

Barq: Distributed multilingual Internet search engine with focus on Arabic language*

T. Rachidi, O. Iraqi, M. Bouzoubaa, A. Ben El Khattab, M. El Kourdi, A. Zahi, and A. Bensaid
Alakhawayn University in Ifrane, Morocco
T.Rachidi@alakhawayn.ma

Abstract - Barq is a distributed multilingual search engine with focus on the Arabic language. The Barq R&D project has involved, over a period of some two years, work on Arabic language processing, Arabic word root extraction, indexing, information retrieval, automatic categorization, focused crawling, distributed computing, distributed database systems, and performance tuning. Barq indexes all documents of the web (and optionally of a particular site) including Word and XML documents that contain at least a single word of Arabic in CP1256, UTF-8, ISO8859_6, ASMO 449 or ASMO 708 code set. The documents themselves can contain other Latin-based characters. This paper focuses on describing the architecture and design patterns of Barq; as well as the various types of search that Barq supports. Issues such as Stemming/Arabic root extraction, indexing, ranking, precision and recall measurements, automatic categorization etc., are presented too, but their details are described in other works.

Keywords: Arabic computing, Search engine

1 Introduction

Up until recently, much of the work on Arabic language was oriented towards enablement. Indeed with the exception of a few research laboratories/companies, the bulk of the work related to Arabic aimed at developing solutions/applications for typesetting and displaying Arabic (e.g., ArabTeX, Windows with Arabic Support, Sakhr.com's proprietary solutions), and for reconciling between Arabic (written from right to left) and standard Latin character-based systems such as Windows, Macintosh etc. This gave rise to a variety of Arabic character codes CP1256, ISO8859_6/ASMO 708, ASMO 449, and UTF-8. Recent advances in handling Unicode [28] allow even the most unfriendly systems such as Linux to handle Arabic scripts in a very easy and native manner. Of course the Champion of this seamless integration of Arabic at the level of the operating system is Microsoft with its Windows 2000 and XP platforms.

With the problem of enablement resolved, more and more Arabic documents are being published on the Web: Aljazeera, an Arabic Television Channel, advertises more than seven million (7,000,000) documents in its web site Aljazeera.net. Furthermore, more and more Arab Internet users use Arabic-enabled email clients such as Maktoob [21] giving birth to a potentially lucrative market for software that handle Arabic documents such as search engines, document management systems, automatic document sorting systems, and corporate portals.

Today the web includes many Arabic portals (the latest of which, at the time this paper is being written, is Arabic.CNN.com) and several search engines [1,4,13,14,19,21]. Some are multilingual; others offer basic keyword based search (see Table 1), with apparent inefficiencies in precision and recall, and in response time (see Table 2). In [7] it is stated that despite the existence of seemingly complete search engines for English, the market for more sophisticated search engines is still growing, and more so for the Arabic Internet.

Table 2. Number of documents returned by leading Arabic search engines for two variant forms of the same Arabic word (Islam), as of May 9, 2003.

Search Word	Google		Al-Bahhar	
	# of docs. returned	Response time	# of docs. returned	Response time
إسلام	272	1350 ms	6246	3500 ms
إسلام	391	1400 ms	118932	5400 ms

The inefficiencies in returning pertinent documents are hindering the use of this fast-growing library that is the Arabic Internet. It is therefore of extreme importance to the development of Arabic content on the web that more efficient techniques and systems be developed. We believe that by incorporating techniques from Artificial Intelligence and Soft Computing [23], and from Arabic word computational morphology [3,17], many of the

* This work was supported financially by Alakhawayn University in Ifrane, Morocco under R&D Grant RPF1/2001 and by CoreSoft SARL.

* 0-7803-7952-7/03/\$17.00 © 2003 IEEE.

limitations could be overcome. Barq seeks to benefit from the experience of the use of English (and other languages) on the Internet for the past decade, and provide solutions comparable to today's leading solutions for English. Of course, such a solution must not only provide state of the art techniques for searching and categorizing content, but must also be provided through an industry-strength platform and a software architecture that is fault tolerant, and that can scale well to deliver low response time for over 300 million potential Arabic-speaking users.

In this paper, we will present the software architecture and design patterns of Barq: a distributed multilingual search engine with focus on Arabic, designed with maximum portability, scalability, fault-tolerance, and easy administration in mind. Focus will be put on how the various components that make Barq are put together to achieve scalability, fault-tolerance, and portability requirement [6]. Naturally content processing techniques such as indexing, precision and recall [11] performance, focused crawling [26], stemming/word root extraction [2,12,20,16,25] automatic categorization [22,8,27,24,5,30,31], concept building, etc., are equally important, and will be briefly described. The details of these components are however outside the scope of this paper.

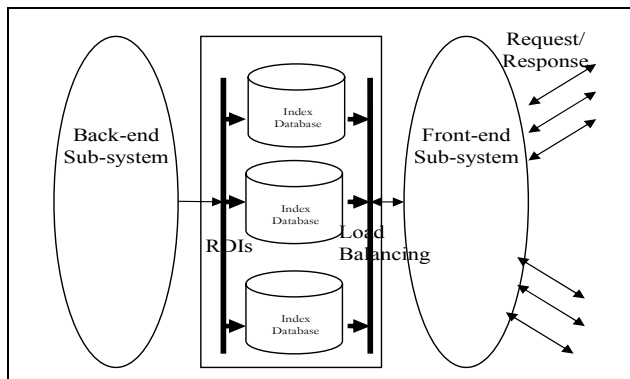


Figure 1. High-level Barq architecture

2 Barq architecture

Barq has been architected essentially around two sub-systems: a backend and front-end sub-system.

These two subsystems share a battery of replicated databases of indexes (RDIs) as shown in Figure 1. The back-end sub-system is in charge of populating the RDIs, while the front-end sub-system is responsible for answering user queries. Access to the RDIs is load-balanced to achieve minimum response time while answering user queries. The following subsections describe each sub-system in detail.

2.1 Backend sub-system

Figure 2 below shows the main components of the backend sub-system. A key feature of this sub-system is

its component-based software architecture. Each component of the back-end sub-system, namely Focused Crawler, Stemmer, Indexer, Categorizer, and Broker can be deployed independently of the other components, and in any number. Remote Method Invocation (RMI) [29] is used as the main mechanism for distribution.

2.1.1 Focused crawling

Focused crawling is used to gather and refresh documents from the web according to user/administrator-defined rules. For the purpose of building a search engine that focuses on Arabic, these rules are defined such that only documents that contain at least one word of Arabic are fetched and cached for offline processing. Intelligent techniques are used to optimize the spidering through hyperlinks in Web pages encountered during the crawling process, in order to achieve improved performances over depth-first or breadth-first spidering. The heuristics and AI techniques used here are out of the scope of this paper, but are described in [26]. The crawler component is multithreaded, and can run on any number of servers to fully use the available bandwidth. Crawler threads obtain URLs to expand from a URL server located with the back-end broker. The level of multi-threading and the location of the back-end broker are easily controllable through a GUI.

2.1.2 Document cache

Due to the dynamic nature of the Web, documents appear and disappear fairly quickly from their host Websites. The document cache keeps all fetched documents. It is used to offer end users a service for delivering cached copies of documents even if these are no longer available at their original Website. These documents are delivered directly by the Web container. Before returning any cached document, a header is appended to it. The header shows interesting metadata like the fetching date and the original URL of the document, the search keywords used for its retrieval etc. In addition, all the search keywords occurrences in the cached document are highlighted to allow the user access the needed information quickly and easily. Cached documents are refreshed regularly. The refresh rate is balanced against crawling from new documents, and is driven by the number of visits to documents.

Offline processing of crawled documents (see sections below) yields a number of indexes that are stored in a standard SQL database.

2.1.3 Stemming

Stemming is used to enrich a concept-based index for documents of the web by reducing all forms (singular, plural, adjectives etc.) of words into one canonical form often termed the root. Current implementation is based on the elimination of postfix, infix and prefix parts of words so as to retain a canonical form of the word in Arabic. The

frequency of appearance of the various “fixes” is used to handle the most probable occurrences and combinations of “fixes”. The canonical form (or word root) is used to build an index in the RDIs. Currently Barq employs a simple non-rule based algorithm that achieves over 90% correct stemming in a first experiment on arbitrary Arabic documents gathered from the web. This is a very promising result, considering that more than 30% of Arabic words are irregular. More results and discussions can be found in [10].

2.1.4 Automatic categorization

Documents that are fetched and indexed are also assigned to one or more pre-defined classes/categories of documents. The use of categories gives users the possibility to browse through categories or to build sophisticated queries and narrow down their search. Document classification in Barq is performed automatically, based on a probabilistic classifier. Cross-validation experiments have been conducted to evaluate the performance of our Arabic document categorizer. We have used a corpus of 1000 documents from a hierarchy of classes with 12 leaves (categories). The number of distinct stems in these classes is 12,902. The overall performance of our Arabic document categorizer goes up to 75.6%. This result is comparable to performance obtained on English Web document categorization. The Arabic document categorizer has also been tested on a separately collected evaluation set. On 3 of the 12 predefined categories, classification models obtained from cross-validation yielded 100% accuracy. The average classification accuracy on the 12 classes is 50%. More results and discussions can be found in [9].

2.1.5 Concepts building

Various concept sets are automatically built from parsing XML documents, from stemming, and from automatic categorization. A different set of concepts are built manually. All sets of concepts contribute to retrieving documents using the concept-based search interface. Such search returns not only documents that contain the exact words of a query or their derivatives, but also documents that contain words of the same concept as the query words. Efficient data structures are used to keep concepts in memory to speed lookups.

2.1.6 Indexing

Indexing is at the heart of Barq and for that matter any search engine. Indexing in Barq is built upon standard Relational tables/indexes. Tables contain for each document its words, roots (canonical word form), and their number of occurrences, their positions, color, font size, whether they appear in Title, Section title, body of the text, or Caption, as well as a comprehensive set of metadata about the document itself. These comprise the creation date, the refresh date, the size, and the format

(pdf, word, or HTML). Barq also builds a Hubs and Authorities structure of the Web that is used, among other things, for ranking documents.

Document words are used for keyword-based search whereas canonical word forms are used to expand user query for concept-based search. Barq indexes both documents and images. Barq indexes an image based on its context (caption and other surrounding text), which is extracted from the image document.

2.1.7 Back-end brokering

The role of the back-end Broker is first to offer a location transparent access to RDIs, and second to manage pools of connections to them. The back-end broker also serves URLs to be fetched by the Crawler threads.

2.1.8 Symmetric replication

Replication is implemented at the level of the RDBMS, without resorting to middleware, such as Tuxedo (see bea.com), to perform this task. This choice is driven by the need for keeping the cost of the solution as low as possible, and for maintaining transparency with respect to back-end sub-system components. The current deployment uses the Oracle 9i Enterprise Edition RDBMS in synchronous multi-master configuration, but other ANSI SQL-compliant RDBMSs that support Arabic/UTF-8 encoding, can be deployed in replication configuration too.

2.1.9 Back-end sub-system deployment and administration

Back-end components are stored in a central server, and are easily deployable via a web interface as needed. A unified console permits the administration of live components deployed in a production system to meet demand for scalability. For instance one can, without staling the system, add more servers for crawling, deploy more indexing components on an existing server, etc.

2.2 Front-end sub-system

RDIs and a Cache Database are used by the front-end sub-system for answering user queries.

Figure 2 also shows the architecture and components of the front-end sub-system. The session builder and User Relationship Management (URM) component records all user queries and user visited documents from the returned results. These are later used to build a context for the user, and to guide the refreshing rate of documents by the crawler, effectively building a control loop between Front-end and back-end subsystems.

2.2.1 Query dispatcher

This is a servlet run by the servlet container and is the unique entry point for all the services (search types)

offered by the Barq search engine (cf. section 3). Current implementation uses Tomcat 3.2.3 as a servlet container. User requests are intercepted by this servlet which forwards them to both a Session Builder and an Intelligence Gathering module, and a front-end Broker. The Query dispatcher also returns replies as Java Server Pages (JSPs). This implements the Model, View, Control (MVC) framework by ensuring a clear separation between control (requests interception) at the level of the servlet, model (processing and business rules) at the level of the Brokers, and view (results) at the level of JSPs.

2.2.2 Front-end brokering

The role of the front-end brokers is (1) to implement the business layer of the search technology, which consists of advanced information retrieval and document ranking techniques, (2) to offer a location-transparent access to RDIs, and (3) to manage pools of connections to the Index Databases. As many as one Broker per set {crawler, indexer, stemmer and categorizer} of backend sub-system components can be deployed to guarantee short response time when user requests are high.

2.2.3 Session building and User Relationship Management

This component is in charge of building session/user information. Session information is used to offer intelligent services to users by narrowing down search space, correlating with previous searches and with most searched keywords, etc. This component is meant to improve the relevance of improved documents, compared to traditional search engines.

2.2.4 Load balancing

Load balancing is built at two levels: at the level of DNS (Domain Name System) and at the level of the HTTP Query Dispatcher. More than one web server/web container can be deployed to service a growing number of users. User requests are, transparently to the user, routed to the web servers/web containers using a load-balancing algorithm. The Query Dispatcher at the level of each Web Server/container uses a round robin technique to forward HTTP search requests to Brokers. Each Broker accesses a fixed instance of the Replicated Database of Indexes (RDIs). More than one front-end Broker can be deployed live without stalling the system as demand grows.

2.2.5 Front-end sub-system deployment and administration

Front-end components are stored in a central server, and are easily deployable via a web interface as needed. A unified console permits the administration of live deployed front-end components to meet demands for scalability. For instance one can, without staling the system, add more web containers/servers, and deploy more brokers.

3 Query processing and document retrieval

User query undergoes a stop word deletion, as well as a query expansion process, and a query correction process. Expansion aims at augmenting the number of keywords in order to augment recall corresponding to the query [15]. Barq uses three (3) thesauri to obtain concepts for each query keyword. These are 1. a manually built thesaurus, 2. a thesaurus built from parsing XML documents, and 3. a thesaurus built from the categorization process. Furthermore, Barq also expands a query by adding roots of each keyword. Query correction computes an alternative query and proposes it to the user with the results of the expanded query. Correction takes into account both Arabic centric spelling mistakes and transliteration mistakes (English to Arabic, and Arabic to English) most frequently encountered with Arab and American natives. Barq puts no limit on the number of keywords; further, the keywords may be entered in any language. Expanded query is used to retrieve documents. Users can choose among the following types of search: Exact key word based search, concept based search, image search. The scope of the search can be set to either the whole Arabic Web or a particular site, Barq also offers similar images of a particular image search, similar document search, cached copies of indexed documents, documents referencing a particular document, and documents referenced by a particular document etc.

URLs of documents and/or images are retrieved from the indexes within the RDIs. All tables materializing indexes and their Relational indexes are pinned to memory (SGA in Oracle terminology) for fast access. The retrieval model that Barq currently implements is Boolean. All retrieved documents must contain at least one of the words/keywords, or their expansions in user query. Stop words are ignored in all cases.

3.1 Advanced queries

This service allows the user to search for documents with user-defined constraints. Currently the advanced search interface supports search with specific Internet domains, and various document metadata such as author (when available), date, document format (pdf, doc, html, xml etc.), etc.

4 Caching

Many components of Barq benefit from memory caching: at the back-end, when indexing documents, the root extraction process uses a cached hash table which proved to offer a performance gain of 1000 fold allowing for crawling and fully processing 2880 documents per day with the current platform (see section 6). At the front-end, query correction process also uses a cached hash table for the correction of query terms. Results of user query are also cached in memory for subsequent queries.

5 Ranking of results

Ranking of retrieved documents is performed using a weighted score that involves hubs and authorities [18], word attributes (such as word font size and color in documents, word position in title, body, etc.), and the relative positions of query words. The net result of this ranking mechanism is that documents that contain more words from the user query are returned always first. Moreover, the positions of words in documents influence the ranking: The documents that contain the exact (not to count stop words) user query are always returned first. Images are also ranked in the same manner, based on the context (caption and surrounding text) extracted from their original documents. As shown in Figures 3 and 4, various pieces of information are retrieved with the document URLs. These comprise document last-refresh date, document size, document type etc.

6 Results

Figure 5 shows the \$20,000.00 configuration deployed for testing and validation. The back-end subsystem runs on three (3) standard workstations (SW) with 256 Mo of RAM and 1.7GHz Pentium IV processor. Two (2) of these run each one instance of the crawling, indexing, stemming, and categorization components. The backend broker also runs on the third workstation (SW). The front-end sub-system uses five (5) servers. Two (2) servers/boxes are standard workstations (SW) with 256 Mo of RAM and 1.7GHz Pentium IV processor. The three (3) remaining servers are PIII 1GHz dual processor with 1Go of RAM servers (DPS), two of which are running Oracle 9i in multi-master replication mode, and one is running Apache/Tomcat 3.1. All servers run SMP Linux Redhat 7.2 and JDK 1.4.

With the available bandwidth (512kbps) shared among some 200 users, the fifty (50) crawler threads fetch on average 50 documents/min. Currently 100,000 Arabic Web documents have been cached and indexed.

Jmeter (from Sun) was used to send multi-keyword search requests to the front-end sub-system. Jmeter is part of the Jakarta project. Original Jmeter does not support UTF-8, so we had to build support for UTF-8 into standard Jmeter distribution for the purpose of testing. Keywords are selected randomly from a list of five hundred (500) different words with various loads (number of requests per second). Table 3 shows the results obtained with this limited infrastructure.

It is important to note that the average response times reported include network latency 10BaseT, and that these results were obtained with no tuning performed on Oracle 9i Databases. Current efforts are directed precisely towards performance tuning.

Table 3. Performance measurements obtained for the configuration described in Figure 5.

500 Random Predefined requests	
# of requests/second	Average response Time (ms)
1 Keyword search	
1	760
5	3170
10	6302
20	12750
50	32750
6 Keyword search	
1	1384
5	6429
10	12125
20	24760
50	60703

These preliminary results show that the response time doubles when the number of terms in query increases six-fold. Other results (not show here for space limitations) show that average response time is multiplied by two (2) when the number of RDIs is set to one (1). Further experimental results are underway to fully demonstrate the scalability of the system.

7 Conclusions

We have described the main features of Barq: a distributed multilingual search engine with focus on Arabic, designed with maximum portability, scalability, fault-tolerance, and easy administration. Barq is leveraging of results of research in Arabic content processing, and supports various search types. The results obtained on a modest platform are promising. It is expected that when deployed on real servers, Barq will answer queries with high precision in a sub-second response time. Current works focus on distributing brokering for the back-end, the use of machine translation techniques to improve recall of multilingual documents, and tuning for sub second response time.

Table 1. Leading Arabic search engines as of June 2002

Search engine	Technology/Platform	Scope	Multilingual support	Stemming	Automatic Categorization	Multi-keyword search
Hahoua.com	Unknown	Internet	Yes	No	No	No
Google.com	CGI/ Linux	Internet	Yes	No	No	Yes
Ayna.com	CGI /Perl/ Linux	Local, Internet, Digital Library	Yes	No	No	No
AlBahhar.com	CGI/	Discussion Groups, News	Yes	Yes	No	No
Al-Idriissi	Proprietary	Local	No	Yes	No	Yes
Konouz	Unknown	Local	No	No	No	No

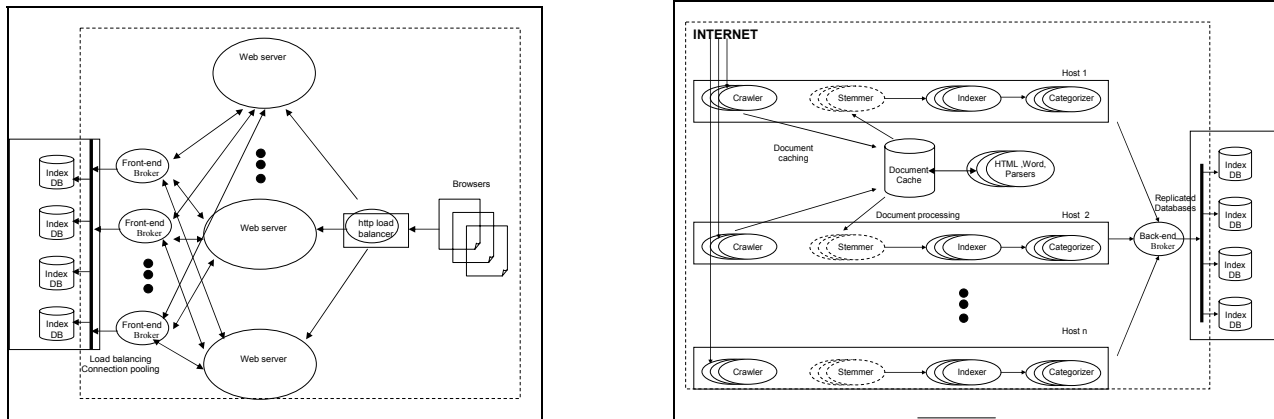


Figure 2. Architecture and components of Barq's front-end and back-end subsystems

Figure 3. Snapshot of retrieved information

References

- [1] Al-Bahar, <http://www.albahhar.com/>
- [2] R. S. Al-Fedaghi, and Al-Anzi, "A new algorithm to generate Arabic root pattern forms", In Proc of the 11th national Computer Conference and Exhibition, pp. 391-400, 1989.
- [3] R. Al-Shalabi and M. Evens, "A computational morphology system for Arabic", In Workshop on Computational Approaches to Semitic Languages, COLING-ACL98, 1998.
- [4] Ayna, <http://www.ayna.com>
- [5] M. Benkhalifa, M. Bensaid, A. Mouradi, "Text categorization using semi-supervised fuzzy c-means algorithm" Proc of the 18th Int. Conf of the North American Fuzzy Information, pp. 561-565, 1999.
- [6] S. Brin and L. Page, "The anatomy of a Large-Scale Hypertextual Web Search Engine", *WWW7/Computer Networks* 30(1-7): pp. 107-117, 1998.
- [7] H. Collier, S. E. Arnold, "Search Engines: Evolution and Diffusion," version 1.2, January 31, 2003, sa@arnoldit.com.
- [8] S. Dumais, J. Platt, D. Heckerman, M. Sahami, "Inductive Learning Algorithms and Representations for Text Categorization." Proc of the 7th Int Conf on Information and Knowledge Management, 1998.
- [9] M. El-Kourdi, A. Bensaid, and T. Rachidi, "Naïve bayes Automatic Categorization of Arabic Web documents", to appear, A.Bensaid@alakhawayn.ma
- [10] M. El-Kourdi, T. Rachidi, A. Bensaid, and A. Chekayri, "A concatenative approach to Arabic root extraction", to appear, T.rachidi@alakhawayn.ma
- [11] W. B. Frakes, Ricardo Baeza-Yates, *Information Retrieval Data Structures & Algorithms*, Prentice Hall, ISBN 0-13-463837-9, 1992.
- [12] M. Fuller, and J. Zobel, "Conflation-based Comparison of Stemming Algorithms", In Proc of 3rd Australian Document Computing Symposium, Sydney, pp. 8-13, 1998.
- [13] Google, <http://www.google.com/>
- [14] Hahooa, <http://www.hahooa.com/nav.php?ver=ar>
- [15] W. Hersh, S. Price, and L. Donohoe, "Assessing Thesaurus-Based Query Expansion Using the UMLS Metathesaurus", In Proceedings of the 2000 Annual AMIA Fall Symposium, pp. 344-348, 2000.
- [16] S. Khoja, and R. Garside, "Stemming Arabic Text", Lancaster University, UK. <http://www.comp.lancs.uk/computing/users/khoja/stemmer.ps>, 1999.
- [17] G. A. Kiraz, "Arabic Computational Morphology in the West", In Proc. of the 6th In.l Conf. and Exhibition on Multi-lingual Computing, Cambridge, 1998.
- [18] J. M. Kleinberg, "Hubs, authorities, and communities", *ACM Computing Surveys (CSUR)*, Vol. 31 No.4, Dec. 1999.
- [19] Konouz, <http://www.konouz.com>
- [20] J. B. Lovins, "Development of Stemming algorithm", *Mechanical Translation and Computational Linguistics*, Vol. 11, No 1-2, pp 22-31, 1968.
- [21] Maktoob, <http://www.maktoob.com>
- [22] A. McCallum, K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification", AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48. Technical Report WS-98-05. AAAI Press, 1998.
- [23] T. M. Mitchell, *Machine Learning*, McGraw Hill, ISBN 0070428077, 1997.
- [24] K. Nigam, A. McCallum, S. Thrun and T. Mitchell, "Text Classification from Labeled and Unlabeled Documents using EM." *Machine Learning*, Vol. 39, No. (2/3), pp. 103-134, 2000.
- [25] M. F. Porter, "An algorithm for suffix stripping," *Program*, Vol. 14, No. 3, pp. 130-137, 1980.
- [26] T. Rachidi, O. Iraqi, M. Bouzoubaa, and A. Bensaid,, "Focused crawling of the Arabic Web", to appear, T.Rachidi@alakhawayn.ma
- [27] H. Ragas and C. H. Koster, "Four text classification algorithms compared on a Dutch corpus", Proc of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval, Melbourne, pp. 369-370, 1998.
- [28] UNICODE, <http://www.unicode.org/>
- [29] M. Wutka, *Special Edition Using Java 2 Enterprise Edition*, Que, 1st edition, May 8, 2001.
- [30] M. Yahyaoui, "Towards Arabic Web Page Classifier", MS Thesis, Alakhawayn University in Ifrane, 2001.
- [31] Yiming Yang, Jan O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization", Proc 14th Int Conf on Machine Learning, pp.412-420, 1997.