Developing components for distributed search engine ObjectSpot

ZDENĚK DRBÁLEK, TOMÁŠ DULÍK, ROBERT KOBLISCHKE
Faculty of Applied Informatics
Tomas Bata University in Zlín
Nad Stráněmi 4511, 76005 Zlín
CZECH REPUBLIC

Abstract: - The development of components for Search Engine ObjectSpot realized within the frame of iCamp.eu. This paper deals with development and implementation of following components: Spell-Checker, PDF/DOC full-text search and content analysis, results grouping / clustering using Search Result Clustering (SRC)

Keywords: - ObjectSpot, iCamp, Spell-Checker, PDF fulltext searching, DOC fulltext searching, result grouping, SRC

1 Introduction

The iCamp is a research and development project funded by the European Commission under the IST (Information Society Technology) program of FP6. The project aims at creating an infrastructure for collaboration and networking across systems, countries, and disciplines in Higher Education. [1]

From the user point of view, at the first glance ObjectSpot looks like Google, but in fact, the difference is huge: Google searches through publicly accessible web sites, while ObjectSpot searches in various digital libraries that have restricted access for the general public.

This paper deals with development of components for ObjectSpot search engine. Development of ObjectSpot is in early stage, however at this time it has many features already implemented. The ObjectSpot is internally a web-based federated search client plus middle-ware mediator for distributing queries and collecting results to digital libraries and learning object repositories that implement the Simple Query Interface [6] (SQI). The latest version of object spot can be found at [2].

The ObjectSpot inner structure is designed as follows. There is central portlet, which passes search queries to a mediator and mediator then tries to connect to each of the active repositories asynchronously via SQI calls. The repository targets then return SOAP messages containing search results wrapped into RSS.
Before the mediator returns the results to the ObjectSpot portlet, it parses these results and computes a relevance rank using a precomputed inverse document frequencies. These rank scores are then utilized by the portlet to insert the new results at the appropriate rank position. [3]

This paper is organized as follows. In Section 2, a Spell-Checker component and its development is described. The Section 3 introduces PDF/DOC searching functionality, conversion, data storage and gathering metadata. The Section 4 contains a brief presentation of Search Result Clustering (SRC). Finally, Section 5 offers the conclusion remarks.

2 Spell-Checker component

In general, a Spell-Checker is a software program designed to verify the spelling of words. The Spell-Checker helps a user to ensure correct spelling, while suggesting corrections for wrongly spelled words. Spell-Checkers are either stand-alone applications capable of operating on a block of text, or they can be a feature of a larger application, such as a word processor, email client, electronic dictionary, or search engine.[4] The Spell-Checkers are commonly used in many search engines such as Google or Yahoo. One of the major reasons that support the idea of Spell-Checker implementation could be the fact that according to some studies, over 15% of search terms entered by users are misspelled.
The solution that was used for the ObjectSpot purpose is based on ”aspell” unix console utility and implemented as another feature to the mediator. The algorithm itself only checks aspell and gives its suggestion for entered term, no matter if the misspelled term has been searched already. In fact,

more advanced version of spell-checker was implemented earlier. That method suggested the term if it was searched more often than the misspelled word. After long-term tests, we found that this approach was not optimal and therefore we rolled back to previous version that is in the production mode nowadays and is described at [2].

The current implementation of Spell-Checker seems to be stable and reliable. However in the next stage multilingual support should be considered. The complete source code of this component can be reached at [5].

3 PDF/DOC search engine component

The PDF/DOC search engine component is in early stage of development. However, it is already capable of indexing PDF/DOC file formats and storing them into database for later use. The component consists of two parts (subcomponents). The first part is the indexer, which converts PDF/DOC files to plaintext, casts the result into a ts_vector [7] and inserts this vector into database. That is much more efficient and brings capability of fulltext searching. The second part is just a simple web application that connects to database and searches through all database records, trying to find the search term. For better understanding, see the figure showed bellow.

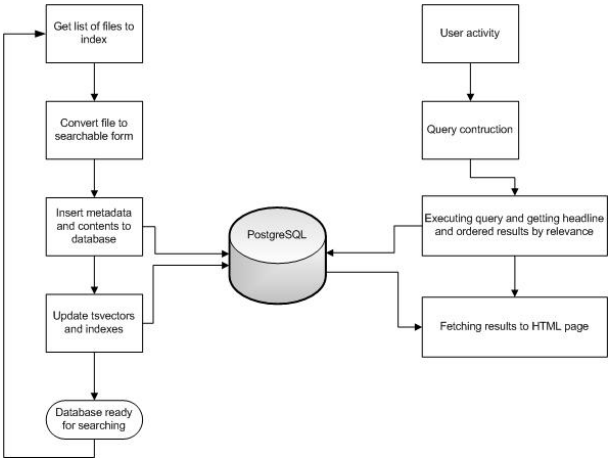

Fig. 1 – Fulltext search system flowchart

All the processes in the left part of the Fig. 1 are running periodically within the definite time interval. On the right side of the figure, there is the querying part, where the fulltext queries for database are constructed and executed and the result is presented to the user.

As long as the development of this component was finished recently, the component will be deployed on our server [8] and complete source code can be found in iCamp project SVN repository at SourceForge.net.

The algorithm has several configuration parameters. The configuration consists of database config section, File system config, Other config and Daemon config.

Example of current release is presented on the next figure.

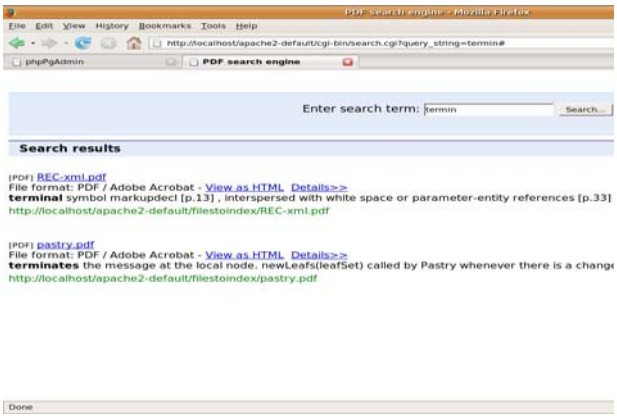

Fig. 2 – Simple view of search engine prototype

Gathering metadata

Another task was the implementation of metadata gathering. In other words that means getting information like author(s), modification date or document outline from the indexed documents. Since getting the document outline from the plaintext is complicated, it is much easier to use the “pdftohtml” tool [9], which tries to parse the document and in most cases it obtains well-formed “document outline” as result. After that, the metadata gathering algorithm goes through this result, finds the document outline and presents it to the user. Some experiments have been done with the “pdftotext” tool, which is normally used for conversion PDF files to raw text, but there was an issue when the structured PDF was created using styles (headings, other style types, etc.), the command converts all this structures to the basic formatting types like font size, font style, bold italic, etc. For that reason it was not possible to reliably distinguish the headlines from normal text in paragraphs.

In future development, there should be no problem to get the whole plaintext, parse it to tokens, then filter stop-words and display the keyword frequencies. The top *N* words could be considered as keywords. Still the most reliable way is force the user to fill all the necessary metadata required. The problem is that many PDF document creation programs do not fill the metadata in default configuration

Further development

In this section, the future development objectives are presented. More time should be spent on these points in order to turn this system to real production state.

- Test the system with much more documents
- Experimentally set the algorithm parameters
- Support word frequencies (keywords)
- Find some other technique for getting metadata
- Optimize source code, make it more reliable
- Fit and implement our PDF/DOC search component into the ObjectSpot mediator
- Implement a "web spider" component, which would be able to crawl through the internet and find the PDF/DOC files automatically without the need of storing them locally
- User-friendly daemon process termination
- More advance technique to distinguish file formats (currently, only file extension is used for this)

4 Search result clustering (SRC)

In this section, we will discuss search result clustering (SRC) [10] and measure the performance of individual grouping algorithm based on existing open-source solution. Not much programming work has been done here in this part. This part is aimed mainly on using existing system that is based on Carrot2.
Carrot2 [11] is open-source Search results clustering engine. It implements all the functionality that is necessary to group search results and to classify them into categories. In addition, some very advanced features like Lingo3D are supported. Lingo3D [12] is the third generation of high-performance document clustering engine featuring hierarchical clustering, ontologies, synonyms and advanced tuning capabilities.

Algorithms description

Algorithm	Speed			Supports hierarchies
	100	200	400	
FuzzyAnts	2.17	8.70	16.93	Yes
HAOG-STC	0.04	0.11	0.28	Yes
Lingo	0.34	0.52	0.84	No
Rough k-means	1.38	6.76	27.73	No
STC	0.04	0.1	0.23	No
Lingo3G	0.03	0.06	0.13	Yes

Table 1 – Grouping algorithm overview

Implementation example

The following figure shows the Carrot2 user interface and the grouping of the results into clusters.

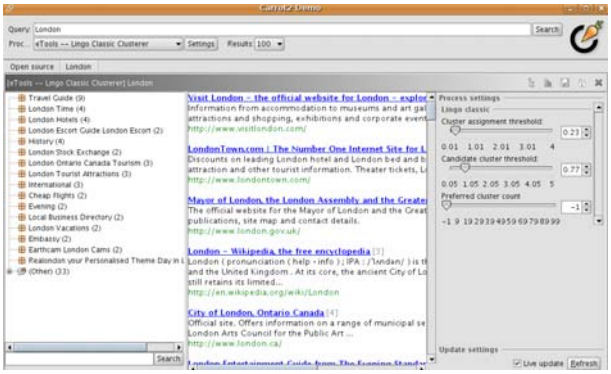

Fig. 3 – Example of result clustering

Note that different parameters can be defined for different algorithms. On the figure above the Lingo classic Cluster is selected. In the left panel, different clusters can be seen. Those categories represent the output of the algorithm. The Lingo supports hierarchical structures also, so if the appropriate term is entered it can create a tree with subcategories.

Further development

Certainly, the clustering of search results is whole field of study. Carrot2 brings very advanced technology that is dealing with CRS. There is also the API written in Java. That means that it should not be a problem to build own application that would meet the requirements based on this engine.

5 Conclusion

Three functional components have been presented in this paper. They are usable and helpful for wide range of users bringing them comfort they demand. It helps them firstly to check their inputs and in case of misspelling correct terms in relevant way. Secondly, it brings more documents to be indexed and so bring more relevant results and finally it helps user with organizing search results by grouping them to thematic categories which will save much time to user.
All the components were more or less well implemented and at this stage, they are tested to be able to work in production modes.
Note that the source code is placed on the iCamp project SVN repository at Sourceforce.net.

References:

[1] Crossing the Border to the Future of Education [online]. ICamp, 2004 [cit. 2008-04-17]. Available on WWW: <http://www.iCamp-project.org/>

[2] Deployed implementation of ObjectSpot available on WWW: http://teldev.wu-wien.ac.at/mediator-2/portlet/

[3] Robert Koblischke, Inside ObjectSpot [online]. ICamp, 2007 [cit. 2008-06-20] Available on WWW: <http://www.icamp.eu/2007/10/10/inside-objectspot/>

[4] Spell checker [online]. 2001, 15 May 2008 [cit. 2008-05-16] Available on WWW: <http://en.wikipedia.org/wiki/Spell_checker>

[5] Source code of Spell-Checker available on WWW: <https://icamp.svn.sourceforge.net/svnroot/icamp/mediator-2/trunk>

[6] Bernd Simon, David Massart, Frans van Assche, Stefaan Ternier, Erik Duval, Simple Query Interface Specification [online] 2005-04-20 [cit. 2008-05-16]. Available on WWW: <http://www.icamp.eu/wp-content/uploads/2007/05/sqi_v10beta_2005_04_13.pdf>

[7] Teodor Sigaev, Full-Text Search in PostgreSQL: A Gentle Introduction [online]. 2007 [cit. 2008-06-21]. Available on WWW: <http://www.sai.msu.su/~megera/postgres/fts/doc/fts-overview.html>

[8] PDF/DOC search engine – deployed at available on WWW: <http://zamestnanci.fai.utb.cz/~drbalek/pdfdocsearch>

[9] PDF2HTML project available on WWW: http://sourceforge.net/projects/pdftohtml/

[10] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma,Jinwen Ma, Learning to Cluster Web Search Results [online]. 2004-09-05 [cit. 2008-05-26]. Available on WWW: <http://research.microsoft.com/users/hjzeng/paper/p230-zeng.pdf>

[11] Carrot2 project available on WWW: < http://project.carrot2.org/>

[12] Lingo3D project available on WWW: < http://company.carrot-search.com/lingo-applications.html>