Open Source Search and Research

Michel Beigbeder
École Nationale Supérieure
des Mines de Saint-Étienne
158 cours Fauriel
F 42023 SAINT ETIENNE
CEDEX 2
mbeig@emse.fr

Wray Buntine
Helsinki Institute for
Information Technology
(HIIT)
PL 68, 00014, University of
Helsinki
Finland
Wray.Buntine@hiit.fi

Wai Gen Yee
Department of Computer
Science
Illinois Institute of
Technology
10 W. 31st St.
Chicago, IL 60616 USA
yee@iit.edu

ABSTRACT

In this paper, we present a review of criteria for the evalu-ation of open source information retrieval tools and provide an overview of some of those that are more popular. The question of interaction between research and availability of open source search tools is addressed.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Open source

Keywords

Information retrieval, Open source software, Search tools

1. INTRODUCTION

The development and deployment of open source software packages have had sustained popularity recently with their increased quality and strong communities. Moreover, in virtually every information system domain, some open source software package exists, some so popular that they have achieved a significant level of deployment in both private and public organizations (e.g., the Apache Web Server[1], MySQL[2], Lucene[3]). This is certainly the case in the information retrieval domain; there is a large diversity in the goals and the features of the different available tools.

We recognize there are four basic categories of information retrieval software. The first category is software to be used in the library context, where some metadata about the documents are available rather than their full text. The second category has been developed for academic purposes, for the study of information retrieval. In the third category aretools for individual Web site or small to medium scale Web search. In the last category are libraries of software that are easy to tailor or to expand in larger search applications.

As with every classification scheme, this four-way split is not perfect and the classes are not completely disjoint. The scheme considers the use or deployment of the information retrieval tools. Other schemes could be considered. For instance, in the academic context, the matching model could be the first classification key. In fact, there could be as many criteria as there are comparable features in the software. In this paper, we survey some characteristics of information retrieval tools that could be used to classify and compare information retrieval software.

We will also address the question about the interaction between the development of open source information retrieval software and research in the information retrieval field. In particular, we will lay some emphasis on the open source aspect. We will illustrate this interaction with the example of open source Web search engines and the question of the deployment of such a tool.

2. HISTORY

The first and still very significant application of computers for information retrieval was in replacing and then enhancing the card catalogs in libraries. Here the contents of (text) documents are not available online, but are represented by some metadata. Typically, these metadata include "objective" information, such as author names, title, publisher, publication date, etc. Most often, they are complemented by some "subjective" metadata which describe the content of the documents. For this part of the cataloguing activity, librarians use either controlled languages such as those provided by thesauri, or uncontrolled language, in which case the librarians freely associate some chosen keywords to the documents. Within this librarian context, the predominant information retrieval model is the boolean one.

With the development of computing power, particularly with regards to storage, many documents became available online with their full content. Moreover, the emergence of computer networks and their rapid deployment made it possible to share, exchange and disseminate the documents either at an organizational level or world-wide. New informa-

[1] `http://httpd.apache.org/`
[2] `http://www.mysql.com/`
[3] `http://lucene.apache.org/`

tion retrieval models were developed to take into account the availability of the full text of the documents: the vector space model [3] and the probabilistic models [4], to name the large categories.

The next major step for information retrieval was the rise of the World Wide Web. Once again, as soon as this information space existed, some retrieval tools were developed. These adopted the same broad approaches (Web directories, full text retrieval), but new techniques were needed to deal with the new scales of both collection size and result set size, e.g., *a priori* or query-independent document rankings are helpful when tens of millions of documents potentially match a query, and to deal with the new complexities of document structure and document linkage.

Retrieval in multimedia domains is now a dominant commercial concern in search. For the text world, however, the latest challenge is posed by large online repositories of content from books, created for various reasons at Google and Amazon. Considering the Gutenberg Project (`http://www.gutenberg.org`), online text repositories are by no means new, but their commercialization is.

3. CRITERIA

3.1 General level

The first points to keep in mind when choosing any software are the broad computing and software issues such as:

Installation requirements: such as operating system, prerequisite libraries (WWW utilities, HTML processing, databases, etc.), and so forth. Some popular packages use Java to skirt these issues. Hardware can be a key issue if parallel computing is desired for improved performance.

License: some licenses of tools allow non-commercial use only, and open source licenses do differ.

Documentation: entry for new users requires several levels of documentation from tutorials to reference manuals. Nutch, for instance, maintains a Wikipedia page to support this. Documentation needs to support the *software installer* (person installing software) as well as the *system configurer* (person designing the capabilities of the specific search engine) and the subsequent *user community* (the user interface and the querying modes).

Community: according to Nutch founder Doug Cutting, the user and developer communities are the lifeblood of a project, and the two groups co-mingle. A strong community engenders a good development project.

Extensibility: search is a complex area so one size does not fit all. A readily extensibly system allows adaptation to specific requests while a good community supports this process.

3.2 Efficiency level

Two points of view must be considered with regards to information retrieval software. The first is about *effectiveness* and the second about *efficiency*.

Effectiveness is mainly addressed by the information retrieval research community. Many information retrieval research papers that present a new model or some variations to a known model finish with an experiment in which the proposition is tested in a well known context. Moreover there exist now many information retrieval competitions in which different systems are evaluated. TREC[4], which has been organized annually since 1992 by the National Institute of Standards and Technology, was the first, followed by NTCIR[5], CLEF[6] and INEX[7].

Efficiency is concerned with the usage of computer resources both at indexing and query time. Of course the algorithms and their complexity are the key factors for this point. Traditional algorithms must be tailored to the information retrieval application because its requirement of extensive interaction between main memory and disk requires more complex cost metrics. Computational benchmarks measure essential characteristics such as indexing time, index size, query time. The first two are related to the size of the document collection, both in terms of overall size and of the number of documents. The last depends on the query model, but it could be related to the number of keywords in the query.

The last point about efficiency is the scalability of the software. There are always some limits, and they are not always explicit. Some Linux installations will have a 2Gb file limit, or the indexer fails to work on network-mounted directories. The scalability of the software is to be evaluated in terms of the number of users and the throughput of queries, and in relation to efficiency to the number of documents and the overall size of the collection. Moroever, some systems support incremental indexing of documents, sometimes without shutting down the runtime. Efficiency and scalability are often compromised by incremental indexing support.

3.3 Corpus level

This level is concerned with localization of the documents, gathering and grouping them on a local store for the search engine.

When the first test collections were created for the purpose of evaluating information retrieval systems, some *de facto* conventions were defined. The common usage is to have many documents (all of them if possible) in one file separated with some tags. The software *smart* 5 used tags with dots in the "first column" as this was quite common in many applications related to text processing at this time. While everything was configurable within the software *smart*, the usage convention was to indicate the documents with the tag `.I` followed by the document number, the authors' names, introduced by `.A` and so on. The TREC competitions used tags with an XML-like syntax, while the Web tracks in TREC completed it in some way with the HTTP syntax.

Developed to ease the evaluation task, this kind of organization is not applicable to many applications as it is rarely the case that many documents are stored in a single file. Some software rely on a file with pointers (e.g., pathnames) to the files to be indexed. With such a scheme, some scripts have to be created for the gathering step.

Files can also be gathered given some criteria and a data source. The most simple form is to give the pathname of one directory, and the gatherer recursively descends the hi-

[4] `http://trec.nist.gov`
[5] `http://research.nii.ac.jp/ntcir/`
[6] `http://www.clef-campaign.org/`
[7] `http://inex.is.informatik.uni-duisburg.de/`

erarchy of files and directories under the given path. Some sophistication could added to this scheme to filter the file names to be indexed (or not indexed) by some kind of regular expressions.

The most sophisticated policies for gathering take into account distributed documents. In this case, the files to be indexed are designated with URLs. If not all the files are to be individually designated, a kind of crawling has to be done. This is only possible of course if the format of the documents allows one document to point to another one (for now, only HTML and pdf files have standard syntax for such designations).

These three categories of document gathering are in fact related to the intended use of the software. The one-file-many-documents scheme is particularly useful for effective experimentation in the evaluation competitions, and hence is aimed at research. The second one in which the documents are gathered on a single computer is aimed at desktop search. The last one where documents are fetched from Web servers is dedicated to site and/or Web search. Benefits could arise from the integration of these different gathering methods. For example, desktop applications could benefit from techniques of the first category. One important example is the mail boxes which contains many e-mails in a single file. Other examples are the archive files which also are eventually compressed (.tar, .tgz, .zip, etc.) in which one could be interested in indexing/retrieving not the archive file but one of the embedded files.

Another point is the format of the files that are indexable. Many software packages can only deal with plain text. But some allow filters that convert other formats (Postscript, PDF, MS Word files, HTML, etc.) to plain text. The most flexible ones are designed for the insertion of plug-ins to deal with a particular format. In this case some development has to be done if "exotic" file formats are to be dealt with. It should be noted that PDF, MS Word, etc., are complex formats that require sophisticated extraction if any structure such as tables and section titles is to be preserved

3.4 Lexical level

Many information retrieval models are based on the extraction of features from the documents. In many European languages, these features are what is commonly called *words*. The lexical level is the definition and implementation of the splitting of the input text in such features.

The lexical aspect is very important if the deployment is to be done in non-English languages. It is generally viewed that internationalization of code is a task that needs to be addressed from the ground up rather than as an after-thought. Some software packages are not even able to deal with 8-bit character sets. On the contrary, some of them are able to deal with the Unicode character set.

Besides the character set, the definition of the tokens that are to be indexed is one key point in the adaptation of a software package to a language. Many software packages rely on some ideas about words and it is rarely the case that they are configurable regarding this point. Even in English-like languages, other problems that do not have a universal answer are the processing of some non-letter characters: digits in relation to numbers and/or identifiers, dots and commas in relation to numbers, dots and hyphens in relation to words (e.g., "MS DOS" versus "MS-DOS", hyphens at end of lines), sometimes we should be interested in the removal of some whitespaces (e.g., "Mac OS X" versus "MacOS X"). Of course the next step at the lexical level would involve effective Natural Language Processing tools to extract the lexical units and in particular the named entities. Other interesting possibilities explored in the research community but that have not yet been developed in open source are related to the use controlled vocabularies or of thesauri. So tokenization, especially in very heterogeneous environments, still is a research area in information retrieval. The best for flexible software is to allow for some plug ins related to that point.

The n-gram model is rarely taken into account though it has some important applications either for some languages, or for OCR collections where there are many misspellings, or even as an alternative to the word level of tokenization in languages like Chinese.

The well known "stop list" and lemmatization phases are often present in the tools. But these features are dependent on the languages so their applicability to the intended corpus should be verified and alternatives may have to be developed.

3.5 Structural level

The structural level is perhaps the most complicated and least flexible one because of its complex inter-relationships with many other levels. In particular, if the query model has to deal with structure (note the CAS (*Content And Structure*) topics in the INEX competitions) the documents must fulfill some requirements with regards to their structure and the whole matching process has to take it into account. In this section we will review the structure families that are dealt with in the information retrieval community.

The first level of structure is the text level itself. Though most often it is not taken into account by the information retrieval models as many of them only rely on the number of tokens in the document and do not take into account their global or relative positions. The exception, related to the query model, is either models where ranking is based on the proximity of the query terms, or the phrase search and its extension with proximity predicates in some boolean extended models.

The second and oldest one is the structure of card catalogs previously mentioned. This structure is related to the metadata models used in the library catalogs.

The third one is that of hierarchically structured documents. The hierarchical structure is usually the structure referred to when speaking about "structured documents". It is now quite widespread with the emergence of XML, but its root are in its wide use in many traditional forms of writing: chapters, sections, paragraphs, etc., are quite common. These forms are implemented in some formatting tools (LaTeX for instance) but are implicit in other ones. Its relations to information retrieval are very diverse: passage retrieval was one of the first uses, though passage retrieval can also be applied with only text structure and no hierarchical structure.

The last one is that of hypertext. This structure has received much attention since the explosion of the Web. Besides its use at the gathering level as mentioned before, it has many applications in information retrieval.

3.6 Query level

The query model is tightly related to the underlying in-

formation retrieval model implemented in the software.

One of the first query models developed in the information retrieval domain was the boolean one because of its relation to the card catalog systems. It has always had wide usage in this application. It is also the preferred one when large recall is desirable, as for instance in legal applications. It seems that many end-users have difficulties in using the boolean query model as the common meaning of the words "and" and "or" is different from their technical boolean meaning.

Moreover the users are now accustomed to Web search engines, which quite exclusively offer a simplistic query model where only few keywords can be entered. As extensions to the basic set of words, some of them take into account the order in which the keywords are given. In a such a case, the query model is a list of words.

A frequent variation of the query model is the possibility for the user to enter phrases, usually by enclosing them in double quotes. The relation to ranking is not obvious, but as a first step it acts as a filter on the document to be retrieved as only those that contains these phrases are to be ranked. In the same idea, proximity operators were introduced to extend the boolean query models. The intended use for them was to relax the phrase constraints.

At first glance, after the set of terms, the next level in the query model is only a variation in quantity, where the limit in the number of words is higher, so that short exerpts of text can be given as queries. In such a case, it is highly possible that a given keyword is used many times, and thus the count of the word occurrences is now a clue for ranking.

Lemmatization is a traditional lexical level tool used at indexing time. An alternative to lemmatization is to allow the user to use wildcards, though the effect on ranking is perhaps not exactly the same with the two methods.

Less widespread query models known as flexible query allow the user to weight either its query terms or some boolean like operators.

Finally, any combination of the previous functionality can be designed in an information retrieval model. Though most software is designed for one or several models, their combination could require the development of new algorithms/techniques that effectively integrate their strengths.

3.7 Ranking level

A ranked list of results is the main output of many information retrieval algorithms. This is the basis for virtually every effectiveness measure as well as search engine interface. So the quality of the ranking and the models on which it is based are fundamental. Much information retrieval research is related to that aspect and it results in many models, variations, and tuning. All major ranking methods are implemented in at least one open source software package: vector space, two-poisson, divergence from randomness, etc.

The features used are diverse but the predominant ones are the frequency of terms in documents and the document frequency of the terms. Other features are the length of the documents or other statistics about the distribution of documents. All of this information is captured at indexing time. But some other information is necessary for their implementation. For instance, keeping the position of the word occurences allows the implementation of proximity or phrase search. Moreover, ranking can fully be based on these positions.

3.8 Interface level

Many of the information retrieval models and their implementation in the tools return a list of results, and ouputing the results consists of displaying this list of some part of it. For applications of the tools in the evaluation tasks of campaigns such as TREC, only the formatting of this list is important. But for actual applications for final users, the output has to be tailored. While this can be also done with some output formatting, it could be more or less easy. Another point is that these applications could need other functionnalities. One of the most well known in Web search is the presentation of results by pages with, say, 10 results per page and some snippet of the document per result. This functionnality could have some repercussions on the internal algorithms, especially if efficiency is to be considered.

4. SOFTWARE PACKAGES GOALS

As said before software could be classified with respect to many criteria. Here we present some key points on some software packages, and we choose to group them by their main usage. We found that three main usages were able to capture practically every software package: Web search, digital library search and research usage. A small set of software tools are either too general to fit in only one category or, on the contrary, do not fit in any of them, so a miscellaneous group was made.

In this section, we describe some of the open source tools available for information retrieval. In considering these tools for usage, considerations must be made for their functionality, their support, and their licensing agreements. Functionality is one of the first characteristics users consider - they must match the user's needs. Related to functionality, however, is the performance of the system as well as its scalability.

Support for systems is likewise important. A system with a large development community is likely to be able to help in tailoring it to the user's needs. It also ensures that existing features are constantly improved and new ones are added.

Finally, licensing restrictions must be considered. In general, licenses protect the publishers from any responsibility in the performance of the product. They also govern how systems could be used. For example, some licenses require all systems that incorporate the open source to be also open. The most common licenses are Gnu, Mozilla, Berkeley, and Apache.

4.1 Web site oriented

- **dataparksearch engine** `http://www.dataparksearch.org/` DataparkSearch is an GPL-licensed open source system for indexing and searching a Web site, group of Web sites, intranet, or local system. DataparkSearch is built on top of a relational database, which must be installed separately.

- **ht://Dig** `http://www.htdig.org/` Similar in functionality to DataparkSearch, htDig does not seem to be supported.

- **isearch** `http://www.etymon.com/tr.html` Isearch is a text retrieval system that supports full text searching, result ranking, Boolean queries, and the Z39.50 client-server protocol. Originally developed in 1994

and adopted for some high profile applications, the original version is no longer supported. However, derivative systems are still available.

- **mnoGoSearch** `http://www.mnogosearch.org/` mnoGoSearch is a popular indexer and retriever that is built on top of a relational database. Users can interface with the search system via standard CGI or PHP.
- **namazu** `http://www.namazu.org/index.html.en` Namazu is a full-text search engine written in Perl. It supports Web indexing, but is primarily intended for desktop search. Namazu has extensive Japanese language support.
- **OpenFTS** `http://openfts.sourceforge.net/` OpenFTS is a full-text search engine built on top of PostgreSQL and written in Perl.
- **swish-e** `http://swish-e.org/` Swish-e is a popular indexing and search system for small repositories (fewer than a million documents). It is appropriate for both local directories and Web sites, and can index several types of files with its extensive filtering mechanism.
- **swish++** `http://swishplusplus.sourceforge.net/` Swish++ is a C++ rewrite of Swish-e and can be compiled in Windows using the Cygwin Linux emulator.
- **glimpse/webglimpse** `http://webglimpse.net/` Glimpse/Webglimpse is a popular search engine/crawler package. It supports result ranking, fuzzy matching, and content filtering.

4.2 Digital libraries oriented

- **cheshire** `http://cheshire.lib.berkeley.edu/` The Cheshire project's goal was to develop a next-generation digital library that bridges the gap between bibliographic and full-text searches. Specifically, it addresses the exclusive problems of search failure and information overload using "advanced IR techniques." Metadata are stored natively in XML.
- **zebra** `http://www.indexdata.dk/zebra/` Zebra is a search system built for structured data (e.g., email or XML). It can handle large data sets (in the gigabytes). Zebra supports the Z39.50 client-server protocol.
- **mg** (version 1.3g) `http://www.nzdl.org/html/mg.html` The MG system was developed in conjunction with the *Managing Gigabytes* text book [5]. Originally designed to be space efficient, the New Zealand Digital Library distribution contains many improvements, such as improved stemming and index merging, as well as support for Windows platforms.

4.3 Academic research oriented

- **lemur/indri** `http://www.lemurproject.org/` Lemur is a set of language analysis and text retrieval tools designed jointly by the University of Massachusetts and Carnegie Mellon University. Indri is an associated search engine. A major focus of the Lemur/Indri project is specialized support of structured queries and documents.
- **mg** (version 1.3g) `http://www.nzdl.org/html/mg.html` See above for a discussion of MG.
- **smart** `ftp://ftp.cs.cornell.edu/pub/smart/` *Smart* is a legacy information retrieval system that was developed in the early days of information retrieval research by Gerard Salton at Cornell University. It is well-known for contributions such as the vector space model.
- **terrier** `http://ir.dcs.gla.ac.uk/terrier/` Terrier is a project from the University of Glasgow. Terrier is a search engine written in Java that is built for scalability and incorporates several novel ranking algorithms.
- **zettair** `http://www.seg.rmit.edu.au/zettair/` Zettair is a search engine developed at RMIT University for indexing and searching HTML and text. It is written in C and has been tested on several platforms.

4.4 Miscellaneous tools

General libraries/tools

- **lucene/nutch** `http://jakarta.apache.org/lucene/docs/index.html` Lucene is a popular search engine API developed by Doug Cutting and supported by the Apache Software Foundation. While Lucene implements several function important to information retrieval, it is not an indexer or search engine. The associated Nutch system is built on top of Lucene and implements search engine functionality. Nutch/Lucene is popular in industrial applications due to its performance.
- **xapian/omega** `http://www.xapian.org/` Xapian is a search engine API developed by the former employees of the now defunct BrightStation, PLC. Xapian supports the probabilistic information retrieval model. Omega is a search engine built on top of the Xiapian library.

Desktop oriented

- **wumpus** `http://www.wumpus-search.org/` Wumpus is a search system for file systems, developed at the University of Waterloo. It is designed for an environment where there are relatively many updates to the underlying documents versus queries.

5. RELATION TO RESEARCH

We will now address the question of relations between the development of information retrieval software — and in particular with an open source license — with research in the field of information retrieval.

The academic research in the information retrieval domain has a long history as described in Section 2. And there is a strong emphasis on experiments with (as much as possible) *actual* data because of the fuzzy aspects of the field itself in particular with the crucial problem of *relevance*. So many experiments are conducted by the teams involved in the field. To design these experiments, some material has to be used. Concerning computer science, this material is of course composed of programs and data.

The data used for information retrieval experiments are the test collections. They are composed of three parts. The

first one, the largest in size, is a collection of documents. They are in some sense *corpora* as they are built by a single organization with constraints that result in some homogeneity in the collected documents. For instance some of them are labelled with the name of the publisher or the journal title from which they were built. Regarding homogeneity the exception could be the Web based collections, but even in this case there is some homogeneity as all documents share at least their origin with some common characteristics tied to the hyperlinks. The second part of a test collection is a little set of information needs (i.e., a set of queries) – about fifty of them. This first two parts are not very difficult to build: the first one is tightly related to common computer usage which resulted in the creation of many documents; the second one is small. The third part consists in the relevance judgments and is the most expensive to produce. Ideally, a person would consider each information need, and rate each document's relevance to it. Such an ideal is impractical to achieve as the document collections could be large (in the millions), and therefore, only approximate relevance judgments are generally available. Because of the large amount of effort necessary in building a test collection, and also because sharing of data is important for comparisons, test collections are built within the framework of evaluation competitions. As such the data produced are free [8], but their legal usage, defined in their license agreements, generally restrict their use to internal evaluation of information retrieval systems.

The second point is the use of these data to feed programs. Of course each team should be able to develop its own program to implement its ideas. In fact the development of straightforward programs to deal with a small collection of documents is quite easy. In practice the size of the collections is so big that these straightforward programs are not usable in terms of efficiency or not runnable at all. Moreover developing systems from scratch is time-consuming. This is particularly true in information retrieval where, at the base, we are feeding documents to a tokenizer, building a dictionnary of tokens, and an index to the documents which contain the tokens. So if the team's work is not particularly on algorithmic aspects about these basic tools, reusing existing software is most often the best choice in terms of efficient time allocation. Furthermore, by working with well-known systems, others can test out any enhancements that are made.

The first information retrieval software available in source form was *smart*. The project started in 1961 by one of the research teams who worked much to the development and the application of the vector model. Several implementations of this in a tool were developed to follow the evolution of computers and operating systems. This software was disseminated before the formation of the modern open source software community which resulted in the definition of software licenses — GPL, the Gnu Public License being one of the most well known. As this software was made available in its source form, many teams of the information retrieval community used it in many experiments at such a level that conferences and books were devoted to its use in academic research. *Smart* gained so much importance in the development of the information retrieval field that it is necessary to know it to understand some research papers. Thirty years after the starting of the project, a panel was dedicated in 1991 to this system in the SIGIR conference [2]. It is an example, if not the first one, of the interaction between the availability of software in a free way and academic research.

But the *smart* software is no longer distributed as an open source software and the last release is from 1992. Other teams involved in information retrieval developed open source software. The MG system was developed by an Australian team in conjunction with a text book. Three versions between 1994 and 1996 were made availbale by the team. Some extensions were made by one of its users, the New Zealand Digital Library. With both of these examples, it can be seen that maintenance in the long term of software soley by the academic community cannot be achieved. Maintenance was also reported as a problem for an academic team by the developpers of more recent academic open source search software in the OSIR workshop [6].

Beside the academic teams, the open source software community is the other community involved in information retrieval open source development. It should be noticed that the goals are not the same for open source software and for academic software. Academic teams mainly develop the software for testing their ideas and open source system developers want good and flexible code. Moreover the community of developpers has to be sustained by a community of users. In many cases there is an overlap between these two communities, as this was the case for the later development of MG as mentionned before. How a user community gains traction is related to the software quality but also to many other aspects that are as difficult to understand as those that make a song becoming a hit. The existence of the community of users is as important as the software quality for a wide acceptance and deployment. This is this community that is able to ensure some mid to long term maintenance of the software package.

6. WEB SCALE OPEN SOURCE SEARCH

As a last point of interaction between search and research, we will present some ideas about a Web scale open source search engine.

Citizens want to have access to the Internet content, and it is only possible with search engines. But now searching the Internet is quite a monoculture, as very few search engines are able to deliver a service at the scale of the Internet. This is not optimal as the users need different ranking algorithms and methods for different needs. Moreover the ranking used in the commercial search engines are secret recipes. So the general public would benefit from alternatives, especially some communitites. For instance specialized search services would be of particular interest for:

Alternative Languages: keyword search is not fitted to some languages due to their rich morphology (e.g., Estonian, Slovenian, Turkish) or their lack of clear word segmentation (e.g., Chinese).

Digital Libraries: services dedicated to libraries offer richer user interfaces and better document and access control than the standard search engines.

Publishing Initiatives: open publishing, open archive, open media and open access initiatives on the Internet foster varied distribution of content.

[8]though most of them are distributed for a fee.

Academic Special Interest Groups: academics have their own document genres and sometimes rich ontologies.

Blogs: several blog search engines already exist, but there are opportunities for social network studies, trend and topic detection detection, etc.

It is in and for these communities that robust development of search engines exists outside the mainstream. Analysis of these communities reveals the potential for incorporating additional capability into a search engine such as subject categories, genre, named entities, and question answering tools. In digital library applications, for instance, this kind of feedback and capability is valued [1].

On behalf of the computer scientists it is argued that *intelligent searching of the Internet* is a problem of international scope and clear need that has its origins and its solutions firmly in computer and information science. To let researchers access this grand challenge we need an open source search engine operating at a larger scale. Such a platform would not only serve as an excellent research and educational tool, it could also support a wide variety of applications and act as an important commodity to cost-conscious organisations that provide services.

But if we have seen that many development of search engines are active in the open source software community, as for now none of them is deployed at the Internet scale. This is probably more a problem of deployment of a solution than a software problem by itself. We believe that such a deployment cannot be made by duplicating commercial search engines technology. The reason is that these technologies are developed for environment where everything is under the control of a single organization. We think that this is not possible or desirable for an open source alternative where the control should be distributed among many organization, for instance at a country level.

With many organizations involved in the deployment, the full service will necessarily be a distributed one, and moreover a distribution with a loose coupling, such as with some distributions of Linux. We think that the schemes either for selection of services, merging of results, routing of queries developed in the distributed information retrieval subdomain would be a good basis for new open source developments and that this will be a good basic technology for the deployment of many open source search services on the Internet.

7. CONCLUSION

We presented some criteria for the evaluation of search engines and a panorama of open source search engines. We showed that there are many interactions between the development of software and research in the domain of information retrieval. This is due to important experimental nature of this domain. We presented the development and deployment of Internet wide search engines as a big challenge for the Computer Science community.

8. REFERENCES

[1] O. Drori. How to display search results in digital libraries - user study. In P. T. Isaías, F. Sedes, J. C. Augusto, and U. Ultes-Nitsche, editors, *NDDL/VVEIS*, pages 13–28. ICEIS Press, 2003.

[2] G. Salton. The smart document retrieval project. In *SIGIR '91, Proceedings of the fourteenth annual international ACM/SIGIR conference on Research and development in information retrieval*, pages 356–358, 1991.

[3] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[4] C. van Rijsbergen. *Information Retrieval*. Butterworth (London), 1979.

[5] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes. Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

[6] W. G. Yee, M. Beigbeder, and W. Buntine. Sigir06 workshop report: Open source information retrieval systems (osir06). *SIGIR forum*, 40(2):61–65, 2006.