Tags allow the file owner to specify the meaning of a word and thus guide the search engine in choosing the correct meaning for the found word. There is, however, a danger in over-reliance on Meta tags, because a careless or unscrupulous file owner might add Meta Tags that fit very popular topics but have nothing to do with the actual contents of the page. In order to prevent this, the spider should correlate Meta Tags with the content of the file or document and reject the Meta Tags that don't match the words on the file or document [5].

2.5- Building the Index

Once the spiders have finish finding the information contained in all the "selected to index areas" and the spidering process can be said complete[1], the Search Engine must store the information in a way that makes it useful. There are two main components involved in making the gathered data accessible to users:

- The information stored with the data;
- The method by which the information is indexed.

In the most simple of cases, a Search Engine could simply store a found word and the path to where that word was found. But this would provide a very limited use for the information. By only storing these two components there would be no way of telling if a word appeared in an important section of the file or in a less important one. It wouldn't be possible to know if a word appears lots of times in the same file or only one time. It wouldn't be possible to know if there are references to other files that contain that same word. It wouldn't be possible to know if the word was used in a key or trivial way in the file. In other words, it would be impossible to classify the files and documents according to their relevance to the made search because there would be no way of building a ranking list.

Due to this fact, most of the Search Engines store more than the word and its associated path. They can store the number of times a word appears in a single file. The engine might assign weights to each entry of the word, according to where the word appears in the page, with

[1] It must be noted that the *spidering* process is a never-ending process, since the *spiders* are always crawling for information.

increasing values to them as they appear near the top of the document, in sub-headings, in links, in the Meta Tags or in the title. Each Search Engine has its own methods of assigning weights to the words in its index. This is the reason why the same query in different Search Engines produces different results, with files and documents found presented in different orders.

Regardless of the combined set of attributes or pieces of information that the Search Engine will store, the data will be encoded in order to save storage space. After the information is compacted, it's ready for indexing.

An index has the single purpose of allowing information to be found as quickly as possible. There are quite a few ways for an index to be built, but one of the most effective ways, and the most common, is to build a hash table. In hashing, a formula is applied to attach a numerical value to each word. The formula is designed to evenly distribute the entries across a predetermined number of divisions. This numerical distribution is different from the distribution of words across the alphabet, and that is the key to a hash table's effectiveness.

In English, there are some letters that begin many words, while others begin fewer. You'll find, for example, that the "M" section of the dictionary is much thicker than the "X" section. This inequity means that finding a word beginning with a very "popular" letter could take much longer than finding a word that begins with a less popular one. Hashing evens out the difference, and reduces the average time it takes to find an entry. It also separates the index from the actual entry. The hash table contains the hashed number along with a pointer to the actual data, which can be sorted in whichever way allows it to be stored most efficiently. The combination of efficient indexing and effective storage makes it possible to get results quickly, even when the user creates a complicated search [5].

2.6- Building a Search

In order to search through the built index, a user must provide and submit a query to the Search Engine. It can be a minimal query of a single word to a more complex query using either a set of words or even Boolean Operators. These operators allow the user to refine and extend the terms of the search.

The most common Boolean Operators found in nowadays Search Engines are:

- ‘AND’ - All the terms joined by ‘AND’ must appear in the files or documents. Some search engines substitute the operator ‘+’ for the word AND.
- ‘OR’ - At least one of the terms joined by ‘OR’ must appear in the files or documents.

- ‘NOT’ - The term or terms following ‘NOT’ must not appear in the files or documents. Some search engines substitute the operator ‘-‘for the word NOT.
- FOLLOWED BY - One of the terms must be directly followed by the other.
- NEAR - One of the terms must be within a specified number of words of the other.
- Quotation Marks - The words between the quotation marks are treated as a phrase, and that phrase must be found within the document or file.

There are several more operators and each search engine can also have unique ones [5].

2.7- Displaying the Results

The results returned by each search made in a given SE will return a list of hits. This list may be ordered in several kinds of ways, or not at all. To order this list one may have several factors to account for. It’s up to each SE to determine which ones to emphasize. Some suggest a page ranking system, others activity based links as means to determine a coherent order for the hits display. Nevertheless, each one has its own ranking method and the same search in different SE over the same index is bound to return different results [5].

2.8- Future Search

The searches defined by Boolean Operators are literal searches in which the engine looks for the words or phrases exactly as they are entered. This can be a problem when at least one of the entered words has multiple meanings. "Bed," for example, can be a place to sleep, a place where flowers are planted, the storage space of a truck or a place where fish lay their eggs. If you're interested in only one of these meanings, you might not want to see pages featuring all of the others. You can build a literal search that tries to eliminate unwanted meanings, but it's nice if the search engine itself can help out.

One of the areas of search engine research is context-based searching. Some of this research involves using statistical analysis on files and documents containing the words or phrases you search for, in order to find other files, like sound, video or image files, or documents you might be interested in. Obviously, the information stored about each document and file is greater for a context-based Search Engine, and far more processing is required for each search. Still, many groups are working to improve both results and performance of this type of search engine. Others have moved on to another area of research, called natural-language queries.

The idea behind natural-language queries is that you can type a question in the same way you would ask it to a human sitting beside you not needing to keep track of Boolean operators or complex query structures [5].

2.9- Desktop Search Engines

DSE are SE meant to act in a one independent machine environment. They scroll through files and folders in the desktop where they're installed and allow searches over the indexed data regarding those folders and files.

2.9-1. Google Desktop Search

With the continuous increase in hard drive capacity and the fast growth in information quantity, it was just a matter of time before big Web Search oriented companies start to invest in the creation of local search engines. Google was the first major Web Company to develop and release a desktop search engine, the Google Desktop Search (GDS). There are two versions to this software, a personal version intended for the common user and a corporate edition oriented to business environments. Since the purpose of this report is the analysis of several SEs in an enterprise network, only the corporate edition will be analyzed. Also, GDS is the only cross platform SE, being available for nearly every OS available.

Having heard of GDS is probably a safe guess because of the big name Google has developed around Information Retrieval in the web. This may lead to the belief that the best choice for a SE would be GDS Corporate Edition, given the accumulated experience and know-how of Google in this area.

It has a simplistic interface, with just a small text input box that can be either side by side with the quick launch area or be called and hidden by double-tapping CTRL and an advanced search form that provides a more complex search [6]. In this form the user can restrict the searchable file types, and thus not needing to remember any specific operator to, for example, exclude file types, as well as refine the search in terms of time, by restricting the period of time regarding the results. Nevertheless there are several operators available to increase the complexity of made queries.

With every search comes a page of hits, the results, which is displayed with the aid of a browser. This page has the most relevant hits from the search and for each one GDS displays the filename and a small fragment of the file with the searched terms highlighted, as the preview of the result is quite limited, but there's a results preview list being displayed along with the search box as the text is inputted .It also provides the ability to do full text search, allowing the

search through specific fields inside each file. GDS can return ranked or unranked results, as per configured [8] [11]. This rank is a variant of the page rank system of Google's web search engine. The ability to rank the results according to their relevance is the one of the most important features of this SE. Nevertheless, the results returned are not categorized in any way. GDS just gives all the results at once, not dividing them into any type of category. Also, the results returned are not context based [10], as it is quite possible that some of the results have nothing to do with the query made by the user except for some multi-meaning word(s). When a file is opened through GDS it is instantly created a copy, of that file, in cache allowing for the recovery of accidentally deleted files [6].

It is also possible to integrate GDS with an IDE, like Eclipse, to search for specific code lines [9] [12].

As far as indexing goes, GDS indexes all fixed drives by default and allows the addition of additional networked drives to be indexed, which provides an easy to set up indexer, but that does not give much control to what is indexed, as there may be not wanted fixed drives being indexed, creating an unnecessary bigger index file, which affects search performance. The interesting part is that indexing the files from the computer must be enabled, otherwise GDS will just use the index file provided by the OS. There's also cross-indexing for multiple machines and sharing panel items directly with friends. In any case, it allows the exclusion of folders or drives from being the target of searches. It is also possible to have an encrypted index file, providing that some restrictions are met, like having EFS activated or having the volumes to be indexed formatted in NTFS. It indexes several data types with a single crawler [7], but it is possible to expand the amount of data types indexable with the aid of plug-ins that can be created by third party developers [6]. The indexing process may take several hours, depending on the amount of information available. This process is unique and will only run when the computer is idle for at least thirty seconds. By doing this there will be no downside regarding the PC's performance [6]. To keep the index updated, GDS indexes emails when they are received, documents when they are created or altered and web pages, through the browser web history, when they are visited [6].

As for administrator features, GDS provides a central management of user preferences using Microsoft Group Policy, in a Windows based environment. It becomes possible to prevent the indexing of specific files or sites and thus prevent the access to sensitive information, and also disable search across different computers. It is also possible to have several user profiles in the same machine with different access policies, providing secure searches across just the files they have access to on shared workstations. Finally it allows for the update of the application on a centralized machine and after that allows its distribution to other computers in the network [6].

As well as SE, GDS is also a widget engine. This additional feature, that provides several widgets which can be displayed in a sidebar or across the desktop, is not meant for a SE and so

if the user doesn't want it he does not have to use it, but it's not possible to install just the desktop search application without the widget engine.

One of the biggest flaws of this SE is that it is not 64-bit compatible. In order to install it in a 64-bit system a special command must be ran in the command line and still some features will not work.

2.9-2. Copernic Desktop Search

With a solid position in the desktop search engine market Copernic Inc developed Copernic Desktop Search (CDS), available in three different editions, Home, Professional and Corporate. For the purpose of this report, only the Corporate Edition will be analyzed.

CDS has won several prizes from independent organizations [13] and is said to be the fastest and most user friendly SE available in the actual market, with a simple but effective UI that allows for an incredibly easy construction of greatly complex queries. It supports several Boolean operators as well as nested searches and fixed keyword phrases. These allow for the refinement of each made query, but they're not the only way to do so. It is also possible to increase query complexity and achieve better results through the input of which specific fields to search for. If, for some reason, the user has any kind of trouble finding what he's looking for there is also a "did you mean" assistant that was created in order to help the user give the correct input to the search engine [14].

Each search returns a results list that is displayed inside the application. The results list starts to be filled as soon as text in inputted in the search box. These results, which are categorized by file type, can be sorted and grouped by several types of parameters, in order to better suit the user needs [13]. There's also a very powerful preview feature that allows the user to see where the input text was found in a given result. There are two ways of previewing results. Hovering over a result brings up a small window with some basic information of it. Clicking a result brings a window at the bottom of the results pane with the exact location of the input words in the result. There's also the possibility of dragging and dropping the results to better adjust the sorting [14]. Searches can be saved for later use and the results list can be exported in several formats [14]. An interesting feature is the "act on result" system that opens folders or files, opens, replies or forwards emails, etc, instantly [13].

The indexing capability of CDS is considerably wide, being able to index over 150 file types, emails and attachments from several email clients, etc [14]. It also gives a comfortable control of what to index giving the user the choice of what folders to index [14]. The expansibility of the indexable file types is assured, since there's the possibility of adding third party developed plug-ins to provide the ability of them being indexed [14]. It has the ability to

index remote drives in the network and the unique indexing process, which also runs in real time to index new files and update existing ones, provides the ability to control the computers resources that are used for the process of indexing documents [13]. If the user does not want a real time indexing, it is possible to create an indexing schedule for the index engine to follow, but it is possible for the indexing process start automatically, as long as there are available resources to do so [14].

The CDS deployment may be configured with certain group policies in order to enforce security, and to create customizable shortcuts in the Intranet search engine or other Web site of interest [14]. It is possible automatically deploy the application in client computers as well as update it seamlessly [14] and dedicated account management is given to each client.

2.9-3. Archivarius 3000

Archivarius Desktop Search is another example of already available SE. There are also three releases, Student, Personal and Business. For the purpose of this report only the Business edition will be analyzed.

Available in 18 different languages [15], it allows the search of all the popular file types, like Word and PDF, and instant full text search of documents and emails [15]. It also allows the search inside archives [15]. It provides integration with a great deal of email client applications, like Outlook Express or The Bat, allowing it to also search through email attachments [15].

It has LAN and Removable Drives support, which makes it able to crawl through CDs and DVDs or remote drives in the network [15]. It is strongly capable of searching through various types of archives, not only Zip or Rar or any of the most common archive formats. It supports over 70 archive types [15]. It also has a built in archive module that grants the ability of unpacking any of those formats, discarding the need for a third party application [15].

It has full Unicode support, being able to read every character encoding available. This way no characters are lost in multilingual documents search [15]. Like most SE, it allows the indexing and search through Instant Messaging Applications logs.

The most notable feature of Archivarius is the possibility of remotely execute queries and have remote access to the files via internet [15]. This means that the user can be away from his computer or away from his network and still be able to access his documents. This can be done through any browser, and, once the search has been made, it is possible to download the file to the environment in which it was made. This mode can be easily switched on and off with a single mouse click. Archivarius allows SSL connections, which provide a sense security for these file transmissions [15].

It is also an easy to customize SE from the administrator point of view. It's easy to add pre-defined queries to hyperlinks on the intranet and to create forms that can only access specific indexes, which is a good way of managing permissions, and it is very usable to add search functionality on an intranet to search content a company network [15]. It makes it easy for an administrator to enforce user rights.

2.9-4. X1 Enterprise

X1 Enterprise is developed by X1 Technologies. With the battle over desktop search engines increasing every passing year, Yahoo quickly followed Google and acquired desktop search engine technology from X1 Technologies and began distributing as the Yahoo Desktop Search (YDS). Still, X1 Enterprise endured and it's a reliable desktop search engine available in the actual market.

X1 was designed in a way that made it look and feel like the Yahoo Web Search Engine. It was built in the X1's commercial application [7].

This SE offers a very flexible and customizable indexing method. It allows the users to select the type of contents to index, like emails, web histories, files, etc. It provides the option of setting specific indexing options for each type of content and gives the user a fine grained control over the indexing process by allowing him to specify which directories to index and which type of content can be indexed [11]. It is capable of indexing a great deal of file types, counted above five hundred [16]. The indexing process can be performed in background, whenever the computer is idle.

One of the most important, and most relevant, features is the index storage compressing capability of this search engine. When building the index, X1 compresses the gathered data to roughly twenty per cent of the primary data size, on average [16].

As for the querying feature, it allows the user to save previously made queries to possibly be used later as well as the generic queries in the search pane and it has a deep integration with most email client applications.

It allows the search through the network and not only in the PC where it's installed, since it's able to crawl over networked drives. As for search results, X1 takes a reductive approach to the display. It's able to sort the results over a vast number of attributes, like the name, date, file type, file size, file location, etc [16]. Nevertheless it provides a full preview of each result over more than five hundreds file types in their native format, so the actual applications are not required to even be installed [16]. This preview is possible not only in text file and documents but also in videos, music and images since the application is media-enabled. It also allows for the search inside compressed Zip files [16]. It allows the search through metadata like "from",

"to", "subject", etc, fields and has advanced searching capabilities like search by proximity or a range search [16]. Also, it is possible to export the results list in several formats [16].

2.9-5. Windows Desktop Search

Microsoft is one of the biggest companies in the world and Microsoft Windows is the most widely spread operating system in the billions of personal computers around the world.

Windows Desktop Search is the SE that comes built within each Windows OS. It comes deeply integrated into the system and provides a user friendly, convenient and powerful interface to search through nearly everything in the PC's hard drive. It can also be configured to include or direct to remote locations, shared resources, intranet locations or even the Web.

It can search through emails and attachments, and it ensures privacy, compliance and protection of intellectual propriety by not displaying results that cannot be access by the user that is executing queries. In order to better refine searches and narrow the results list there's an Advanced Query Syntax (AQS) that has several commands and proprieties which provides the user with better means to specify what he's looking for [17], but that doesn't mean he'll always find it [17].

The index is close to the file system so it knows when a new file is created or there's a file update quickly. New files, or file updates, are made available in the index as soon as there are available resources to do so, thus not damaging the system's performance [17]. Nevertheless, WDS periodically re-indexes all files, which has its impact over the system's performance [17]. With administrator privileges it's possible to add new folders or drives to index as well as block the indexing of certain file types [17]. There are over two hundred file types natively indexable. In any case it's possible to index other file types with the aid of IFilters. There are several IFilters already available for download, but it's possible to code an in house IFilter [17]. Encrypted files are also indexable and made available to user access in search, but it won't be available to queries made through the network, rather only in the same computer where the encrypted file is located [17].

In its most recent release, WDS offers a new evolutionary step in SE technology. Every search made is correlated with contextual information in order to provide contextual searches within every explorer.

It provides intuitive file organization capabilities. The support for Live Icons, file tagging, and advanced navigation provides WDS users the ability to flexibly organize and view their files in virtually any way they wish. The hits list for every search made can be returned with rank or unranked results as per configuration [7].

2.9-6. Zettair

Zettair is an open source SE coded in C [18], developed by the Search Engine Group at RMIT. It has been ported to many different operating systems, including Linux, Microsoft Windows, Sun Solaris, FreeBSD, and Mac OS X [19] [18].

It is capable of indexing large amounts of data, having already indexed up to 426 GB of information [18]. The indexes created are portable between different machines but not between different architectures [18]. The indexer only works over HTML and text, so in order to work over different types of data it's necessary to convert them to text first. This can be made by using several filters, such as antiword or ps2ascii [18]. It can also extract text from SGML-based languages and it is possible to select which portion of those languages is indexed. To do so it is necessary to edit a configuration file [19]. By containing full word positions, the indexes created by Zettair allow the resolution oh phrase queries instead of just simple ranked queries [19]. Also, the actual implementation roughly allows a configuration of the amount of memory used in the indexing process [19].

The results provided in each query made can be ranked, as per configuration. There are several supported algorithms available to sort the results list in a ranked manner, although the pre-defined one is Okapi BM25 [19].

The SE does not have a great deal of functionalities like many of the already described. It was built to deal with large amounts of text with simplicity, speed and flexibility [18]. It has been used in TREC competition to test and improvement. Given that, it can read directly from TREC query files and the rank output given is compatible with trec_eval [19].

Finally, if Zettair has full access to the original text source it can provide query-biased summaries [19].

2.10- Database Search Engines

DBSE act like a DSE but in a different environment. Instead of dealing with folders and files in a machine they deal with databases, allowing a user to index a search the contents of a database.

2.10-1. ZeBAze

ZeBAze is an interesting DBSE that runs over Microsoft Windows platform only. It only has one edition, but to be able to fully use all the available features the software must be purchased. Otherwise, after thirty days, some features are locked.

By using Sun's JDBC technology the connection with virtually any database is assured, because 99% of the databases available in market allow connection through this type of driver, but this also means that a driver of this type must be bought. It also provides an assistant to quickly connect to a MS Access or Excel file, by helping in the selection of the correct driver for that purpose.

It provides an innovative query system through the use of proprietary artificial intelligence technology that enables the user to better filter the information and not restrain his search ability to common operators. This technology allows the definitions of words to found or words to be excluded. In addition to this, it also gives the user the ability to create regular expressions, and even combining them, providing increased query complexity. All this is done in a very user friendly environment.

The interface is divided in 4 modules. A configuration module, where the user can define the settings to apply in searches to execute and a results module, where the results are displayed, sorted by several attributes, modified and printed, along with detailed information regarding ranked values. The other two modules are both related to query construction. One regards number and date criteria, where a minimum and a maximum can be defined a level of importance is associated to each attribute. The other addresses text and Boolean criteria, allowing the definition of words to be found and words not to be found, and the construction of regular expressions with them.

There's also the possibility of writing search results directly in the database. This way there's a full integration of the information returned by ZeBAze in the system. It is possible to create several search segmentations that can be combined and queried again [20].

2.10-2. GplexDB

GplexDB is an application that seamlessly integrates in the desktop and that crawls through entire databases and projects, indexing all the source code available or all the functions, tables, columns, metadata and much more. There are four versions of this software: Home, Simple, Full and Pro. For the purpose of this analysis only the Pro version will be evaluated.

It's a simple piece of software with three components. The Crawler setup, which is extremely simple and doesn't take much time, the crawler itself, which is capable of crawling through three types of databases, SQL Server, Oracle and DB2, although this last one is still in beta phase, and the search user interface. It is also possible to specify which file extensions are to be crawled, with no restriction. All this is done from a single interface. By adopting this methodology the application provides a centralized way of accessing all the information contained in all projects and databases in the computer.

It’s an easily configurable application, being extremely streamlined, that requires no special work from the databases it will index. The interface is intentionally similar to the web search engine from Google in order to provide a familiar environment to all users, since Google has the biggest market share in web search engine area.

The indexing process is made on regular intervals, in order to maintain all the data up to date. Searches can be made as soon as the indexing process starts, quickly delivering results over the already indexed data. All searches made return a results list with highlighted data corresponding the terms that were inputted [21].

2.11- Hybrid Search Engines

HSE combine the capacity of both DSE and DBSE, allowing a user the index and perform searches over both desktops and databases.

2.11-1. Oracle Secure Enterprise Search

Oracle Secure Enterprise Search (OSES) is a standalone DBSE developed by Oracle. It provides a user friendly interface, similar to a web search engine, in order to give users a familiar environment that allows the finding of relevant documents inside a company’s network and improve knowledge sharing throughout the company

It’s a cross-platform application, coded in java, and is capable of indexing data in any HTTP-compliant server including Netscape Enterprise Server, Microsoft IIS and Apache. It safely crawls over public, private and shared content across several data sources in the intranet. For better performance it is possible to launch multi-threaded java crawlers and dispatch them in several machines. Binary files require filtering technology to extract text information from them, in order to be indexed, and for that OSES natively provides filters for more than 150 file types. The crawling process also extracts metadata from files, documents and databases, enabling better searches by allowing the search through specific fields in each one. There is also security measures associated with the crawl process.

Safety received great focus in this application. Due to that fact there are several security layers implemented. It can work in a centralized authentication scheme, such as UNIX or Windows, and a user is only given permission to access data identified as available for the login in session. It’s also possible to associate an ACL to each document as a component of the search engine index, which can be done in different ways. Finally, to prevent accesses form users whom privileges have been changed, there’s a query time authentication in which the return of a hit list by the search engine triggers the check of each element of the list to verify if the user still

has access to it. To improve even more the security level, the application is prepared to be integrated with several identity management solutions, like Oracle Internet Directory or Microsoft Active Directory.

Executing searches returns a hits list in which the most relevant hits are displayed first. In order to determine which hit is more relevant there are several ranking methods that OSES applies to the result list. It incorporates the Oracle Text relevance ranking system, with re-calibrated algorithms to better suit intranet loads; it builds different indexes for data and metadata and uses an unique weighting system to prioritize metadata over data; it performs an URL analysis similar to web search engines where it is possible to do so, since, in an intranet, the Page Rank methods for relevance evaluation is not appropriated; finally it includes a de-aliasing and disambiguation system to contextualize searches and remove duplicated entries, since the similar files, like copies, may be in several places. Metadata search is also automatically incorporated in Basic Search to determine which documents are the most relevant but it can also be explicitly invoked from Advanced Search. Another interesting feature is the Federated Search that enables searches not only over OSES's crawled repositories but also other data sources that have its own independent crawling system.

It's also integrated with GDS Enterprise to search the local desktop content, but this feature is optional and meant only for enterprises or users that adopted this SE for local searches [22].

2.11-2. Google Search Appliance

Google Search Appliance (GSA) takes all the information retrieval experience from Google and applies it to the enterprise world. It's a standalone application that brings the ease of Google Search to intranets, with algorithms specially designed to make data on servers, content management systems, databases and business applications instantly available.

There are two versions of this application, GB-7007 and GB-9009. The only difference between them is the indexing capacity. The first one supports installations of up to 10 million documents and the second one has unlimited capacity. In any case, the application can scale up to billions of documents with the GSA-to-GSA Unification Link, a feature that allows not only for multiple appliances to communicate and deliver a unified set of results, by combining search indices from several GSA deployments but also the integration of data across departments or geographies. The application is dynamically scalable, allowing the addition of more scale to the existing enterprise search deployment without disrupting the previous deployment, and intelligent to automatically determine the better way to distribute the crawling process, offering a fast and parallelized indexing process.

The appliance fully supports mirroring. Having the ability to clone itself gives the application the possibility of handling additional query load, but the clone can act as a backup. This clone receives real time updates from the primary application, and thus the crawling process only occurs once.

Searches with this application become increasingly precise with its use. This is due to the self-learning scorer that analyses user activity and reaction to specific links, and improves relevant search capabilities. GSA is able to crawl through structured and unstructured enterprise data. It indexes over 220 file types and it's able to natively crawl over several CMSs. It also provides an API to help connect to CMSs not natively supported. There are several features implemented to enhance user experience and improve results.

Security was greatly addressed in the application. GSA integrates with the existing enterprise security system and enforces the already established policy, providing document and user level control across all content. In any case, GSA provides several security protocols that add a strong security component to the crawling-indexing-searching processes [23].

2.12- Search Engine Libraries

A library is not an application, but allows a programmer to develop one. A Search Engine Library allows programmers to develop applications focused on information retrieval.

2.12-1. Apache Lucene

Apache Lucene is a high-performance, full-featured text search engine library. Its original implementation is in Java, but it has already been ported to several other languages, including C#.NET. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform [24]. In recent developments Lucene became a serious threat for big enterprises like Microsoft and Google, not only by being free by also for getting better performance results. For the purpose of this study only the C#.NET implementation will be evaluated.

Lucene.NET is a replica of the original implementation of Lucene. It's a class-per-class, API-per-API and algorithmatic port of the original Java Lucene to the C# and .NET platform using Microsoft .NET Framework [25]. Lucene.NET preserves all the names and terms of the original APIs and classes in order to give Lucene.NET the look and feel of the C# language and the .NET Framework [25].

In addition to the APIs and classes port to C#, the algorithm of Java Lucene is ported to C# Lucene. This means an index created with Java Lucene is back-and-forth compatible with the

C# Lucene; both at reading, writing and updating. In fact a Lucene index can be concurrently searched and updated using Java Lucene and C# Lucene processes [25].

It has a scalable high-performance indexing engine capable of constructing an indexing with roughly 20~30% of the original data size [24][27].

In order to execute searches one must execute a query. There are several algorithms to perform searches [24]. The results returned are ranked according to their relevance and sorted so that the most relevant are returned first [24]. There are several types of queries that can be performed, such as phrase queries, wildcard queries, proximity queries, range queries and more [24]. It's possible to search through specific fields inside the indexed data, like title or author and it's possible to sort the results list by several fields [24]. Another interesting feature is the possibility of searching through more than one index file and received merged results [24]. Search and update processes can run simultaneously, and thus saving the time of needing to do only one thing at once [24].

2.13- Search Engines Comparison

2.13-1. Financial Comparison

All the previous work was conducted in order to give an overview over the possibly available choices and to help determine what would be the best choice in a possible SE acquisition. One major point to analyze, in order to make a better decision, regards the financial costs that each one of the possibilities carries. Even if one of the possibilities proves to be the best, if it's not affordable it stops being a possibility, and so the first ruling out filter will be the financial aspect.

Desktop Search Engines

Google Desktop Search Enterprise is free. In any case, it requires that some publicity/marketing criteria are met. For example, the Google logo must be present and the Google search engine must always be an option in every search box.

Copernic Desktop Search Corporate costs $59.95 USD per unit. All transactions to buy this product directly from Copernic are made in American dollars and there might be taxes applied to the purchase, depending on the country where the purchase is being made, but all the details regarding those aspects are provided before the order is completed.

If a big volume of units or licenses is bought there are special discount to apply to the transaction, according to the next table:

Number of licenses/units	Unit Price
1-49	$59.95 USD
50-99	$49.95 USD
100-499	$39.95 USD
500-999	$32.95 USD
1000-2499	$29.95 USD
2500-4999	$27.95 USD
From 5000	$24.95 USD

Table 1 – Price Table of Copernic Desktop Search Corporate

Also, for online deliveries there are no additional costs but for the delivery of physical products there are shipping costs associated. These costs are calculated per product/line item, which bears the price of $3 US. This means that if we have 'x*Item A' we have $3 US extra to pay, and if we have 'x*Item A', 'x*Item B' we have $6 US extra to pay.

Archivarius 3000 Business has a unit cost of 45€. Nevertheless volume orders also have associated discounts.

Number of licenses/units	Cost	Formula
1-11	45€ to 285€	24 * x + 21
12-99	295€ to 991€	8 * x + 199
From 100	995€	-

Table 2 – Price Table of Archivarius 3000 Business

Each copy of the *X1 Desktop Search Professional* costs 35.95€. But in order to have software support from X1 Technologies and received free software updates one must have the X1 Software Assurance, which costs 14.95€ per year.

There's also the possibility to buy X1 Backup CDs, which cost 7.37€ each and that require shipping fees.

Windows Desktop Search is completely free and comes installed by default in Windows 7, although it is possible to install it in previous versions of the OS. Since Glintt HS has some contracts with Microsoft, using this SE might improve the relationship between the two companies.

Zettair is an open source search engine, which also means that it's free. It's under a license, a BSD-style license that poses no problem or threat to its use.

Database Search Engines

ZeBAze is buyable online only, and there's no option of physical delivery of the software. Each license bought is only available for a single user. The unit cost for a license is $99 US [56]. If no JDBC driver is available one must be bought from the several available.

In order to acquire a no limit copy of *GPlex* one must pay a onetime fee of $999 USD. But to received proper software support costs an additional $200 USD per month.

Hybrid Search Engines

Google Search Appliance has a starting cost of $3000 USD, valid up to 500.000 maximum documents indexed. It's possible to index a greater number of documents, but the price grows as does the number of documents to index. Nevertheless, this price reflect a combination of software and hardware, since it includes the server

Oracle Secure Enterprise Search is an expensive product. The price of a single license, for only one year, is $6900 USD and the first year of support from Oracle costs $7590 USD. If one chooses to buy a perpetual license for the product, instead of a one year license, the price changes from $6900 USD to $34500 USD. There's still a weighting factor that has to be applied to the price which can maintain or decrease the price of the application, depending on what processor the machine on which it will be installed has.

Libraries

Lucene is an open source search engine library, and thus it's free. To use it one must respect the Apache license applied to it, which is not a problem. In any case, once the application is built, with this library, one must consider all the implementation and deployment costs.

Financial Ruling Out

To rule out one of the SE in this study, that SE must be above the company's budget range, and thus unaffordable.

A classification will be given to each SE in the study, according to their financial performance, according to the following list:

- Free – self explanatory, has no costs to the company;
- Affordable – has tolerable costs to the company;
- Hardly Affordable – has costs that the company would be reluctant to bear;
- Prohibitive – it's not affordable or simply too expensive.

Desktop Search Engines	
Google Desktop Search Enterprise	Free
Copernic Desktop Search Corporate	Affordable
Archivarius 3000 Business	Affordable
X1 Desktop Search Professional	Affordable
Windows Desktop Search	Free
Zettair	Free
Database Search Engines	
ZeBAze	Affordable
Gplex	Affordable
Hybrid Search Engines	
Google Search Appliance	Hardly Affordable
Oracle Secure Enterprise Search	Prohibitive
Libraries	
Apache Lucene	Free

Table 3 – Financial Ruling Out

From the financial rule out we automatically exclude Oracle Secure Enterprise Search since it's off budget and considerably more expensive than any others, even Google Search Appliance, which falls in the same category.

2.13-2. Technical Aspects Comparison

All the SEs in this study have the ability to index the various types of file needed, and thus that is excluded as a weighting factor in the choice.

Desktop Search Engines

These SEs are meant for indexing files and documents stored in the hard drive. Although it might be possible to create a plug-in to grant them the ability to crawl over a database that is not what these SEs were created for. Also, not all of them provide an SDK, and thus not allowing the creation of plug-ins, and for the ones that do provide the SDK, it is not certain that it will be possible for a plug-in to grant the ability to crawl databases, since those SDKs are meant for the creation of plug-ins that enable the SE to index new file types.

Google Desktop Search

GDS provides a free SDK which can help in the integration with the system and in the customization of the application. It's a widely spread product and it's bound to have a lot of available information and documentation that can be a great knowledge base to correct problems and even improve the product. In any case, the SDK provided is intended for the creation of new plug-ins to add the capability to index new data types. Also, it is not constructed over .NET technology. Another relevant point is the 64-bit incompatibility of this software.

Copernic Desktop Search, Archivarius 3000, X1 Enterprise

These SEs are buyable applications and do not have free SDKs to allow developers to change the application's behavior and customize it. Since they stand as leading products in their field of application the amount of information available regarding these SEs should be sufficient to help in the resolution of problems. In any case, all of them provide support to any problem that a user might incur. It is important to mention that in order to receive proper support for X1 enterprise one must have an X1 Software Assurance active.

Windows Desktop Search

WDS is a Microsoft product, and thus .NET compatible. Like GDS, it provides a free SDK that allows developers to customize the application to better suit their needs, and still like GDS, this SDK is intended for the creation of new data types to index plug-ins.

Since it's a product from an incredibly renowned company, Microsoft, and the community surrounding it is enormous, the information available is more than enough to help solve any given problem.

Zettair

Zettair is an open source SE coded in C. Since the company uses mainly .NET technologies it may prove to be a disadvantage to choose it. The community surrounding it is quite small and the information available is limited. This means that any bug or problem found should be hard to solve. There's a support contact to communicate bugs and even to request new features, but it's not wise to rely on a free service to be always available to help. The last news about a product update was made several months ago, and waiting so much time for bug corrections is a mistake.

Database Search Engines

These SEs are meant to deal with databases and grant the ability to search over the data contained in them.

Gplex

Like the others DBSEs, this one also has the ability to index all the databases in the company. It's coded in C#.Net, and thus, is fully .NET compatible.

The source code of this application was already made freely available, so it is possible to edit the application and customize it.

There's also a support contact directly to the application's creators. Nevertheless, the development team seems quite limited, and support might not be fully adequate.

ZeBAze

This DBSE allows the user to connect to a database and search through one of its tables. It's coded in java and makes use of the JDBC technology to connect to virtually any database existent. In any case, it's not .NET technology.

It does not provide any SDK, so there's no possibility of customization and scalability.

Finally, this application was developed by a small company. The support it is capable of providing is limited, and should not be deemed reliable.

Hybrid Search Engines

Google Search Appliance

GSA is able to index an incredible amount of documents. The number can go up to billions. This DBSE is coded in C language and not in .NET technology but it is compatible with all the documents and databases required.

Support provided by Google covers all relevant aspect regarding this software and should be regarded as more than sufficient.

Libraries

A library provides a collection of classes and routines to help in the implementation of software applications.

Apache Lucene

Apache Lucene has a great advantage over most of the SEs in this study. It has a port to .NET technology, C#.NET. Since one of the requirements of the possible application to developed is to be compatible with the applications of the company, which are developed in .NET technology, this proves to be a trump card for Lucene.

Also, the community surrounding Lucene is of great dimension, as there are already several applications that use this library. This is good news that helps improving the odds of choosing this SE. In any case, there's no direct support line besides forums and email threads.

Finally, Lucene is a library that provides the ability to crawl both databases and desktops and not an application per se, so it's fully customizable to the user needs.

Technical Aspects Ruling Out

To rule out a SE according to the technical aspects studied, one must not comply with the presented requirements or constraints.

A classification of *Compliant* or *Not Compliant* will be given to each SE.

Desktop Search Engines	
Google Desktop Search Enterprise	Not Compliant
Copernic Desktop Search Corporate	Not Compliant
Archivarius 3000 Business	Not Compliant
X1 Desktop Search Professional	Not Compliant
Windows Desktop Search	Not Compliant
Zettair	Not Compliant
Database Search Engines	
ZeBAze	Not Compliant
Gplex	Compliant
Hybrid Search Engines	
Google Search Appliance	Compliant
Libraries	
Apache Lucene	Compliant

Table 4 – Technical Aspects Ruling Out

By filtering the study subjects through compliancy we eliminate a great deal of, until now, options and are left with only three options.

2.13-3. Features and Interface Comparison

Knowing what an application allows the user to do and how much control it gives over the data with which he works is another important thing to account for. The importance of what an application allows a user to do is a decisive factor in the choice to make.

Gplex

This SE has a simplistic interface, quite similar to the web search engine from Google. It is quite easy to configure and to perform searches. To perform a search one must simply input a string in the search box and press search and the results appear in a scrollable list. But before that the SE must create an index. In order to do that the application provides a setup area where it is possible to specify the file extensions to index and the data sources to crawl. This setup area proved to be quite limited. Although it is possible to specify which databases to crawl it is not possible to configure what to crawl in the database.

At this point it was assumed that everything in the database was indexed, tables, procedures, triggers, functions, etc. After some tests made with this application, it was possible to verify that in fact it indexes everything, except for the information contained inside each table. Since the main purpose is precisely to index the information contained in the database and not the code and metadata surrounding it, although it may also be indexed to provide more precise searches, this SE fails to pass this phase of study. During the tests with this application it was noted that this SE is based on the Apache Lucene Library, but it was built with the purpose of search through the metadata and code regarding a database instead of its content.

Google Search Appliance

The SE is commercialized in a Software/Hardware combo, which means that the software is sold directly with the server where it is installed. Consequently, there's no way of testing the application unless it is acquired first.

Apache Lucene

Lucene is not an application, but a library that enables developers to created search applications based on it. It provides several functionalities and customization features that, when allied with a coding language like C#.NET, give developers the ability to create applications suited to their every need.

It's possible to create applications to index local and remote locations or to index the all the file types needed. Also, it allows the indexing of, not only desktop files, but entire databases as well. In this process it can be associated an importance weight with every document indexed.

It allows the creation of several types of searches, since it provides multiple algorithms to search through the index. It's possible to create ranking methods in order to get better results from searches.

Since Lucene is a library and not an application, it has no interface, and one must be created, so it can be used by the most common of users.

Features and Interface Ruling Out

Since GSA couldn't be analyzed and Gplex failed to comply with all the requirements and constraints presented by the problem at hand, the only option left is Apache Lucene. As it was said before, Apache Lucene is not a SE per se, but a search library instead. It allows for the development of an application that complies with all that was presented. In order to do so, an application must be architected first.

2.14- Conclusions

There are several SE solutions actually available. Although all of the options studied fall in the SE category, there are several sub-categories to better classify each one of them. Between Desktop Search Engines and Database Search Engines or even Hybrid Search Engines, which are a merge of the first two, there are a lot of options available when it comes to choosing a SE.

Each SE has its own specificities, different features, supported file types, among a lot of other things, which that although all of them are SE, all of them are different from each other and a decision of choosing one of them over all the others must be carefully planned.

The analysis over all the SE in the study revealed the features and capabilities of each one of them.

Although index folders and several file types, like any DSE should be able to do, is a requirement, a DSE proved to be a bad choice. Even the DSEs that provide a SDK to allow developers to work over the application were deemed unfit, as these SDKs do not allow the development of any kind of plug-in to allow the indexation of databases.

The same conclusion applied to the DBSEs since the capability of crawling through desktop folders is not possible.

The only viable possibilities that remained are the acquisition of a HSE, which provides index capabilities of not only databases but also desktop folders, or the creation of a new application, through the use of a SE library. Further analysis revealed that HSE are either too expensive or impossible to test, since they're sold in a software/hardware combo.

The study proved that, in order to comply with the full problem's restrictions and constraints the best course of action is to develop a new solution to meet all the demands

without overturning any of the presented conditions. The option present to do so is Apache Lucene, a search library that enables programmers to create powerful search applications.

3. Building the Solution

Since Apache Lucene was chosen from the study, a full Lucene based application must be built in order to comply with the company needs. After a complete analysis, over the company's system, development methodology and the needed features for the implementation of a full application, the architecture for it must be created.

3.1- Architecture

The built architecture describes a client-server application in which all the clients can remotely connect to the server and perform actions, obtaining a response from the server. Upon some research it was decided that this connection is made through web services [29].

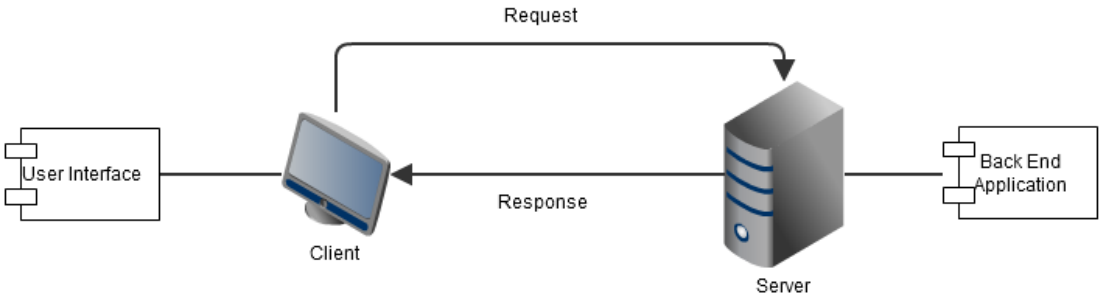

Figure 3 – Client/Server Application Structure

The client side of the application is merely composed by the interface with which the common user works to perform actions and get results. The main core of the application is installed in the server side. In any case, without the client side of the application the index becomes useless as there are no clients to perform searches over it. The same goes for client side of the application, because without the server side of the application there would be no index created and the client side of the application would also become useless, since there was no index to be targeted for queries.

The built architecture is as follows:

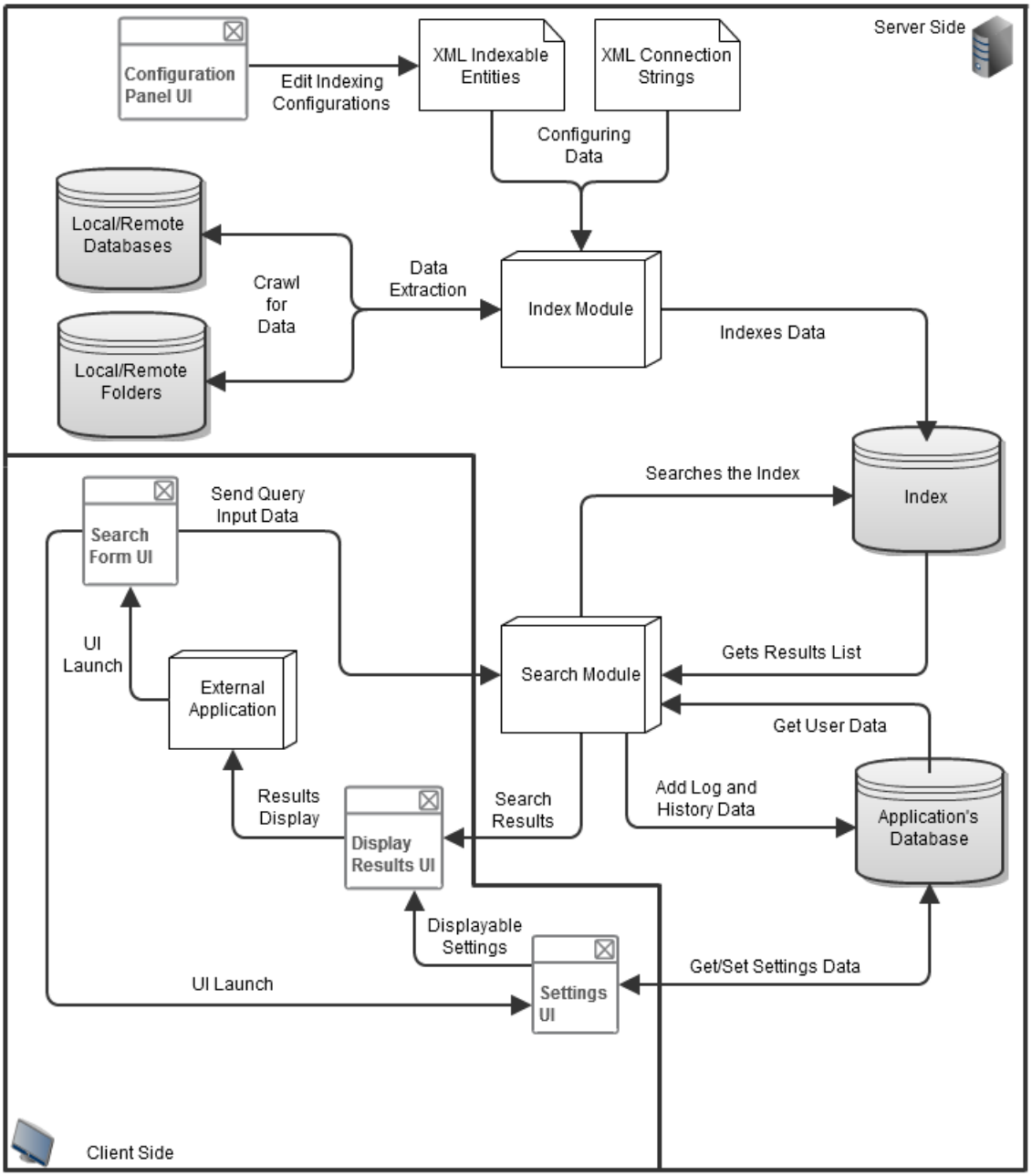

Figure 4 – Solution Structure

3.1-1. The Application's Database

The Application's Database is the storage for user settings as well as user permissions and application's configurations. This Database won't grow much after it is implemented since the only information being added will be search and access logs. All the other information will be inserted during the database implementation. There's the possibility that there's the need to add new Data Sources or Entities to the database, but that overhead is irrelevant.

3.1-2. The XML Input

These XML files are the base input that allows the application to run. Without them the application has no means to know where to fetch information, and consequently will stop. A structure for them must be defined in order for the application to parse them and be able to consistently extract the necessary information from them.

3.1-3. The Index Module

This module is responsible for dealing with the XML input, fetching and indexing the information from local and remote databases and folders, creating an index. In it, the spider(s) will be launched to fetch the information from the Data Sources and Entities contained in the XML input. During the spider(s) crawl, the module will build Lucene documents and index them, creating an Index.

3.1-4. The Index

The Index will be the target of all the queries performed by users and is composed by several files. The number of files will change as the index suffers updates, inserts, queries.

3.1-5. The Search Module

This module is responsible for receiving query input by the users in the interface in client side of the application and to search through the index file previously created. After it performs a search through the Index it returns a response to the client side interface.

3.1-6. The Interface

There are four different interfaces defined in the application:

- The Search Form: The main client side interface, it allows the user to input keywords and tags to perform searches;
- The Display Results UI: A complement to the Search Form interface, it displays the results of performed queries, returned by the server, if any;

- The Settings UI: Also client based, this interface accesses the application's database in order to edit the user's settings regarding his experience with the application;
- The Configuration UI: This interface is the only server sided interface and it's meant only for administrative purposes, allowing the configuration of the application.

This interface is to be developed in Silverlight using Prism as a development methodology. Prism ensures that the interface is developed in a modular way and each module is launched whenever needed.

3.1-7. Validation Protocols

This tool is meant to be incorporated in the company's applications and one of the requisites for it is that it inherits the validation and permission protocols of the environment where it is inserted.

If no protocols, or means to get one, are available, the tool will have its own validation system. The application's database will contain a set of roles, associated with users and entities. These roles will determine a user's clearance level to perform searches. When performing a search, the search module will analyze the clearance level of the user and perform a search over the index accordingly. There was the possibility of searching the index and execute this analysis when building the results to return. The difference between both methods is simple. The first method analyses the user permission level and retrieves the entities a user can access form the application's database. The query to execute over the index will have more conditions and retrieve only valid results, according to the user's permissions. The second method will retrieve all the result, no regarding permissions and then analyze each one for its validity towards the user's clearance level. The first method was used since there can be a vast list of hits in the search, and analyzing each one of them would be too much time consuming.

3.1-8. Connectivity

The standard means of remote communication between the several modules of the application or even between the application and external applications, whether the application is fitted in said external application or just uses its provided resources, will be made through web services.

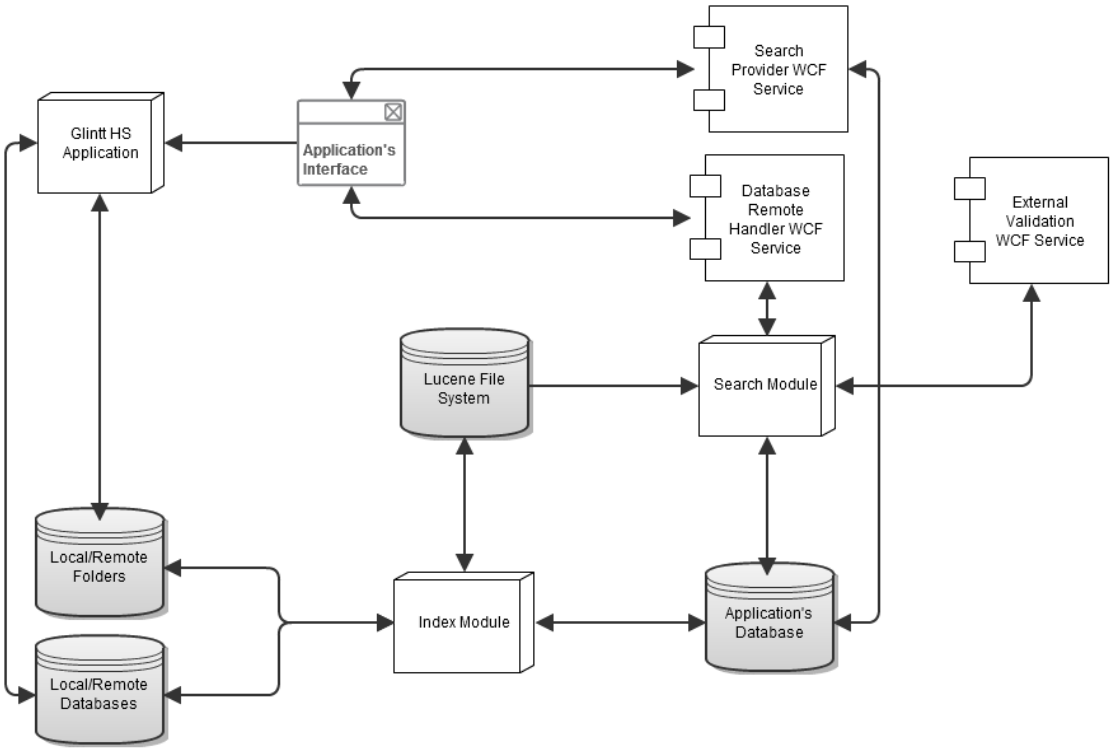

Figure 5 – Web-Services in the Solution

Three Web Services are needed to comply with all the demands of the application. All these Web Services will be constructed using the WCF methodology. These Web Services are:

- The Search Provider WCF Server: by providing a search service in the server where the application's core is installed, searches across the network are easily executed;
- The Database Remote Handler WCF Server: given the fact that the application's database is installed in the server side, all user settings must be remotely edited. This web service provides the connection interface that allows users to edit their personal settings from anywhere across the network;
- The External Validation WCF Server: whenever possible, the application must inherit the validation protocols of the environment where it is installed. From this constraint rises the necessity of some type of interface between the application and the external validation tool implemented in the given environment, which is provided by this web service.

3.2- System's Horizontal Decomposition

Both Searcher and Indexer have three abstraction layers. Two of them, the lowest two, Data Access and Middleware, are common. Nevertheless, the Middleware is different between them. In the Searcher module, the Middleware is responsible for checking the permissions from the user who requested the search and to order the results by the existent ranking method. In the Indexer module, the Middleware is responsible for crossing the data from all the entities, allowing an efficient storage of the information, making an entity's register searchable in a unique way. This means that each indexed register will have all the information regarding the entity it is representing.

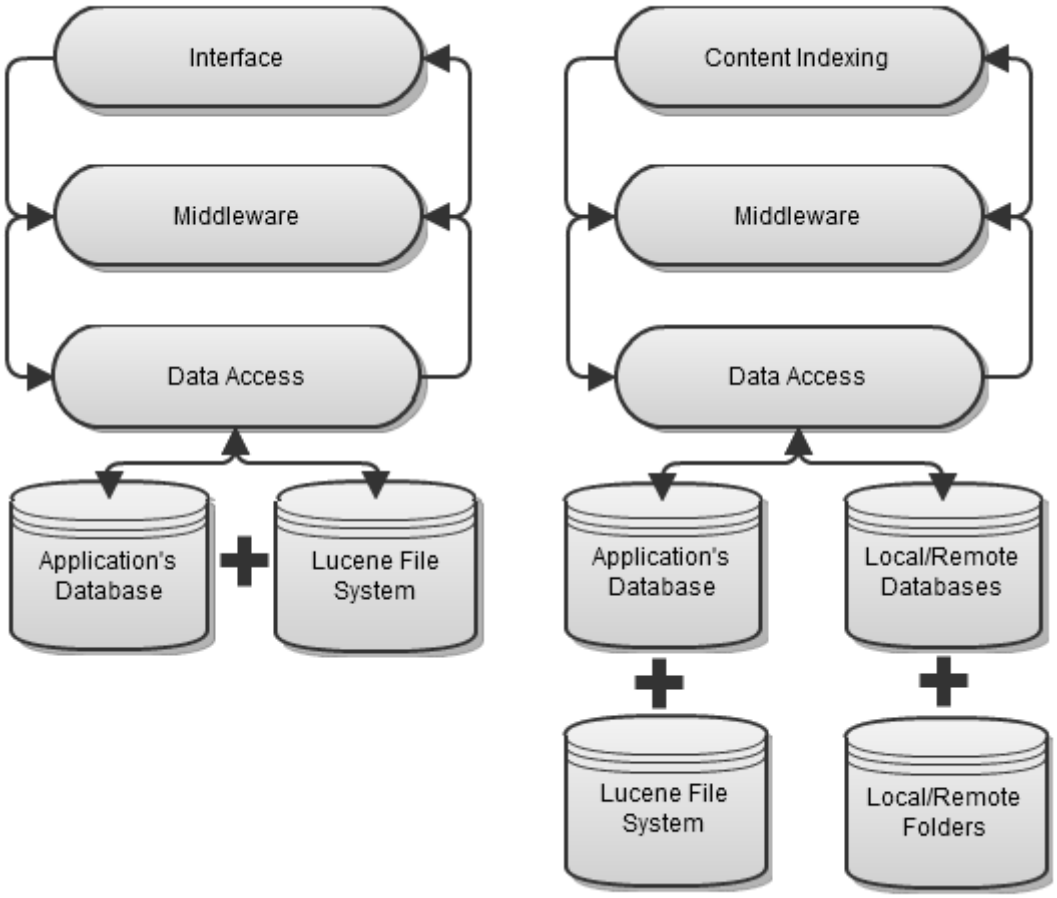

Figure 6 – Solution Horizontal Decomposition

The higher level layer is completely different in both modules. In the Searcher module it is responsible for ensuring that the user is executing his searches in the system and that he saves all his configurations and settings regarding it. In the Indexer module the highest layer is responsible for inserting into the indexing system all the documents from Lucene that have to be

indexed, according to the indexable entities configuration. It is also responsible for the re-indexation of updated content, after the first indexation process.

Another relevant aspect regards the Data Access layer. In the Searcher module it only connects to the Application's Database to get and insert information and the Lucene File system to perform searches. In the Indexer module it accesses the application database to insert information and the Lucene File System to create the index.

3.3- System's Vertical Decomposition

The vertical decomposition of the two main modules of the application describes all the sub-systems contained in the complete application and their functionalities. These sub-systems cover all the development layers.

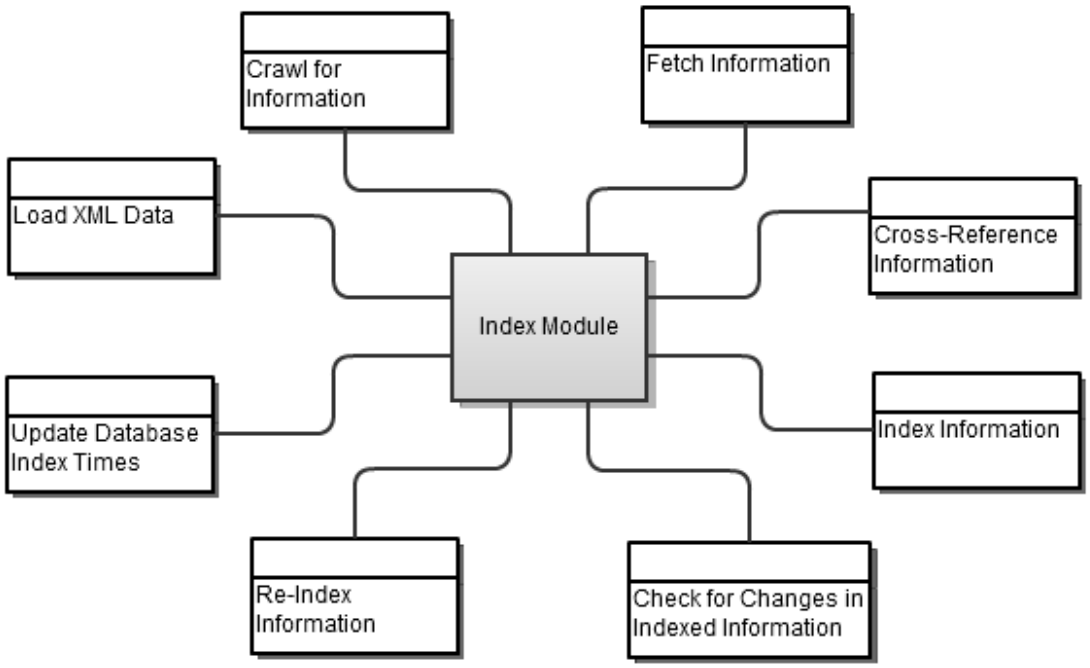

Figure 7 – Index Module Vertical Decomposition

The Index module is responsible for fetching and preparing all the information in the network so that it becomes available to be searched by the search module in an efficient way.

It loads the input information from the parsed XML input files and uses that information to perform the crawling process. While fetching the information, from the entities to be indexed, the index module will cross-reference the information, as per XML input, and build documents to be indexed, providing a better search base. After the crawl and indexation process is finished, this module will, periodically, check for changes in the indexed information as well as for new information to be indexed. If updated or new information is found, the module will index the new and update the already indexed information. In the end of the first crawl and indexation process and for every periodic check, the indexation timestamps in the application's database will be updated.

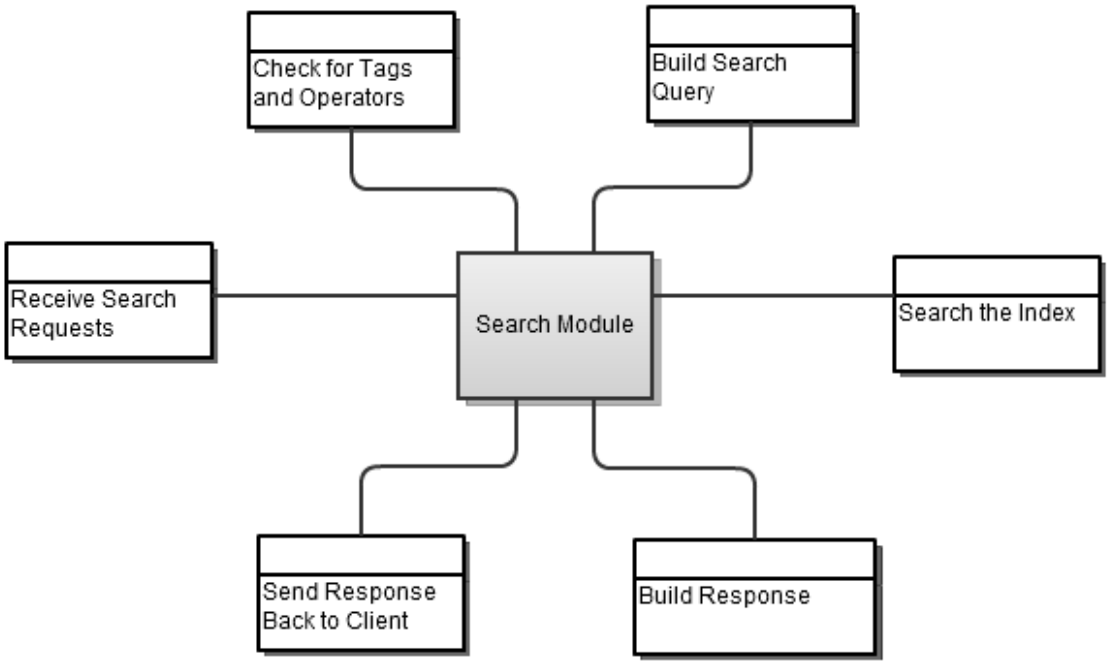

Figure 8 – Search Module Vertical Decomposition

The search module is responsible for receiving search requests and executes those searches in order to return a response to the asking client.

It starts by receiving an input string from a client. This string will contain all the information the client can provide to the search to be performed. The search module will then analyze that string, checking it for any available tags and operators contained in it. After the analysis is finished, the search module will build a search query with all the tags and operators found, if any, and with all the remaining keywords, if any. The built query will then be used against the index file, executing a search operation over it. Once the search process is finished, a response will be built with any results obtained from it. Upon finishing building said response, it will be sent back to the client.

3.4- System's Package Decomposition

With the base structure of the application defined, it's still necessary to organize it in terms of development. For that, a packages diagram was developed, in order to establish how the different classes of the complete application are going to group up.

The client based part of the application is composed only by UIs and only interacts with the application's database and search engine. The first connection is meant only to retrieve history and log data, to set customized settings and also to define personal tags to improve search experience and is made through the Database Remote Interaction package, since the application's database is located in the server side of it. The other one will process the inputs given in the Search Form UI and send them to the search engine and wait for the server's response with the search results, if any.

The Search package holds the searching core of the application and can interact with both the application's database and the index, to, respectively, get data relevant to perform search, such as permissions, and execute searches over the index, in order to create a response to send back to the client side of the application.

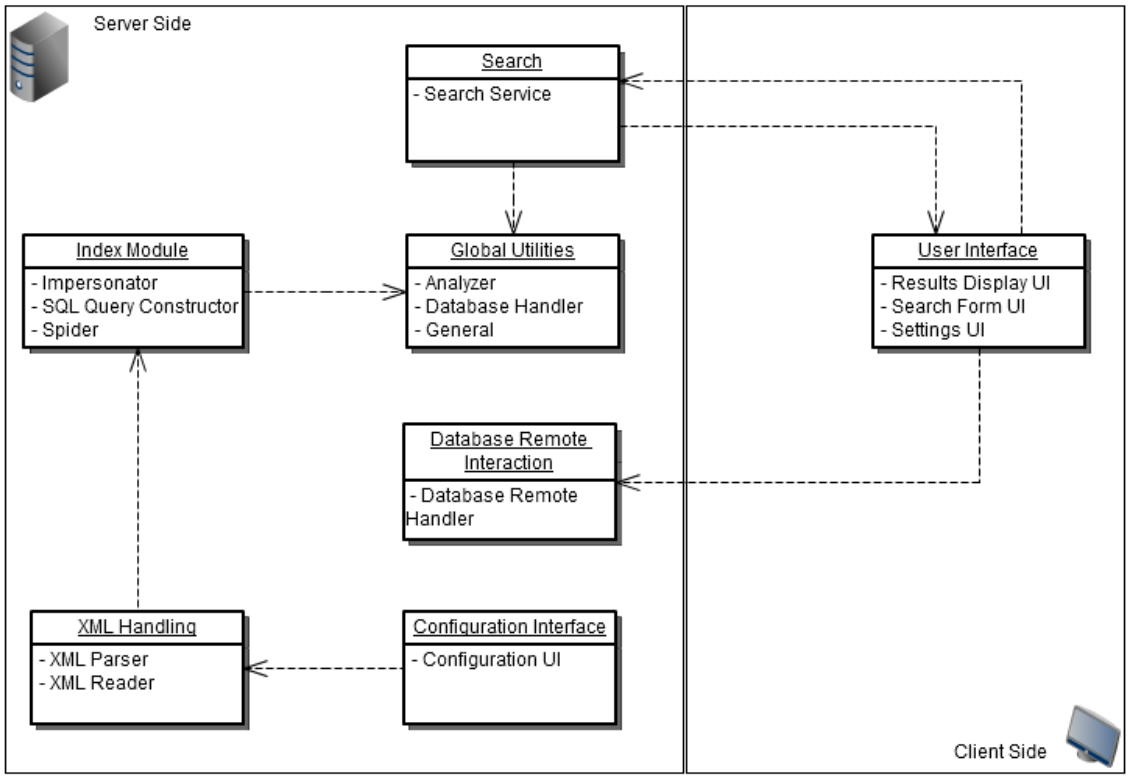

Figure 9 – Solution Packages Diagram

The Global Utilities package holds information that is relevant to several components of the application. Some good examples are global constants or structures. These are contained in the General class. It's also through this package that all the server side components of the application connect to the application's database. This is due to the fact that the application's database is local to them, and so the connection needed to access it is different from the client side of the application. The Analyzer is a component required by Lucene to create the index. It contains the rules and constraints used by the index creator to filter the information indexed.

The Index Package regards the crawling for data, the manipulation of said data and its consequent indexation into a single index.

The XML Handling package contains the classes responsible for introducing the starting settings into the application through the parse of XML files.

Finally, the only server based UI, the Configuration UI, addresses the configuration of the application, but for administrative purposes.

3.5- Use Case Diagram

A use case diagram describes what a user can do in a given environment, in this case, the application to be developed. There are two types of users in this application, the common user and the administrator. The common user only deals with the user interface from the client side of the application. Although there are several sub-types of common users, which grant different clearance levels for accessing indexed information, all of them have the same options when using the application, being the information access restriction the only nuance between them. The administrator is responsible for the configuration and maintenance of the application's server side. There's only one type of administrator.

3.5-1. Common User

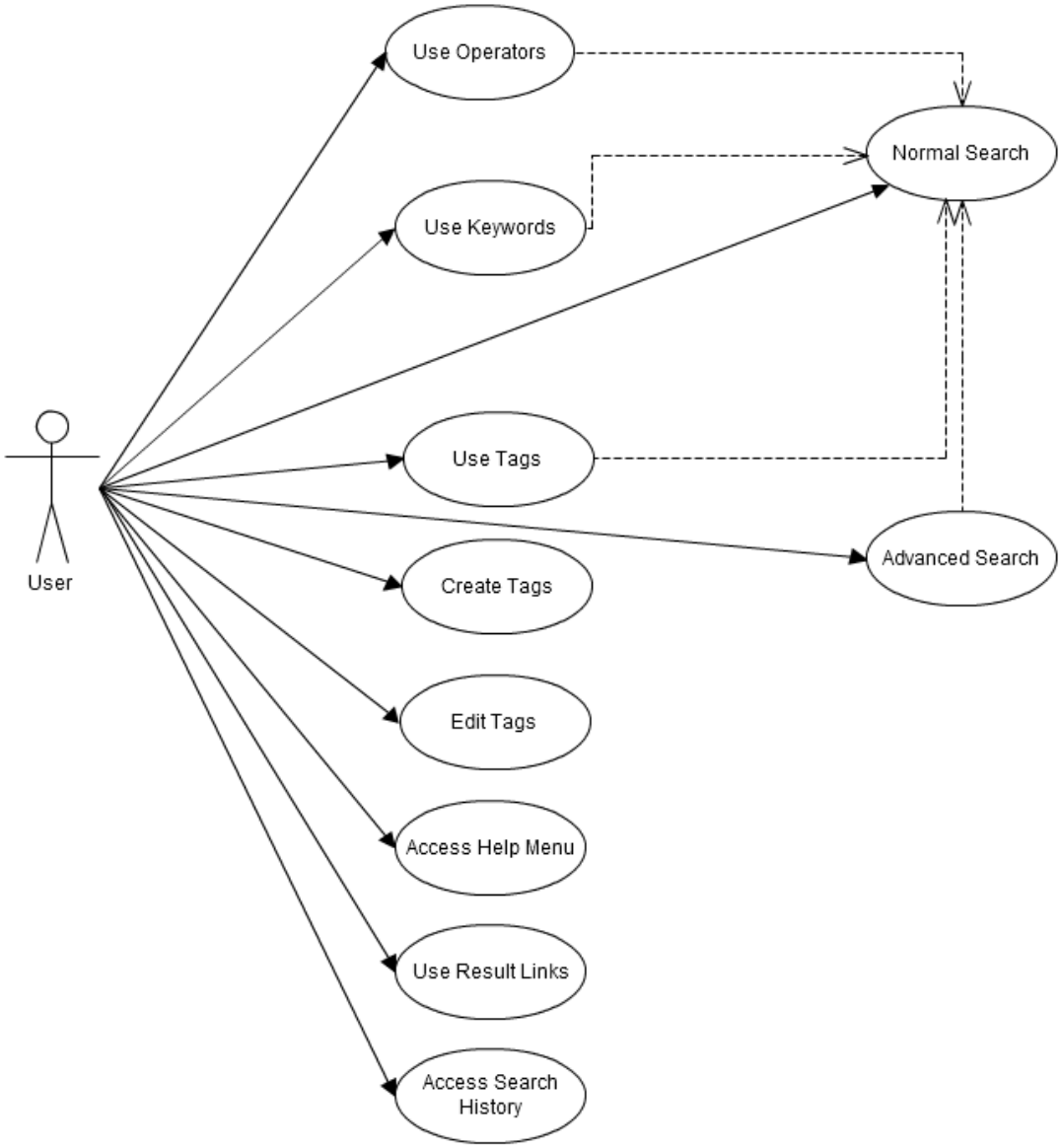

Figure 10 – User Case Diagram for Common Users

A common user can perform searches over the previously created index. If no index exists no searches can be performed. To perform searches a user can use not only basic keywords, but also operators and tags. Operators, or Boolean operators, can be use to combine keywords and improve the search's granularity. Tags can be created by the user to improve is experience with the application. A tag is a keyword that is used to determine over which data sources and entities the search is going to be performed. It is also possible to access the search history of the actual user, providing the possibility of clearing it or using previous searches.

3.5-2. Administrator

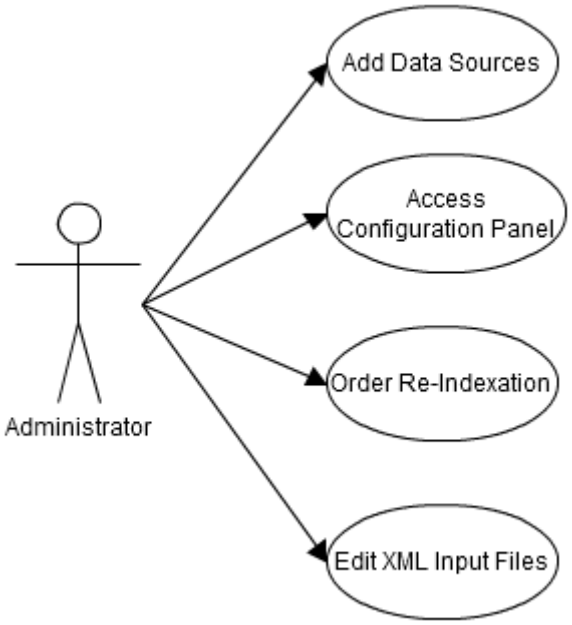

Figure 11 – Use Case Diagram for Administrators

An administrator is responsible for maintaining the application. He has the possibility of adding new data sources the index module, so that more information is available in the index. It is also possible for him to order a forced re-indexation process over a given data source. Since all the information input in the application is provided via XML files he can edit these XML files. Finally, the administrator can configure the application through a configuration panel, for instance, he can edit the max number of running indexer threads.

3.6- Conclusions

With the architecture of the application fully developed the assumption that it is possible, in theory, to develop a solution that is directly meant to deal with all the company's needs and featuring all the constraints and requirements presented by it, becomes real and doable. Nevertheless, theory must be proven by practice and, although the project is theoretically possible, it must be proved whether it is viable from the resources, financial, physical and intangible, of the company.

To prove that this solution is possible from a practical point of view, and to prove that it is viable to the company to undergo with such project, a prototype of this application must be developed. This prototype must possess the basic functionality of the full application and be a proof of concept to the application's full development. This means that, not only it must prove

that the application's development is viable by having a working prototype but it also needs to provide sufficient knowledge, not only from the implementation of the prototype but also from the theoretical development of the full application, to infer that the full application is in fact viable.

4. Developing the Solution

With all the given information and with the application's architecture defined it's time to define the necessary components to develop a working prototype of the full application. This prototype must be defined in order to provide the necessary base of knowledge to infer the viability of the project.

4.1- The Prototype

It is known that most of the searches and indexation processes to be performed upon the release of the application will occur over databases, by a very large difference, when compared to files and folders searches and indexation processes. For this reason the focus of the prototype to be developed is to create an application that is able to crawl and index databases and that is able to search over them, relegating the search and indexation over folders into a second plan. Nevertheless, it must be proved that it is also possible to do said search and indexation over folders. Otherwise not all the requirements would be fulfilled. Also the system's configuration and user settings modules go along in being relegated to a second plan.

The prototype will then require that the application's database or at least part of it be implemented as well as XML input files handling. After that it should be implemented a database crawling and indexation module that is fully functional over ODBC databases and a search module over the index created by it. It should also be developed a primary files and folders crawler and indexer, to prove that the system can be hybrid.

4.2- The Application's Database

The first thing to do before defining the data model for the database is to decide which DBMS to use. In order to comply with all the company's restrictions to software use, two options remain available: Microsoft SQL Server and Oracle Database Client. Since Microsoft SQL Server does not require any other piece of software to run, when installed in a Microsoft Windows environment, and it's a Microsoft technology, and thus largely available throughout

the company since the company is a partner of Microsoft, it was chosen to be the DBMS to the application's database.

With that decision made, the next step is to define the data model.

4.2-1. Microsoft SQL Server

The SQL Server is the DBMS from Microsoft, but it was first developed by Sybase. Some of its characteristics are:

- Easy to use;
- Offers scalability, since it can be initially implemented for a desktop and can be easily migrated to multiprocessing systems;
- It's able to implement Data Warehouses, through Analysis Services;
- It's a Microsoft technology, to whom the company is a Gold Member.

The SQL Server can be seen as a client/server DBMS, since it incorporates different types of platforms and possesses functionalities that are dividable by clients, which provide one or more interfaces that will be used to send several requests to the server, or servers, and that process those requests and send a response with the results back to the clients.

The SQL Server has its own SQL (Structured Query Language) dialect, the Transact-SQL. Although the traditional languages, like C++, VB or Delphi, among others, can only process one record at a time, SQL works with data records and does not need to detail how it performs a given task, just describing what the final user wants. This description can be divided in two sub-groups:

- Data Definition Language (DDL), which uses instructions to describe the database schema;
- Data Manipulation Language (DML), which uses instructions to manipulate the data.

SQL Server is a robust system that is used by several corporative systems from all kinds of sizes.

4.2-2. Data Model

This database is a simple support to the rest of the application. It provides a simple storage facility to the permissions each user has to the documents he can search as well has the storage ability for history and logs regarding searches made and documents accessed. It will also make possible the use of tags, which can be personally defined by each user, to address either Data Sources or Entities or even both.

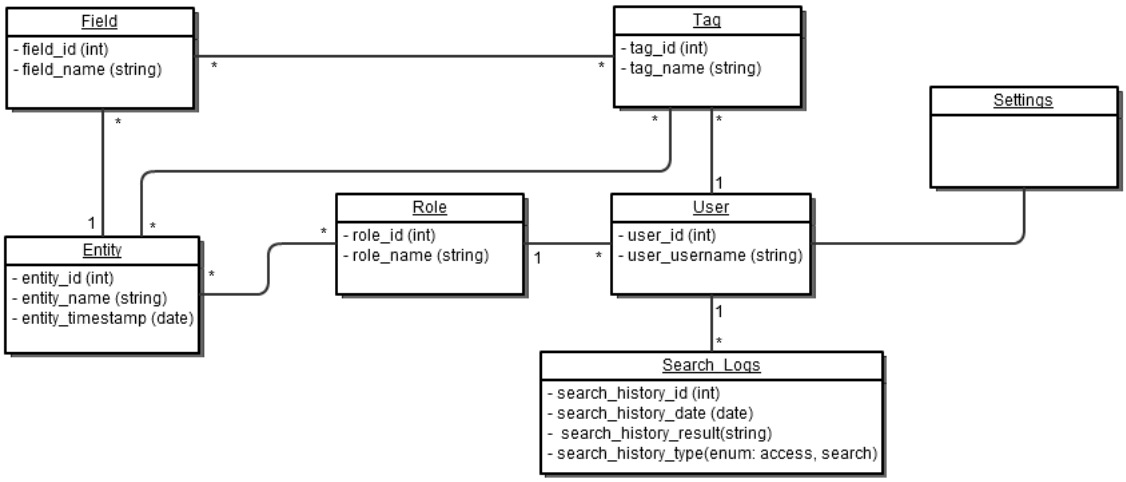

Figure 12 – Solution's Database Data Model

This simple data model will be the base for the application's database. The database will store the users and their role, which determines their clearance level for performing searches.

The Search_Logs table will record every search made and every document accessed for each user, creating a user history, by matching each row of the table with a search made or a document accessed in a certain date. This way it will be possible to monitor the users' actions with the application.

A user can also define personal tags to improve search results and help the search engine to better filter them.

Among the information stored, there will be a table to store entities and another to store fields. In this particular case, the concept of Entity is a little different from the previously defined. In here a DataSource is used like an Entity. This is done so that a tag can address either single Entities or full Data Sources without the need of an additional table. In any case, there's a particular table for fields, which can also be associated with tags. A tag can then be associated to either Entities or fields or even both. This way it's possible to increase the search granularity and get improved results.

Finally there will be a settings table, which will store the personal settings of every user, which can be edited by them in order to change their experience with the application.

4.2-3. The Database Handler

The Database Handler is a class with the single purpose of providing the interaction between the Index Module and the application's database. It contains the functions needed by the Index Module to not only connect to the database but to update records or get information that is necessary for it to properly function.

4.3- The XML Handler

The XML technology plays an important role in this application, since it's the standard way of input data into it. By knowing this, it was created a project to solely deal with the system's XML input files. In order to maintain consistency in the input data a global structure for them was defined.

4.3-1. The XML Structure

This structure, validated by a XSD document, ensures that the XML Parser created will always be able to fully parse the input files and extract every bit of information from it.

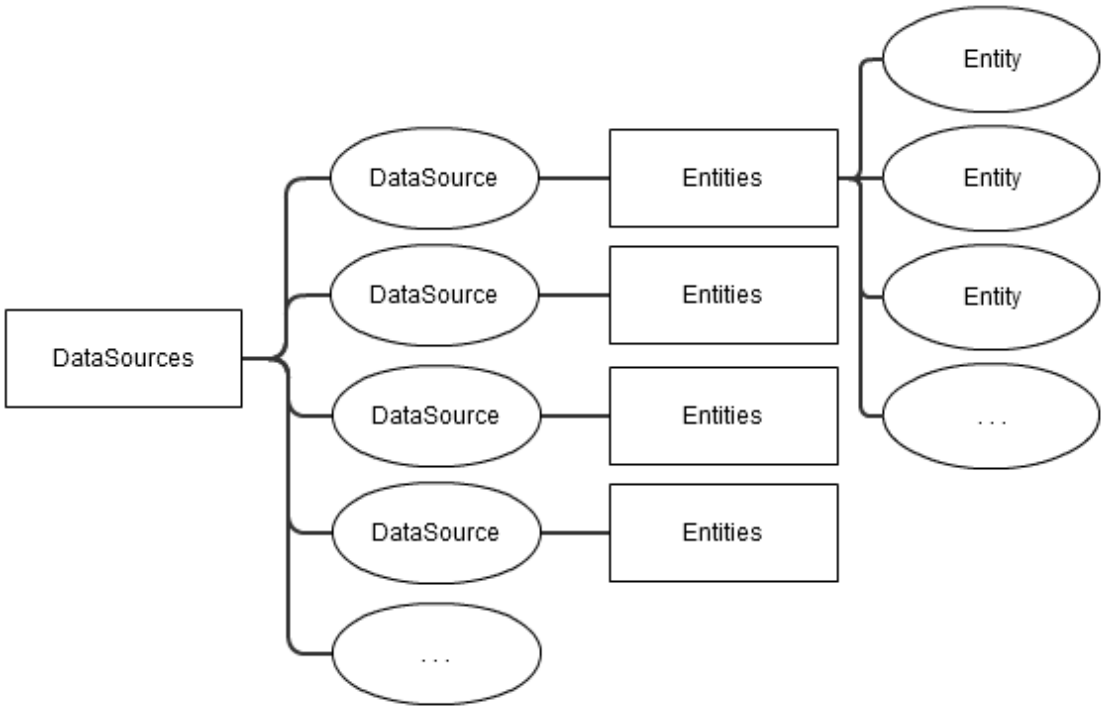

Figure 13 – XML Input File Structure

It has a main content container, called Data Sources, where all the DataSource elements are stored. Each DataSource unit has a sub-container, called Entities, that contains every Entity elements. This sub-container could be avoided and it's only meant to improve XML legibility.

It's a simple, yet effective, structure that provides a solid and consistence base of information extraction.

DataSource

Besides the sub-container for all the Entities to be index in the DataSource, there are also some attributes that are needed for the Index Module to be able to correctly function. There are two types of Data Sources and for each of those types the Index Module requires a different set of attributes to be able to perform:

- Server DataSource: from the two types available, the Server Data Sources is the most demanding one in terms of attributes information requirements. It needs an ID to identify the server and all the login information to access said server, if needed. The login information is composed by a username, a password and a domain. In addition to these attributes, there are two

flags to tell the Indexer Module if the server requires login and if the server is a remote DataSource;

- Database DataSource: this DataSource type only needs two attributes, which are an ID for the DataSource and an ID for the Database. This is due to the use of an in-house dll that provides a transparent means of database connection. By providing the two IDs to the in-house dll connection method, we get a connection straight to the wanted database in the network.

Entity

Just like the Data Sources, there are two types of Entities, quite different from each other, and still like the Data Sources, each type of Entity requires a different set of attributes to be provided. Each Entity type matches a DataSource type. For Database Data Sources an Entity will be a Table and for a Server DataSource an Entity will be a Folder:

- Folder Entity: a folder requires an ID to identify which folder in the server is to be crawled. This ID corresponds to the folder's path. It also needs a role, which matches one of the roles inserted in the application's database, in order to provide a security protection mechanism if none is available for the application to inherit. Finally it requires an update period time. This will tell the Index Module with what frequency to check for new or updated information to index;
- Table Entity: although it has some common attribute with the folder type Entity, like the update period time, the role and the ID, which in this case matches the table's name, the table Entity type is a composed type as it requires not only more attributes, it also contains some more sub-containers with key information. The table's primary key and the table's timestamp column complete the list of attributes this type require and it can have up to three different sub-containers:

  - Simple Relations: from all three possible sub-containers this is the only imperative one. In it are contained all the columns to index in the given table;
  - Complex Relations: this sub-container contains the columns that are related to columns in other tables. Each of the complex relations has an identifier tag that connects if to the relations sub-container;
  - Relations: describes the information needed to perform a connection between two tables. This information is composed by all the matching keys between the two tables and possible intermediate tables. Each Relation contains a flag that tells the Index Module whether the Relation is a one-to-many relation or a many-to-many relation.

After building the structure for the XML input files and proceed to its validation by creating an XSD file, the XML files need to be dealt with by the Index Module. In order to do that we need to parse them, so the next logical step was to create a XML parser.

4.3-2. The XML Parser

The Microsoft .NET platform provides a tool able to create classes if given an XSD file. This tool is called xsd-to-code. By having previously created an XSD file, the task of creating the parser gets easier. The tool was executed over the XSD file and the execution generated a file that acts like a parser which is able to parse the XML input files, if they are correctly structured. This file could have been directly used in the application. Nevertheless, some adjustments were made to it so that its use could be easier.

With a defined structure and an xml parser there's just one more thing needed so that the application can read the XML input files, an XML Reader.

4.3-3. The XML Reader

This reader is nothing more than a serializer of the previsouly created parser's type. The only thing the reader needs to know is what file is it supposed to read, and so, the only input it gets is the file path and it will return either a XML Parser object or a null object, depending on the success or faillure in parsing the file.

4.4- The Index Service

The Index Module is composed by three main components, which are an SQL Query Constructor, to build database interrogation queries, an Impersonator, responsible for logging in the module in secure data sources, and the most important of all, the Spider, which is responsible for searching, extracting, manipulating and indexing the information.

This module is meant to be a Windows Service that is activated automatically when the machine, where it's going to be installed, starts running. This way there's no need for an active session in order for the Spider to run, but only an active machine.

After getting a grip on how to post the data in a relational database available for querying through a search engine [28] and how to create high performance indexes [33], the following components were constructed.

4.4-1. The SQL Query Constructor

The content of a database is accessed through the execution of queries in SQL. Those queries return all that information that matches them. So, in order to extract information from a database so that it is possible to analyze and manipulate it to prepare it to be indexed the spider must be capable of querying a database, when crawling over it. The SQL Query Constructor is responsible for building the queries that the spider executes over databases.

There are four types of queries that this constructor dynamically delivers:

- Simple Queries: these queries only retrieve the x given columns of a given table;
- Complex Queries: relate tables between them and return columns of multiple tables that are related to each other;
- Update Queries: used to find new or updated records in a given table of a given database in order to build new simple or complex queries, depending on the table, to update the index;
- Record Queries: address a specific record in a specific table.

Cross-Referencing Information

Information extracted from databases is not index plain as it is extracted. Prior the indexation process, the information is manipulated so that searches can be more accurate. For instance, let's assume a table that contains information regarding bills and a table that has information related to products. The products are referred in each bill through their identification key. If we simply extracted and indexed the two tables, when searching for bills that contained a certain product we would have to search for the key to that product instead of the products name. By cross-referencing the information we can instead index a document that contains the entire bill's information and replace the products key by their respective name.

4.4-2. The Impersonator

A network has several security protocols. Amongst them is the possibility of protect files or folders or even complete access to a machine or network through a login system. In order to

gain access to resources protected in such a manner, one must input valid credentials in the login system and be validated.

The Impersonator is responsible for executing the login of the application in databases, folders, servers, etc, every time such is needed, whether those files or data sources are local or remote, allowing the spider to crawl through every wanted data source. The login information is provided in the XML input files.

4.4-3. The Spider

After the parsing of the XML file, containing all the indexable data information, has been complete the crawling for information, and consequent indexation, is ready to start.

There were several approaches made to the construction of the crawler. Since there was no way of knowing for sure how the application would perform under each approach, all of them were tested in order to find which one would be better. All the approaches made have a common base structure, which is a simple single threaded algorithm that connects to a single Data Source and deals with a single entity at a time, in order to extract information, and indexes each Document built from said information.

There were four different approaches, which are variants of this algorithm, targeted for performance benchmarks:

1. Multi-threaded with one index writer (MT1W): in this approach several threads were launched to fetch information from several entities at the same time. All the threads share the same index writer, which means that the writer must be locked when used by a thread in order to ensure thread safety;

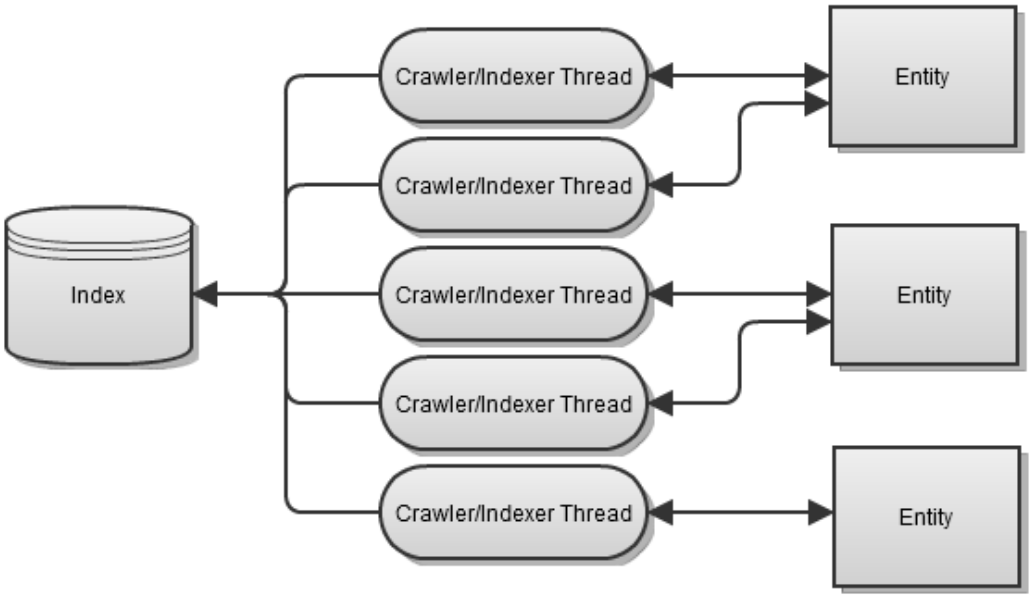

Figure 14 – Multi Threaded With One Index Writer Spider

2. Multi-threaded with several index writers (MTSW): same thread philosophy as the previous approach but each thread has its own index writer and writes its own index file. When all the threads are finished all the index files are merged into a single index file;

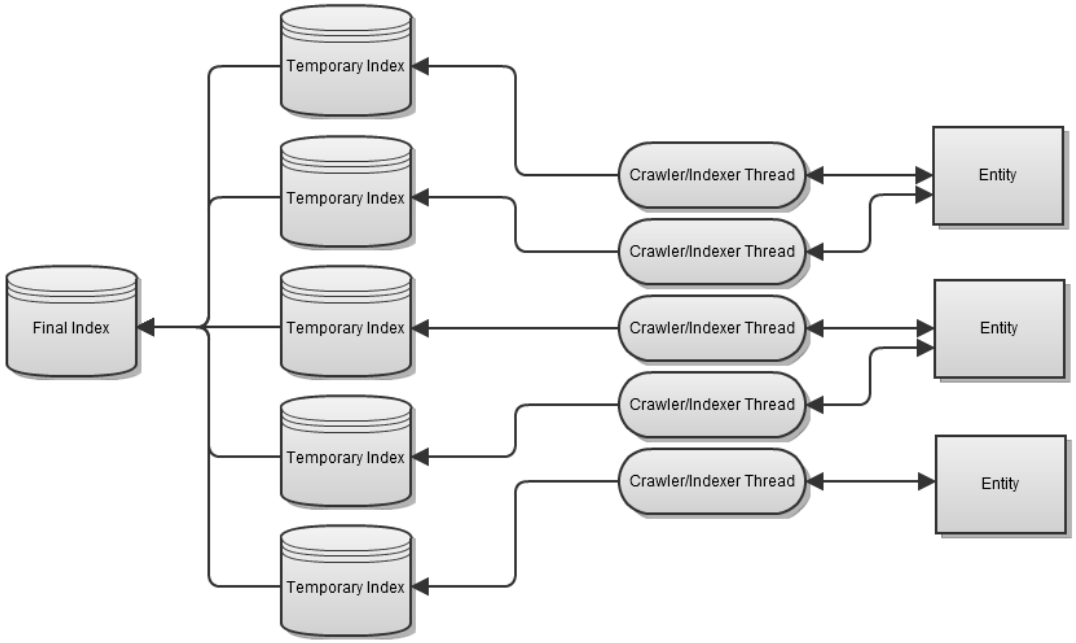

Figure 15 - Multi Threaded With Several Index Writers Spider

3. Dual Layer Multi-threaded with one index writer (DL1W): since the biggest overhead in the indexing process is the index file writing the fetching of information and Document construction was separated from the index file writing process. In this approach the same several threads are launched to fetch information and build the Documents to index but instead of writing the Document in the index file the Document is allocated in an indexation queue. In order to write the index file several threads are launched with that single purpose. These threads access the indexation queue and write the Documents in it in the index file. This means that the indexation queue must be locked when being written by the fetching threads and when being read by the indexation threads. Also all the indexation threads share a common writer, which means that, when writing the index file, the indexation threads lock not only the indexation queue but also the index writer;

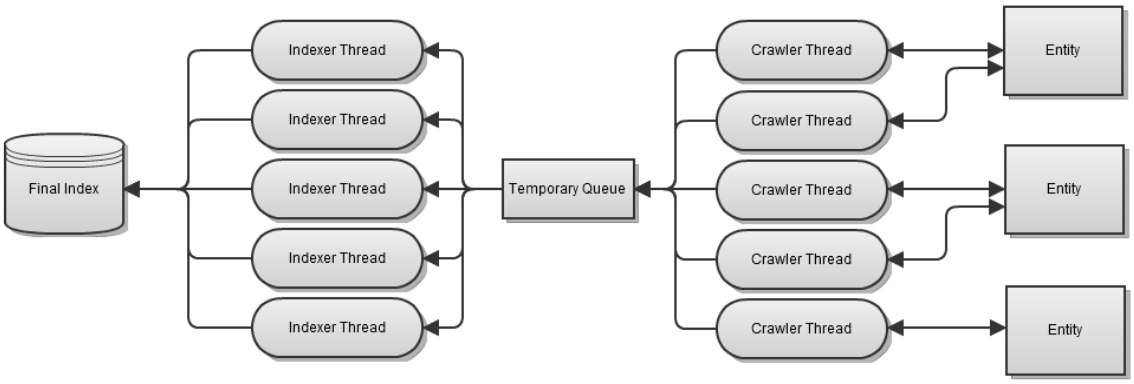

Figure 16 – Dual Layer Multi Threaded With One Index Writer Spider

4. Dual Layer Multi-threaded with several writers (DLSW): the thread philosophy is the same has the previous approach but, again, each indexation thread has its own index

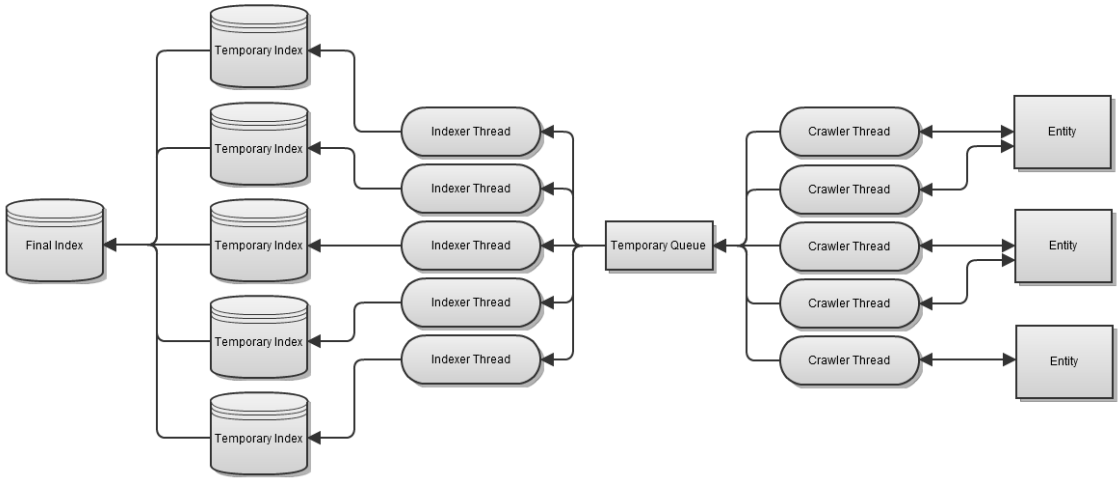

Figure 17- Dual Layer Multi Threaded With Several Index Writers Spider

writer and writes its own index file, and, again, in the end all the index files are merged into a single one.

Spider Benchmarks

Test Bed and Performance Benchmark

In order to benchmark the performance of the four different approaches a target Test Bed was built. This Test Bed is composed by six Data Sources, combining up to eleven Entities, providing a total of 17,345 files and around 1,350,000 database table records. A huge difference between the amount files and table records was chosen due to the fact that over 95% of the searches are supposed to be made over indexed table records.

The Data Sources are scattered along several locations, i.e. servers and remotes desktops, in order to better simulate the environment in which the indexer is supposed to work. It must also be referred that the condition of each benchmark test aren't always strictly the same due to the fact that the servers are in use by several other users and applications, which may be performing costly tasks in the server at the same time that a benchmark test is being run. Although these circumstances allow for the testing of the several indexer approaches in a real environment and to get a real feeling of how they would perform, they might also give an erroneous result when it comes to choose the best approach. In order to obtain more accurate results from the benchmarks, each of the four possible solutions was run ten times. These runs were not sequentially made on each approach. Instead, the benchmark tests were all ran in different order. This is meant to even the action of the crawler/indexer with the work load of the servers where it's being tested. The work load of the servers grows along the day, which means that it is smaller at the early hours of the day get heavier as the hours go by. So, to get a more accurate feeling of how each approach would perform all four approaches were tested at different hours of the day. After all the benchmark tests were run, an average running time was calculated.

Results Analysis

After running all the benchmark tests the average run time was calculated to each of the four different approaches. The index optimization process was excluded from all the benchmark tests, since it's basically the same for all the crawler structures under evaluation. Also, these times include the overhead of writing in the console at each thread cycle, which was meant to control the behavior of the application. The results obtained from these tests are displayed in the following table:

	MT1W	MTSW	DL1W	DLSW
Average Run Time (mm:ss)	42:12	53:33	47:17	55:15

Table 5 – Solution's Spider Benchmark Test Results

These results show that the best approach is the Multi-Threaded with a single writer. Although it may seem that the results are close to one another, the fact is that the indexed collection, used for this benchmark, is relatively small. When in the presence of a real life sized collection, the gap between each approach grows.

The reasons for the better performance shown by the Multi-Threaded with a single writer approach are:

- On both Multi-Threaded and Dual Layer Multi-Threaded with several writers approaches there is a big overhead for merging all the indexes into a single one, although the actual indexing process is faster with several index writers writing in the disk instead of only one. Since each index writer creates its own index file, in a separated directory, because there can be no more than an index in a single directory, the merging operation is quite costly as it has to re-write all the sub-indexes into the final directory and join them.
- In both the Dual Layer Multi-Threaded with a single and with several writers approaches there's a small but cumulative overhead in the access to the intermediate buffer. The reason for this is, in order for all the process to be thread-safe, each time a thread accesses the queue of documents to index it has to lock it so that no other can read or write from it while this thread is doing it so. Since all the threads gathering information and creating the documents to index have to access the queue to add the documents to it and all the threads responsible for indexing those documents have to access the queue to index said documents, there's too many queue locks in all the process, which have a considerable impact in the overall performance.

The Multi-Threaded with a single writer approach revealed itself as the most balanced approach between disk writing and lock policy to enforce thread safety. Also worthy of mention is the fact that, due the inexistence of a queue and a smaller number of running threads, both single and several writers' Multi-Threaded approaches are less memory consuming and less processing time demanding. But the Multi-Threaded Single writer approach has the advantage of not having to merge index files, which is one of the most demanding operations of all, since it has to re-write all the sub-indexes.

The Multi-Threaded with a single writer approach has a memory consuming peak of around 100mb and a peak processor load of around 50%, although the average processor load is far lower. This amount of processor load is sometimes reached when the maximum amount of possible threads is running.

Spider Implementation

The spider was implemented to crawl over databases and folders, both types being already crawled and indexed. The indexation process for databases is a more complex and evolved process, since it was required for the prototype. The indexation process for folders does not manipulate information in order to provide some sort of cross-referenced data to index and to get better search results. In any case, the crawling process for both databases and folders is fully functional.

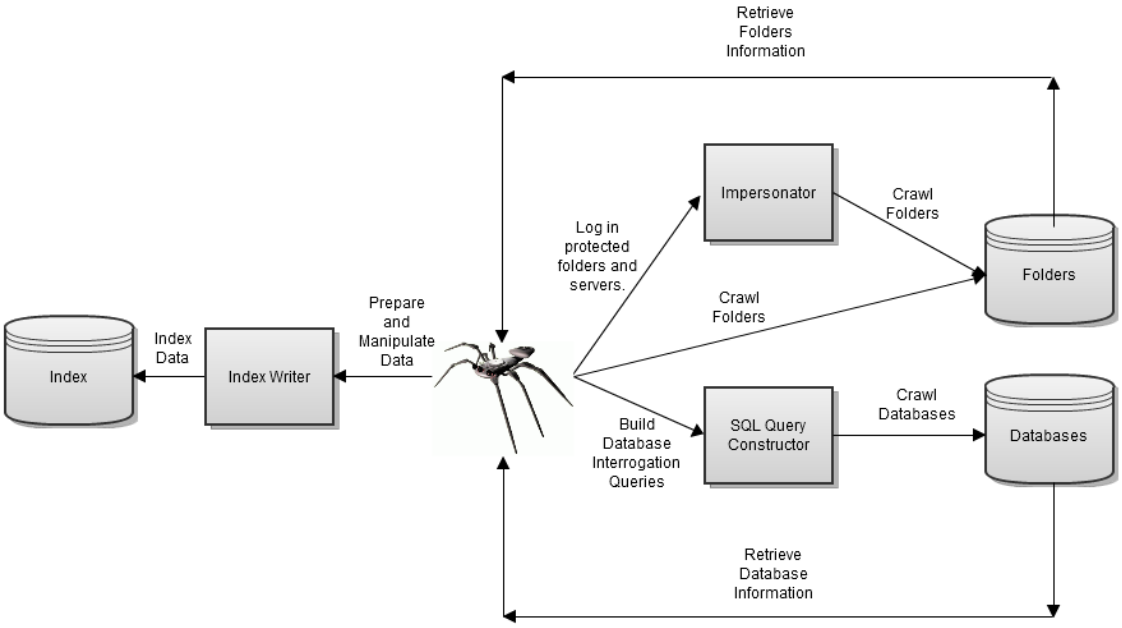

Figure 18 – Spider Execution Process

4.4-4. The Index Writer

The Index Writer is responsible for writing Lucene documents into the index file. In order to create an Index writer one must provide the folder where the index is supposed to be created.

Besides the folder path it is also possible to configure a couple of attributes of the writer. One of those attributes is the merge factor. The merge factor determines how many documents the writer will write into memory before flush them into the hard drive. The default value for the merge factor is 10. This means that, for every 10 Lucene documents indexed we have one hard drive access. Since this will definitely slow down the indexing process, bringing the performance level to a lower average, this value was changed to ten times more, 100. This means that the spider will require more RAM memory than it would with the default value, but will perform faster. This value, 100, was chosen after some testing. No random number should

be set without carefully analyzing how the spider would perform in the server. There will be more than one application running the indexation server, so the spider can't consume a prohibitive level of RAM memory, as it would perform faster at the cost of slowing down all the other applications. With this value the spider as a memory consume peak of around 100mb, being perfectly bearable for the server, and improving the spider's performance.

There was also the possibility of writing the entire index in memory and only flush it to the disk when the indexation process was complete. This type of indexation is the fastest type, since it has only one disk access and writing in memory is quite faster than writing in the disk. But to do so required an enormous amount of RAM memory in the server, since the collections of data to be index are of considerable size. For that reason this method was discarded.

4.4-5. The Monitoring System

The Monitoring System is responsible for keeping the index up-to-date. When the first indexation run is executed a silent thread is launched for each entity indexed. A silent thread is a thread that only runs periodically. The period set between thread executions is determined in the indexation process, as per XML input.

When the defined period of time as passed the thread awakes and performs a check over its entity contents in order to discover new or updated content. If the entity is a folder, the thread will access the folder, through the Impersonator class whenever required, and check every item in the folder, and sub-folders, comparing the last indexation date of that entity with the actual date of the items. If the entity is a database table, the spider will make use of the SQL Query Constructor to build an update query which will return new or updated records from that table. If any updated records are found, the old ones in the index are deleted and replaced. If new records are found they will be added to the index through the same indexation process as every other record.

Maintaining the index up-to-date is a crucial process for a search engine. This monitoring system easily fulfills that purpose. If, by any reason, there's the necessity to re-index or update a data source or entity, the administrator can issue a command to the application to force that action.

4.4-6. The Index Algorithm Scalability

Lucene provides a great deal of different methods and concepts to create powerful search applications. In this application the document concept of Lucene was largely used to create a simple, yet effective index algorithm.

The index is not filled with raw information directly from the extracted data sources. Instead it's filled with Lucene documents. These documents can have any number of fields, and a filed can contain anything the programmer wants.

All the data sources and indexable entities are input in the application via XML file, and there is no other way to add more data sources or entities except for the XML input. But the way the information contained in all the entities of each data source is indexed can be manipulated in several ways and indexed differently in each one of them. So in order to expand the search types available and to provide different experiences with the index one must only change the way a Lucene document is created, which is a very simple process. Once the information is extracted from the source the programmer just has to create a new field, associate a tag to it and then add the content to that tag in the field.

This simplicity in the index creation method allows for the creation of several types of indexes and promotes expansion.

4.5- The Index File

The Index File is the file written by the index. This file will be the target of all searches, since it contains all the indexed information. Also, all new or updated information is added to this same index.

After studying methods of mining the results to get better results in order to create the index, an information correlation algorithm was created so that the results it provides are better and searches are easily performed [30][31].

The index is built in a single folder, to be set by administrator decision. In this application this folder should be a protected one and should never be remotely accessed. The end user will never have any direct interaction with the index. In fact, in a normal course of action, a search is transparent and the end user doesn't even know of an index existence or its location. This is done to enforce security upon the index, which is the base of the search process. Without an index no searches can be performed. For this reason the index must be protected at all times and should only be accessed by the application's components and never by external users or applications that can corrupt it and destroy it, rendering the search module useless until new indexation.

When the first indexation process is finished a silent thread to optimize the index is launch. This thread is responsible for periodically optimizing the index. If this is not done the index will be scattered along several files. Although this does not mean that no searches can be performed over the index, if a Lucene search needs to search in only one file instead of a group of files, the

overhead of opening and closing files and creating file readers is reduced since it's performed only once. Also, each operative system has a limit for open files. Since the collections of data to be indexed are deemed to be of great dimension, there would be a huge number of files open to perform a search, since Lucene does not open and close the files sequentially during that process, which proves to be another reason for this silent thread.

4.6- The Search Provider

After the creation of the index the only thing that disallows users or applications to perform searches over it is the fact that the index is inaccessible to them. So in order to enable searches without compromising the index integrity a Search Provider was created.

The Search Provider is a web service that is remotely accessible and that executes local searches over the index upon requests. These requests are made through the input of keywords in the client's interface, which are then sent to the web service as a string.

There were two possibilities to build the web service. It could be either a WCF web service or an ASP.NET web service. In this case it was built a WCF web service since the company primarily builds web services in this technology. This is done so that services can be hosted in IIS instead of the Visual Studio Development Server.

4.6-1. Query Construction

The Search Service needs to receive a request, in the form of an input string, in order to build queries to be executed over the index. Upon receiving a string, the service will parse it. This process allows the service to create queries using each keyword individually as well as in subsets or even in a full set. This process allows the creation of the simplest to the most complex query. When allied to the implemented tag system, it is possible to get better filtered results.

Tag System

Tags are keywords created by the user to reference sets of data sources or entities. In order to mark a keyword as a tag in the string sent to the search provider a user must user the '#' key prior to the keyword. This way the search provider will know that he should deal with that keyword as a tag.

When parsing the keywords string, the service will search for tags in the string. Upon finding a tag the search provider will access the application's database to get information of that tag. The search provider will start by verifying if the user has the input tag and, if verification is successful, the provider will get all the entities and data sources associated with it.

With all the entities and data sources references gathered and with the keywords, resulting from the string parsing process, available, it is possible to create a complex query that associates entities and data sources references with the keywords and execute it upon the index. This way the results are more precise.

4.6-2. Hits Classification

Every time a search is executed and results are found they are returned from the search method to the provider in a certain order. This order is not random. Internally, while performing the search, Lucene sorts the results by the most relevant to the least relevant [26][33]. Only testing can prove if this default sorting algorithm is enough to provide the correct results. If that does not happen, there are several methods to create a custom sorting algorithm.

Lucene supports sorting through one to several fields. There are a couple of restrictions to be met first:

1. The field used for sorting must be either a Integer, a Long, a Float or a String;
2. The field must be index, not stored and not tokenized;

4.6-3. Results Construction

The Search Provider is a WCF web service, which uses SOAP as the protocol for messages transfer. This means that, when any request is processed by it and a response must be returned, this response will be in XML, with a default structure define by the type of response. Instead of the default structured response this web service would give to any received request, a response structure was created. This structure takes the form of an array of objects. These objects are composed by three simple elements:

1 – The Hit's Entity's Name: this is meant to know where the hit was found. It can be a folder or a database's table;

2 – The Hit's Content: to test the ability of the connection between the client side and the server side of the application to transport information all the indexed content of the hit is also returned;

3 – The Hit's Key: since the purpose of the application is to find information in order to better access it there's the need of a key that enables the user or the application where this tool

is incorporated to access the hit. If the hit is a folder the full path to that folder is returned. On the other hand, if the hit is a table record the primary key for that hit is returned.

4.6-4. Domain Handling

Since this provider is supposed to be accessible inside a full network and that a network can have several different domains, the provider must be able to serve requests from every client in the network, whichever the domain that client is in. In order to equip the provider with the capacity of doing so a cross-domain policy was developed.

Since the networks where this tool is supposed to be installed in are closed to the outside, it is safe to create a policy that accepts all the http requests from all the domains. Otherwise, a specific cross-domain policy would have to be created for every environment.

4.6-5. Search Provider Implementation

The Search Provider was implemented to access the index created by the Index Service. It receives strings from the client side of the application and parses those strings in order to search for tags and operators and to be able to use each keyword individually. In order to test the prototype it was created a result structure that defines a result, which means that, by having

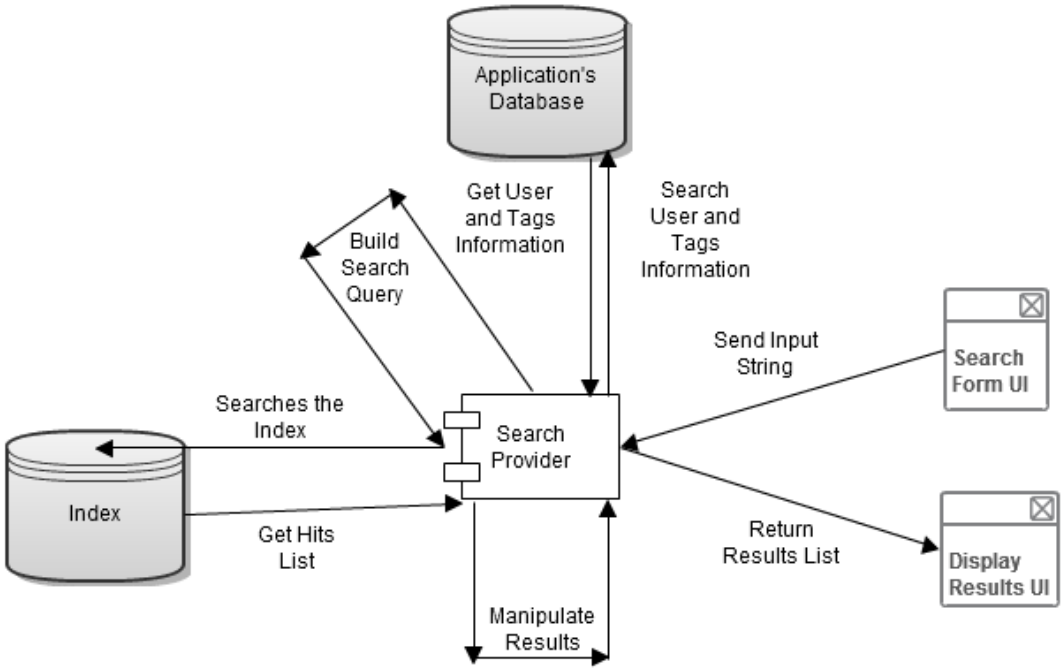

Figure 19 – Search Provider Execution Process

several hits, a search executed by the Search Provider will return an array of this structure.

4.7- The Interface

There are four interfaces conceptualized for this application. The prototype requires only the Search Form and the Results Display UIs. In order to test the application a client interface was created. This interface is a merge of both Search Form and the Results Display UI.

This interface is very simplistic. It's only composed by three elements:

Input Box: this is a mere text box where the user writes all the keywords, operators and tags he wants to apply to his search;

Search Button: this element, a simple button, triggers the execution of the search. By clicking it all the keywords in the input box will be input into the system and sent to the search provider, in order to execute a search and get a set of results back;

Results Box: a list box that displays all the results the search provider web service sends back upon executing a search.

This interface is meant to be a first sketch of the interface of the client side of the application

Figure 20 – First Sketch of the Search Form UI

It still is pretty rudimentary and primitive, but it is only meant to fulfill the needs of the prototype and it's able to do just that nicely.

4.8- Conclusions

The prototype has been successfully developed with all the features required for a basic functionality and some more. These functionalities although basic, are the base for the full application's development. Although all the implemented functions are up and running perfectly they're bound to not be the final version of themselves. Nevertheless they allow studying the working potential of the full application, if developed, and even if this last one is viable.

Since the application works in two very distinct locations in the network, it required two different solutions, one for the client side of the application and other to the server side of the application. The client side only contains the developed interface and remote access to the Search Provider web service and the server side contains the Index Service and said Search Provider.

5. Simulation and Results

In order to test the prototype it was conducted a simulation experience. This simulation consisted simply in indexing a collection of data and performing searches over it in order to prove that the prototype in fact works. If the correct results were to be obtained then the application is working, even if in an alpha phase.

5.1- Simulation Scenario

This simulation required two different distinct machines, one for the server side of the application and other for the client side. Also, these machines were deemed to be in different domains, in order to also test the effectiveness of the cross-domain policy.

The target collection for this simulation was the previously created test bed, which already contemplated several documents and records to index scattered across the company's network.

The two machines, although with different hardware specs, had the same operative system, Windows 7, and software installed. Nevertheless, the application is supposed to work in such an environment of different machines but with the necessary software.

5.2- Experimental Results

The Index Service was started in one of the machines in order to verify if the Index Module was working as it should and to create the index that was meant to be the target of all the searches executed in this simulation.

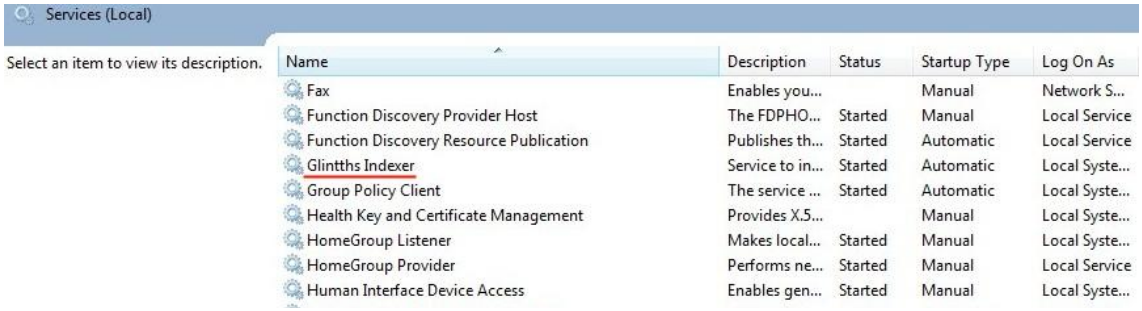

Figure 21 – WindowsOS Services Display

It was verified that the index was created and that each one of the entities' timestamps, that regard their last indexation time and are used in the monitoring and update system, were in fact updated.

entity_id	entity_name	entity_last_indexation
1	[redacted]	22-06-2010 12:04:50
2	[redacted]	22-06-2010 12:04:51
3	[redacted]	22-06-2010 12:04:51
4	[redacted]	22-06-2010 12:04:51
5	[redacted]	22-06-2010 12:04:50
6	[redacted]	22-06-2010 12:04:50

Figure 22 – Entities Index Timestamp Update

Instead of directly proceeding to the search part of the simulation, the monitoring and update system of the indexer module was tested. Prior to the start of the Index service, the XML input file was edited so that all the entities had an update time of only one minute. After the indexation process was completed some changes were made to some records in a couple tables and some files were edited. After the full minute had passed the silent threads, launched in the first indexation, awoke and start checking for changes and found all the updated records and files and actually updated the index and thus rendering the monitoring and updating system functional.

After that test is complete, the Search Provider and interaction with the client side of the application must be tested too. One of the tables indexed contains logs of activity, in which most of them are errors. By searching for the word error we obtain a list of results, with the data source of the hit and its content. It was also created a hover event for each hit, which displays the key that will enable, later on, the access to a specific table record or file. If the hit is a table record that key is the primary key value for it. On the other hand, if the hit is a file the key is its full path. Since this is only a prototype, and only the basic search feature was implemented, the exact word we're searching for must be input, because no proximity search system is yet developed. For example, if a user decides to search for the file called *example_file.txt* he must input *example_file.txt* in the input box.

In the first example several hits were returned, as there are many records with the word *error*. Nevertheless, and to improve performance, the search results return limit is set to 100 hits. The second example only returned one hit, as there is only one file with the given name. The results of these queries were confirmed with direct interrogation of the database and by directly searching through the collection of indexed files.

The Search Provider and the subsequent interaction with the client side of the application are also functional, although in an alpha phase too.

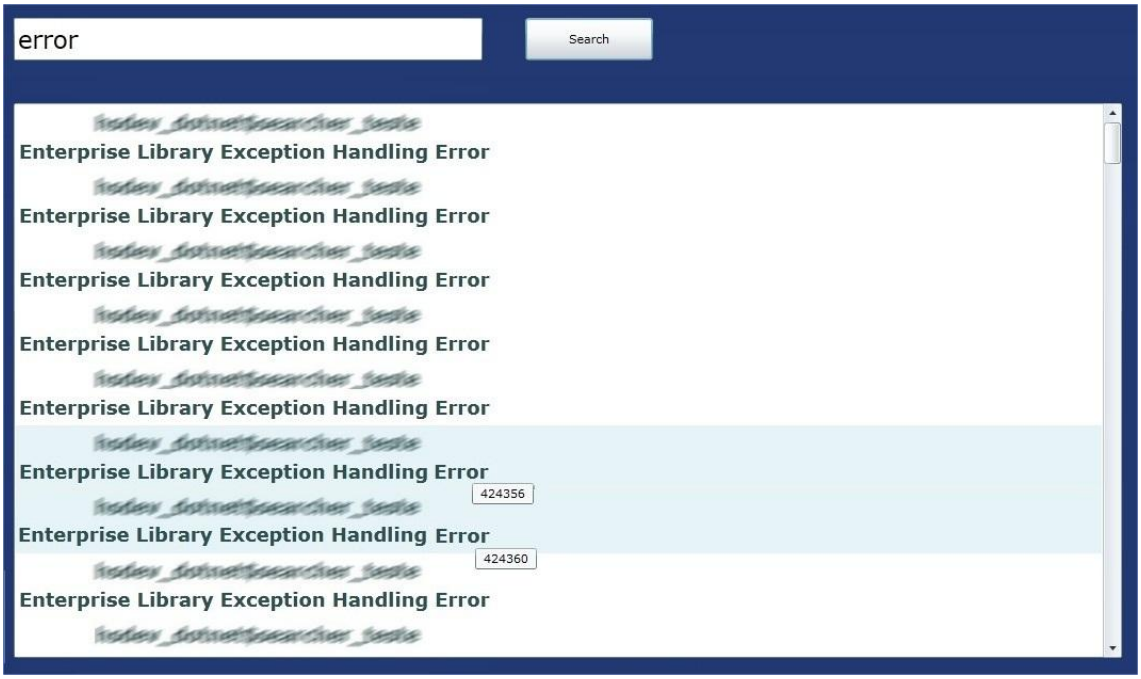

Figure 23 – First Search Example

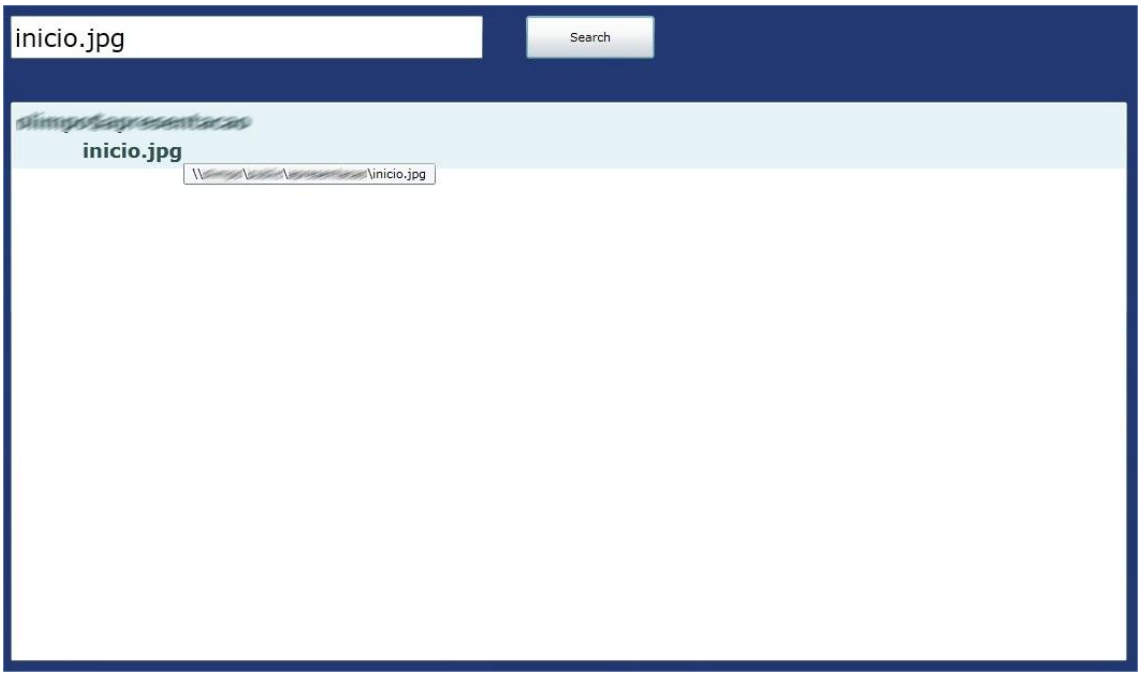

Figure 24 – Second Search Example

The prototype is up and running. Since all the not developed and already architected components of the application use the same technologies and similar development methodology, it is safe to assume that its development is a viable project, from a development point of view.

6. Discussion and Conclusions

6.1- Future Evolutions

Not being an evolution per se, since it is already architected, the first step is to develop the full application. Nevertheless, once that step is concluded it is possible to create new modules and improve the application.

In order to improve the results returned by the search provider, one evolution that could be made is to develop an own method of classifying the results, and thus manipulate their return order [31]. This way, instead of letting Lucene internally classify the results, as it was previously described, it is possible to explicitly do that and get a more accurate classification to the given environment. This can be done through methods provided by Lucene itself [26], which enable programmers to create their own scoring structure for a certain index searcher. This can be allied, for example, to a weight value associated with each entity in the XML input files, and thus re-using an already implemented schema.

Another interesting evolution is to create a resource management module that would manage the Index Service execution time and resource consumption. Like many other search engines, it is possible to command this service to stop its activities when there's high processor time consumption from other applications in the machine where it is installed and to resume its functions when the machine is idle.

To get a better a better grip of the application's functionality, it could also be implemented a module that would allow an administrator to shut down and start specific threads in runtime, particularly silent threads. Specifically for this type of threads it could also be changed the update period time. An administrator is supposed to have a good knowledge base over the network environment. If he knows that, for some reason, a specific entity will no longer be updated, or the update necessity of said entity has changed, instead of stopping the application, configure it and restart it, an administrator could make these changes in runtime.

To allow the Index Module to produce a different kind of index, an UI oriented to the construction of the Lucene document can be created, making it even easier to change the way the module will index the information extracted from the several data sources.

Add Metadata in the search process in order to better filter the results. This can only be done after and if the application is incorporated in another application [32].

These are just some examples of possible evolutions for this application.

6.2- Conclusions

It was presented a problem regarding a company, Glintt Healthcare Solutions, SA, which presented not only several requisites but also several constraints. This problem is based in the crescent difficulty in finding information in the exponentially growing sea of available information that is created every day. In order to ease the information finding process and to decrease the time spent in such a process, the company wants a tool that allows the users to search information in the its network, fulfilling all requisites and constraints of the given problem. It was concluded that the type of tool needed by the company was no more than a search engine.

In order to solve the problem at hand a study was conducted over the actually available search engines and whether they matched the constraints demanded by the company and possessed features that enabled it to fulfill all the requisites. After analyzing each search engine and comparing them, over several aspects, versus the restrictions and constraints presented it was concluded that none of the available applications suited the company's needs while being compliant with the constraints of the problem. Nevertheless, the only non-application element in the study, Apache Lucene, a search engine library, was deemed viable for allowing the development of an application that not only complied with the given constraints but also fulfilled all the requisites. This conclusion set the course of action to the development of an in-house solution for the problem.

In order to develop a solution one must first define the structure of it. Since there are several demands to the project, it was imperative that the structure to be defined was built in a way that made it compliant with them. Since the application is supposed to work in a network environment it was concluded that the best course of action was to divide the application in two distinct parts: a client side and a server side. With a client-server application a communication method between them is needed. To comply with the company's development method and to put to good use the software largely available in it, it was concluded that this communication would be made via WCF web services. This type of communication not only allowed for local and remote communications but also promotes the usability of the application. It was also decided that the XML technology should be largely used to configure the application.

Once the analysis and conception processes were complete the development stage was theoretically ready to start. Nevertheless, before the development it was decided to build a prototype to prove that the application was not only possible but also viable. This prototype should contain sufficient elements to allow the programmer to infer if the full application should be developed or not. Based on the built application structure, it was decided that a basic indexer over databases and a simple searcher over the index created that indexer should be enough.

Although these are only basic functionalities, in order for them to work compliantly with the created structure, they should follow certain rules that allow the programmer to infer viability. The prototype required a partial application's database, a communicating web service, a functional searcher and a functional crawler and indexer. Although it was only required to develop a indexer module over ODBC databases, the actual prototype took a step forward and already deals with several types of databases, and already promotes data relation, and with folders and files. These last two still in a simple and basic way. Still, there are several features already implemented that were not required for the prototype, like a monitoring system that checks for updates and acts accordingly towards the index.

After the development of the prototype a simulation scenario was created and a simulation run was executed in order to study if the application functioned accordingly to the expectations and, if so, if the full application was viable. Since the simulation proved that the application's prototype was working as expected some conclusions were extracted. Since the search provider web service was working perfectly in the network the full set of web services, of the complete application, are supposed to work as well. All the tests ran returned the pretended results and deemed the results classification algorithm at the very least sufficient. The simplicity with which it was created the index algorithm promotes the adaption of the index to the specific environment where it is located.

The cost of developing the prototype allows for extrapolation for the cost of developing the full application. Also, since the application is to become an in-house product, the maintenance and support of it bear no more costs. By summing all the costs including the future support and maintenance costs, this application is more economically viable than any other of the possibly compliant options presented in the study. This means that with a solution that is compliant with all the constraints and requisites that is not only viable but also more economical, the decision of developing an in-house solution with the available technologies by using the Apache Lucene library was well made.

References

[1] Shoben, Abigail, Kyle Rudser, and Scott Emerson. "Estimates of Information Growth in Longitudinal Clinical Trials." *The Berkeley Electronic Press* (2010).

[2] Gantz, John, Christopher Chute, Alex Manfrediz, Stephen Minton, David Reinsel, Wolfgang Schlichting, and Anna Toncheva. "An Updated Forecast of Worldwide Information Growth Through 2011." *The Diverse and Exploding Digital Universe* (2008).

[3] Saint Paul, Gilles . "Economic growth and the design of search engines." *Birkbeck Working Papers in Economics & Finance* (2009).

[4] "Glintt - Início." *Glintt*. N.p., n.d. Web. 25 June 2010. <http://www.pararede.com/>.

[5] Franklin, Curt. "HowStuffWorks "How Internet Search Engines Work"." *Howstuffworks "Computer"*. Web. 25 June 2010. <http://computer.howstuffworks.com/internet/basics/search-engine.htm>.

[6] "Google Desktop - Features." *Google Desktop*. Web. 25 June 2010. <http://desktop.google.com/features.html>.

[7] Cole, Bernard. "Search Engines Tackle the Desktop." *IEEE Computer Society* (2005).

[8] Kabutoya, Yutaka, Takayuki Yumoto, Satoshi Oyama, Keishi Tajima, and Katsumi Tanaka. "Quality Estimation of Local Contents Based on PageRank Values of Web Pages." *Proceedings of the 22nd International Conference on Data Engineering Workshops* (2006).

[9] Poshyvanyk, Denys, Maksym Petrenko, and Andrian Marcus. "Integrating COTS Search Engines into Eclipse: Google Desktop Case Study." *Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques* (2007).

[10] Dittrich, Jens-Peter, Björn Jarisch, Donald Kossmann, Marcos Antonio Vaz Salles, and Cristian Duda. "Bringing Precision to Desktop Search: A Predicate-based Desktop Search Architecture." *IEEE Computer Society* (2007).

[11] Lu, Chang-Tien, Manu Shukla, Siri H Subramanya, and Yamin Wu. "Performance Evaluation of Desktop Search Engines." *IEEE Computer Society* (2007).

[12] Poshyvanyk, Denys, Andrian Marcus, Xinrong Xie, Dapeng Liu, and Maksym Petrenko. "Source Code Exploration with Google." *IEEE Computer Society* (2006).

[13] "Copernic Desktop Search Solutions." *Copernic Desktop Search Solutions*. Web. 25 June 2010. <http://copernic-desktop-search.blogspot.com/>.

[14] "Copernic Desktop Search - Compare Versions." *Copernic - Software to Search, Find, and Manage Information*. Web. 25 June 2010. <http://www.copernic.com/en/products/desktop-search/cds-compare.html>.

[15] "Document Search – Archivarius 3000 :: Likasoft." *Document Search – Archivarius 3000 :: Likasoft*. Web. 25 June 2010. <http://www.likasoft.com/>.

[16] " X1 Email Search, Application Search and eDiscovery ." *X1 Email Search, Application Search and eDiscovery* . Web. 25 June 2010. <http://www.x1.com/>.

[17] "Windows Search." *Microsoft Corporation*. Web. 25 June 2010. <http://www.microsoft.com/windows/products/winfamily/desktopsearch/default.mspx>.

[18] "Zettair Homepage." *Search Engine Group*. Web. 25 June 2010. <http://www.seg.rmit.edu.au/zettair/>.

[19] Billerbeck , Bodo , Justin Zobel, John Yiannis , Hugh E. Williams , William Webber , Nicholas Lester , Abhijit Chattaraj , and Adam Cannane . "RMIT University at TREC 2004." *TREC 2004* (2005).

[20] "ZeBAze - Database Search Engine." *ZeBAze - Database Search Engine*. Web. 25 June 2010. <http://zebaze.com/>.

[21] "Gplex Database." *Gplex Database - iPhone InfoStore*. Web. 25 June 2010. <http://www.gplexdb.com/gplexdb.html>.

[22] "Secure Enterprise Search." *Oracle Technical White Paper* (2009).

[23] "Google Search Appliance 6 Technical Datasheet." *Google Inc.* (2009).

[24] "Apache Lucene - Features." *Welcome to Lucene!*. Web. 25 June 2010. <http://lucene.apache.org/java/docs/features.html>.

[25] "Lucene.Net." *Welcome to Lucene!*. Web. 25 June 2010. <http://lucene.apache.org/lucene.net/>.

[26] "Apache Lucene.Net 2.4.0 API Documentation." *Welcome to Lucene!*. Web. 28 June 2010. <http://lucene.apache.org/lucene.net>

[27] Wan , Jian , and Shengyi Pan. "Performance Evaluation of Compressed Inverted Index in Lucene." *Research Challenges in Computer Science, 2009. ICRCCS '09. International Conference on* (2009).

[28] Brazile, Robert, Brian Harrington, and Kathleen Swigger. "Using a search engine to query a relational database." *Information Reuse and Integration, 2008. IRI 2008. IEEE International Conference on* (2008).

[29] Zhang , Hongbin , and Juefu Liu . "Search Engine Design Based on Web Service and Lucene." *Information Engineering, 2009. ICIE '09. WASE International Conference on* (2009).

[30] zhou , Ning , XiangRong Zhang , HongQin Chen , ShaoLong Zhang , and JiaXin Wu . "Mining Weighted Association Rules with Lucene Index."*Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on* (2007).

[31] Arash Habibi , Lashkari , Vahid Ghomi, and Fereshteh Mahdavi . "A Boolean Model in Information Retrieval for Search Engines." *2009 International Conference on Information Management and Engineering* (2009).

[32] Wu, Zonghuan , Vijay Raghavan, Hua Qian, Vuyyuru Rama, Weiyi Meng, Hai He, and Clement Yu. "Towards Automatic Incorporation of Search Engines into a Large-Scale Metasearch Engine."*2003 IEEE/WIC International Conference on Web Intelligence (WI'03)* (2003).

[33] Zhang , Yong , and Jian-lin Li . "Research and Improvement of Search Engine Based on Lucene." *Intelligent Human-Machine Systems and Cybernetics, 2009. IHMSC '09. International Conference on* (2009).

[34] "Lucene Structure." *Lucene Structure*. Web. 25 June 2010. < onjava.com/onjava/2003/03/05/lucene.html>

[35] Booth, David , Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. "Web Services Architecture." *W3C Working Group Note* (2004).

[36] "Introduction to Windows Service Applications." *MSDN | Microsoft Development, Subscriptions, Resources, and More* . Web. 25 June 2010. <http://msdn.microsoft.com/en-us/library/d56de412(VS.80).aspx>

[37] Windows Communication Foundation ." *MSDN | Microsoft Development, Subscriptions, Resources, and More* . Web. 25 June 2010. <http://msdn.microsoft.com/en-us/netframework/aa663324.aspx>.

Appendix

Important Concepts

Apache Lucene

The Document Concept

The indexes created by Lucene are document based. This means that Lucene will only index documents. These documents are not the regular type of documents, but Lucene documents instead. A Lucene document can address any kind of textual information and it's composed by fields, as many as wanted. A field is composed by an identifier tag for the information being index, the actual information, whether and how the field is going to be indexed and whether and how the field is going to be stored in the index file.

The Index Optimization Concept

When indexing, the index is written in several different segments, all of them containing the same number of indexed documents, which is defined by the *merge factor* flag with the default value of 10, except, maybe, for the last segment, that will have only the remaining documents to index. "This value tells Lucene how many documents to store in memory before writing them to the disk, as well as how often to merge multiple segments together. With the default value of 10, Lucene will store 10 documents in memory before writing them to a single segment on the disk. The merge factor value of 10 also means that once the number of segments on the disk has reached the power of 10, Lucene will merge these segments into a single segment." [34]

Web Service

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [35]

Windows Service

"Microsoft Windows services, formerly known as NT services, enable you to create long-running executable applications that run in their own Windows sessions. These services can be automatically started when the computer boots, can be paused and restarted, and do not show any user interface. These features make services ideal for use on a server or whenever you need long-running functionality that does not interfere with other users who are working on the same computer." [36]

Windows Communication Foundation

"Windows Communication Foundation is a part of the .NET Framework that provides a unified programming model for rapidly building service-oriented applications that communicate across the web and the enterprise." [37]

Data Source

In this report, this term refers only to either Databases or Servers accessed by the spiders in the process of information search and extraction of information. When in the application's database, data sources are referred as entities.

Entity

In this report, this term refers to either Database tables or Server folders, which are seen as the entities to index in the given data sources. This term only gets a different when the subject is entities in the application's database. In this case entities refer to not only this spectrum of the term entity but also to data sources, this is entities equals' entities plus data sources.