XMLibrary Search: an XML Search Engine Oriented to Digital Libraries*

Enrique Sánchez-Villamil, Carlos González Muñoz, and Rafael C. Carrasco

Transducens
Departamento de Lenguajes i Sistemas Informáticos
Universidad de Alicante
E-03071 Alicante

Abstract. The increase in the amount of data available in digital libraries calls for the development of search engines that allow the users to find quickly and effectively what they are looking for. The XML tagging makes possible the addition of structural information in digitized content. These metadata offer new opportunities to a wide variety of new services. This paper describes the requirements that a search engine inside a digital library should fulfill and it also presents a specific XML search engine architecture. This architecture is designed to index a large amount of text with structural tagging and to be web-available. The architecture has been developed and successfully tested at the Miguel de Cervantes Digital Library.
Key words: Passage Information Retrieval Systems, XML Search Engines, Digital Libraries.

1 Introduction

During the last years, the amount of digital information available has grown quickly. Then, it is compulsory the development of tools that would allow a user to get easily and quickly the needed information. A digital library owns a large work collection that is made available to readers. However, the user does not often enter the library to *read* a work but to *look up* in it. In classical libraries only browsing or catalogue searching are possible.

Once the information is digitized, a wide range of exploitation possibilities is available. The next stage in accesibility is the development of search engines, that allow to find information inside the works (content and structure), to free the user from the task of searching manually the required information.

The next step is to take advantage of the structure of the digitized works. When digitizing works, it is interesting to store not only the text, but also some structural description. For this purpose, XML tagging is used. The arborescent structure of the XML documents allows higher level searches.

At present, there is a wide variety of XML searching tools and some of them are distributed as open source, such as `Fxgrep` (1), `Xset` (2), `Sgrep` (3), `XQEngine`

* Work partially funded by the Spanish Government through grant TIC2003-08681-C02-01.

(4), `Lucene` (5), `TSep` (6) or `eXist` (7). However, in general, they are document information retrieval systems (`TeraXML` (8), `XIndice` (9), `DataparkSearch` (10)) rather than passage information retrieval systems (11) like `XMLibrary Search`. Besides, we are not aware of any non-comercial search engine that generates links to the document passage in which the occurences are located; they generate links to the top of the document instead. There are some search engines that use morphological information to maximize their recall (`Convera RetrievalWare` (12)), which obtain interesting results for research purposes. Our engine offers some query suggestions[1] to the user instead, which allow them to choose the level of coverage. The use of index based search engines, which in general are faster than database based search engines, is appropriate since digital libraries do not change very often the content of their XML documents.

Search engines can be used as a basic tool for a wide range of services that could be useful for the users of digital libraries, such as summary generation applications, document analysis reporting tools or the development of a collection of learning objects for language and literature courses.

Some of the benefits of the architecture proposed in this paper are:

- The handling of a large text collection.
- The processing of a relatively complex set of queries.
- The speed to solve the queries.

The XML search engine that is presented in this paper is integrated in the Miguel de Cervantes Digital Library.[2] The paper is divided into the proposed distributed system architecture and the internals of the search engine.

2 Requirements

During the design of a search engine, some incompatible goals or features have to be compromised to achieve the needed requirements. For instance, a digital library needs fast searches while in an information extraction system the speed is not the priority although they usually deal with complex queries.

The desiderable features could be classified into two groups: functionality features (introducing constraints to the set of documents, sorting the results by their relevance, showing topics related with the query...) and system features (as speed, accuracy, robustness, safety...). Croft (13) proposed a different classification that emphasized the integration of information retrieval systems (search engines) with other systems and also stressed the distributed architectures. A different classification according to digital libraries needs, follows.

- Efficacy: As Croft stressed, a good efficacy level can only be achieved by a properly designed user interface. This interface should be both powerful enough to allow complex queries and easy enough to be used by unexperienced users.

[1] Based on morphological information, synonyms and word similarity.
[2] http://www.cervantesvirtual.com/herramientas/textos/buscador_i.shtml

- Speed: The user should be able to see the results very quickly even if many users are using simultaneously the system. The more complex the queries are, the slower their solving will be. Furthermore, the speed should not decrease significantly when increasing the size of the text collections.
- Information Retrieval: The system should not only redirect the user to the information but also preview what the user is looking for. If the user is looking for a sentence, it will not be enough to show in which work the sentence is found. The system should also show the context of the phrase and also redirect to the exact point of the work where the phrase was found.
- Safety: The search engine uses some private information that has to be preserved. Safety is directly related to information retrieval because any search engine needs full access to the works.
- Multimedia: The multimedia digital libraries have extra requirements, due to the fact that the handled data is much larger in size and, therefore, processing time is longer. The search engine presented in this paper does not handle this kind of data. However the proposed architecture would easily allow the integration of this feature.
- Query expansion: The query expansion consists in the generation of variations of the user's query. That variations can be generated after the search, as a suggestion of related queries, or before the search to offer higher quality coverage results. These expansions are usually based on thesauri and natural language processing heuristics.

Any architecture for a search engine should take into account all of these topics, so that its quality along with its funcionality could be properly measured.

3 Search Engine Architecture

The architecture presented in this paper is a distributed system with three differents layers, as can be seen in figure 1. These three layers are fully independent and communicate each other using TCP/IP protocols. To maximize the performance it is recommendable to execute each layer in different computers. The performance of the system is given by the maximum number of searches that the system can process per second.

Users access the search engine through the web server, which executes the search engine client to send queries to the user server. The user server is the responsible of query distribution to the available query servers. Once the query server has finished attending to a query the user server sends its result to the corresponding client. All query servers run independently, so that they need a local copy of the search index.

This architecture offers a very good performance, and if the average load of the system is small then all servers can run on the same computer. The different layers are explained in detail below.

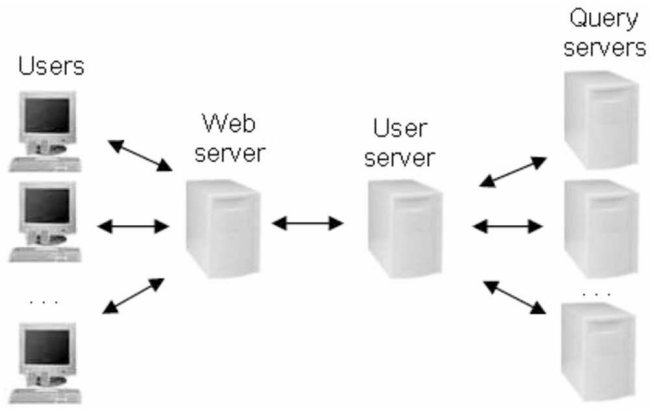

Fig. 1. Distributed architecture of the search engine.

3.1 Search Engine Client

The search engine client is the volatile part of the architecture, provided that each client processes a single query. The functionality of the client is restricted to the designed interface,[3] which is shown in figure 2. It builds a data packet containing the query and information related with the user and sends it to the user server. Additionally, the client can provide an adaptation between different encodings.

The query syntax is specified by a grammar designed to allow only possible queries that can be resolved quickly with the index.[4] This protocol optimizes the querying of the index to speed up the solving process. The query does not specify only what to look for but also how to search in the index.

The data packet is sent to the user server and, after that, the client waits for the result. The result consists of a webpage fragment, so that the client adds the header and the footer, which allows the rest of the search engine to be completely independent of the webpage design. Once the result has been received, the client builds the webpage to show it to the user and finishes its execution. In figure 3 a sample query result is shown.

3.2 User Server

The user server coordinates the whole system. It receives requests from all clients and distributes them between the available query servers.[5] After receiving the

[3] Each digital library should build its own interface.
[4] Depending on the architecture some queries are easy to process and others are not (see Baeza-Yates (14)).
[5] Except for some requests that can be answered directly by the user server.

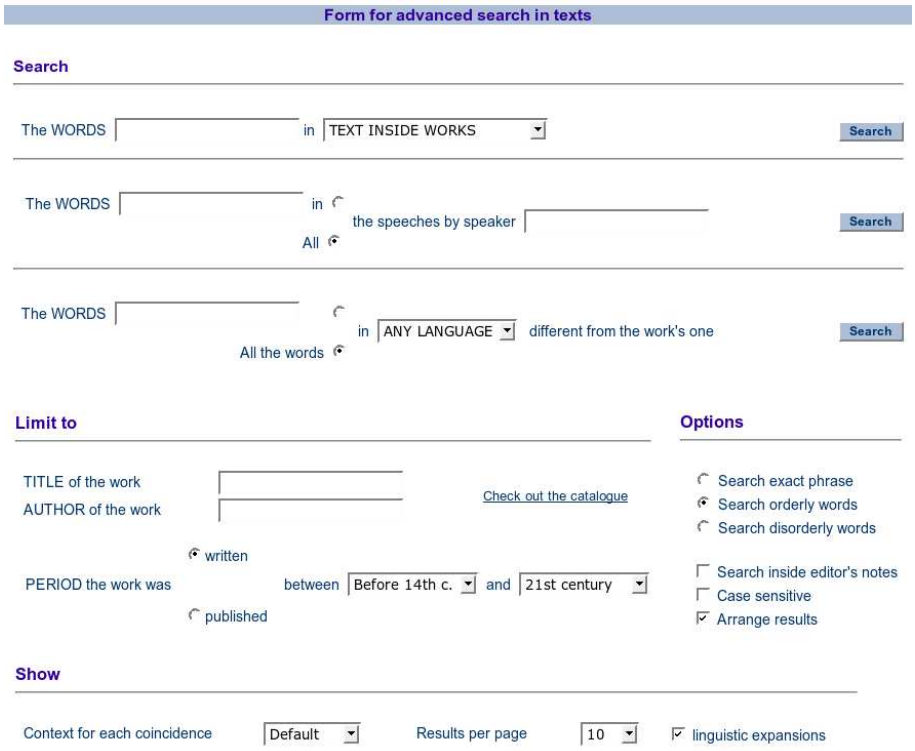

Fig. 2. User interface of the XML search engine running at the Miguel de Cervantes Digital Library.

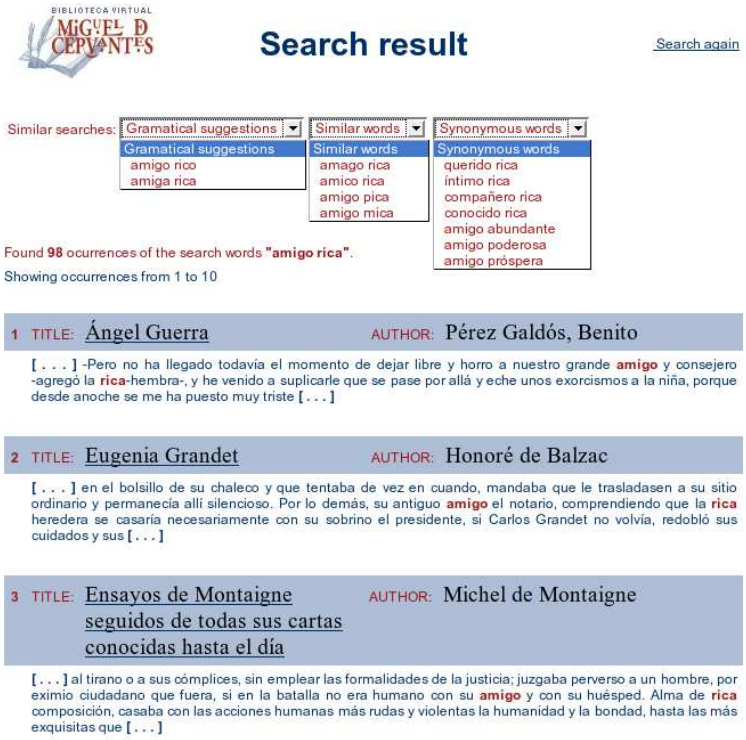

Fig. 3. A sample query result when searching the words "amigo rica".

results from the query servers it reroutes them to the corresponding clients. The user server can directly resolve some of the requests because it manages a query cache.

The system is coordinated by a heterogeneous database, which contains data of several kinds:

- Query servers: Availability, load, remaining query queue, idle time, resolved requests.
- Users: IP addresses, queries requests, waiting time.
- Queries: Users that requested it, query server in charge of solving it, status, number of occurences found, time spent in solving it, result of the query.
- Statistics: Working time, number of queries, average time per query, total size of the query results in bytes, etc.

The user server, acting as a cache, keeps the data of the queries for some time, storing temporarily their result, so that another client posting the same query would be directly answered with the result. Furthermore, these data allow queries that have been enqueued for a long time to be discarded (only in case of an overloaded system).

User data is not only used for the redistribution of the result of the queries, but also as a password controlled access. Moreover, the user server distinguishes between several privilege levels, so that some queries may be restricted to authorized users.

The user server generates a log with all requests served, discarded or denied, and any occured incidence. Furthermore, it manages an on-line statistics generation system, which allows administrators to follow the execution of the whole search engine.

The search engine is controlled only by a single user server that centralizes the generation of statistics and log modules. This centralization allows for a high performance and easeness of maintenance of the whole system. However, the only limit of the performance is the number of requests that the user server can process. In our experiments, only 0.06% of the time needed to solve a query was spent in the user server. Therefore, in practice, the performance will be limited by the number of query servers available.

3.3 Query Server

The query server attends to the query requests received from the user server. The queries are queued and solved individually. After solving a query, its result is sent back to the user server, along with some statistics such as the time spent in solving it.

The process of attending to the queries is divided into two different phases. In the first one, the query is processed to generate a list of occurences. In the second one, the XML files are accessed according to the list of occurences to extract their context[6] and to obtain the result of the query.

[6] The context is retrieved as plain text to preserve the XML format.

A certain range of occurences to show is always specified, i.e, the occurences from 1 to 50, so that the query server needs to access a limited number of files without slowing down the process. The search engine implements a passage information retrieval system (11). Given the structure of the XML documents, the content shown is the same passage[7] in which the occurence was found.

The division between these two phases allows the query server to store the results of the first phase into a cache, so that when the same query with different range of occurences is requested, only the second phase has to be executed again. The second phase takes a significant amount of time, due to the fact that the XML files are stored in the hard disk. After retrieving the contexts, the query expansion generation, which will be explained in section 4.3, is executed.

4 Search Engine Components

The search engine system, which is integrated in the query server, is the responsible of attending to query requests. Its performance relies in a previously generated index obtained from the collection of XML files. This index provides direct access to the words, tags and attributes included in the collection. This kind of index ensures that query solving time grows atmost linearly with the size of the collection of XML files.

Figure 4 shows the structure of modules that compose the internals of the search engine system. The search engine analyzes each query request to create an execution plan. Once the plan is created, queries are processed and expansions are also analyzed to suggest them to the user. The following paragraphs describe in detail the different modules that compose the engine.

4.1 Index Generator and Index Manager

The index generator module is the responsible of indexing the information from the XML file collection, which is related to the exact position of each item type (word, tag or attribute) within the collection. The generation process takes about 15 minutes to finish using a Pentium IV-1.5GHz machine for an XML files collection of 300 megabytes.

The data structure of the index, contains a hash table for each of the different item types, as well as one index file for all of them. This index file contains for each item all the positions within the documents where they appear. Each one of the hash tables will contain a different entry for each item, so that, when one of these items is searched, it is possible to obtain both the exact number of occurrences of the item and the position within the index file where the related information is located.

The size of the set of index files is comparable to the size of the whole XML file collections; no compression algorithm is used. Due to this, it is not feasible to

[7] In our case, the search engine considers some TEI elements (paragraphs, verse lines and citations) as passages (see http://www.tei-c.org).

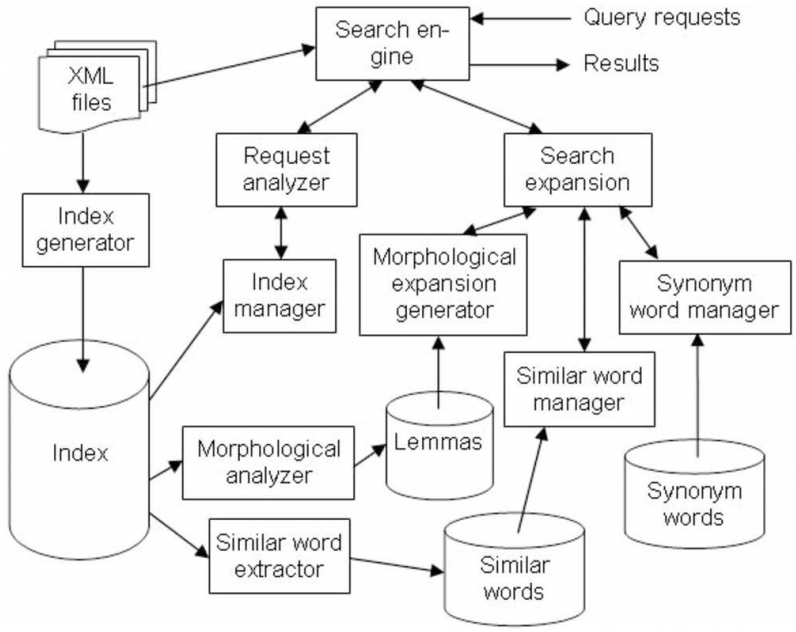

Fig. 4. Search engine internals scheme with all its modules as well as the databases.

maintain the whole index in memory. This design allows to maintain in memory only the hash tables. Therefore, to access the index file at most a few disk reads (of contiguous pages) are needed.

The index manager module exposes a simple interface to search words, tags or attributes, offering also mechanisms to perform nested searches exploiting the XML arborescent structure.

4.2 Request Analyzer

This module is the responsible for studying the query requests sent by the client. It requires access to the index and the XML files to generate the results that satisfy the query. To generate the results, the collection of XML files has to be accessed, and this slows response time.

Digital libraries may have their works available on-line. In such case, it is possible to provide a link to the exact position in the work for each of the query results, so that direct access to the required information is enabled.

4.3 Search Expansion

The task of this module is to suggest further queries similar to the posted ones. This search expansion module relies in three modules: the morphological expansion generator, similar word manager and synonym word manager.

The morphological expansion generator provides access to the morphological analysis of the words in the XML documents. The generation of these analysis is performed right after the index generation process, so that delays are avoided when solving queries. The morphological analysis module that the system uses is part of the Spanish-Catalan machine translation system interNOSTRUM (15).

The similar word manager returns a list of the similar words, that is, words that are ortographically close. Information about all the similar words among the words found in the XML documents is also automatically generated right after the index generation phase. This task improves user experience as it provides semiautomatic correction of typographic errors.

The synonym word manager obtains the synonyms of the most common words contained in the XML documents. These synonyms are stored in a handicrafted database, which is accessed during the query solving phase.

The search expansion works differently depending on the number of words contained in the query. When the user is searching a single word, similar and synonym words are suggested, whereas when more words are contained in the query, these are analyzed to check their gender, number and case (if it is a verb), and the suggested expansions keep morphological concordances.

5 Experiments

It is not easy to automatically evaluate the grade of acomplishment of some of the requirements proposed in section 2, such as efficacy or safety. However, we can analyze if the system meets or not requirements such as information retrieval, query expansion and multimedia features. The speed of the system, which is one of the most important requirements, can be measured in terms of queries solved per second.

In our experiments, we have tried to estimate the speed of the system by emulating a real case. We have built a test set of 10,000 queries, containing the following types of queries:

- 2,500 single word cached queries with large result sets.
- 2,500 complex cached queries with empty result sets, that are not cached.
- 2,500 complex cached queries with small result sets.
- 2,500 single word uncached queries with large result sets.

The experiments were performed running all the servers in the same computer and the system was able to answer aproximately 5.5 queries per second using a XML file collection of 300 megabytes in a Pentium IV 1.5GHz with 1GByte of memory.

6 Conclusions

The global analysis of the architecture reveals that the required objectives are achieved. The architecture offers a high efficiency, given than the user queries

are always processed and suggestions are made. However, the efficacy is very dependant on the web interface given to the application. So, if the users can not express precisely what they want, the efficacy is decreased.

The context of the occurrences of the searched words is obtained using a system based on passages, thus satisfying the information retrieval prerequisite. Furthermore, unlike other XML search engines, direct links are generated to the exact positions within the published documents, which allows the user to access directly to all the information related with the query.

Execution time is very dependant on the implementation, but in theory, it is very low, as new query servers can be easily added to the system. The process time needed by the user server is minimal. Therefore, it will be seldom overloaded regardless the number of users accessing at it. Network speed is essential as it determines the maximum number of requests per second that can be received; if the user server is able to listen to all of them, the global performance will never be limited by the user server.

Safety of the information is quite high, because queries pass through the web server, the user server, and eventually, also through the query server. The user server contains a request register and can hence restrict the access to the resources. Additionally, the query server accesses the XML collection and returns only plain text to preserve the original XML files.

Finally, the search expansion, based on grammatical criteria, words similarity and synonymies, improves the user interaction with the system. Besides, the proposed architecture is ready to easily integrate new services needed in the future, i.e., working with multimedia data that will become usual in the next years.

7 Future Work

At present, we are working on an open source version of the search engine. This new version will be easy to install and integrate into any digital library. The aim is to allow webserver administrators to successfully install `XMLibrary search`, but still allowing them to customize it to fit their needs. Furthermore, the open source version would also allow the indexation of HTML files, after being turned into XHTML files.

In addition, we are researching index compression algorithms that would allow the search engine to be faster, provided that smaller indexes are read faster from the hard disk and decompression time is minimal. Moreover, we are studying the new services that a suffix array based index (16) would allow.

Eventually, the integration of several tools of natural language processing like part-of-speech taggers and machine translation tools would improve the system.

Bibliography

[1] Neumann, A., Berlea, A., Seidl, H.: Fxgrep: A XML querying tool. In: http://www.informatik.uni-trier.de/~aberlea/Fxgrep/. (2000)

[2] Zhao, B.Y., Joseph, A.: Xset: A lightweight XML search engine for internet applications. http://www.cs.berkeley.edu/~ravenben/xset/html/xset-saint.pdf (2000)

[3] Jaakkola, J., Kipeläinen, P.: Using sgrep for querying structured text files. http://www.cs.helsinki.fi/TR/C-1996/83/ (1996)

[4] Katz, H.: XQEngine - XML query engine. http://xengine.sourceforge.net/ (2003)

[5] Goetz, B.: The Lucene search engine: Powerful, flexible and free. http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html (2000)

[6] Noehring, O., Jedlicka, M.: TSep: The search engine project. http://tsep.sourceforge.net/ (2004)

[7] Meier, W.: eXist: An Open Source Native XML Database. In: Web, Web-Services, and Database Systems. (2002) 169–183

[8] Doclinx: TeraXML enterprise search. http://www.doclinx.com/products/ftxml.html (2002)

[9] Liota, M.: Apache's XIndice organizes XML data without schema. http://www.devx.com/xml/article/9796 (2002)

[10] Zakharov, M.: DataparkSearch engine. http://www.dataparksearch.org/ (2004)

[11] Salton, G., Allan, J., Buckley, C.: Approaches to Passage Retrieval in Full Text Information Systems. In: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. (1993) 49–58

[12] Convera: Convera Retrievalware. http://www.convera.com/ (2004)

[13] Croft, W.: What do people want from information retrieval? http://www.dlib.org/dlib/november95/11croft.html **D-Lib Magazine volume 1** (1995)

[14] Baeza-Yates: Proximal nodes: a model to query document databases by content and structure. ACM Transactions on Information Systems (TOIS) **Volume 15, Issue 4** (1997) 400–435

[15] Canals-Marote, R., Esteve-Guillén, A., Garrido, A., Guardiola-Savall, M., Iturraspe-Bellver, A., Montserrat-Buendia, S., Ortiz-Rojas, S., Pastor-Pina, H., Pérez-Antón, P., Forcada, M.: The Spanish-Catalan machine translation system interNOSTRUM. 0922-6567 - Machine Translation VIII (2001) 73–76

[16] Manber, U., Myers, G.: Suffix arrays: A new method for on-line string searching. In the first Annual ACM-SIAM Symposium on Discrete Algorithms (1990) 319–327