Available online at www.sciencedirect.com
ScienceDirect
Decision Support Systems 45 (2008) 621–640
Decision Support Systems
www.elsevier.com/locate/dss

# SpidersRUs: Creating specialized search engines in multiple languages

Michael Chau [a,*], Jialun Qin [b], Yilu Zhou [c], Chunju Tseng [d], Hsinchun Chen [e]

[a] School of Business, The University of Hong Kong, Pokfulam, Hong Kong
[b] Department of Management, University of Massachusetts Lowell, Lowell, MA 01854, USA
[c] Information Systems and Technology Management, George Washington University, Washington, DC 20052, USA
[d] Department of Management Information Systems, The University of Arizona, Tucson, AZ 85721, USA
[e] Department of Management Information Systems, The University of Arizona, Tucson, AZ 85721, USA

Available online 27 July 2007

**Abstract**

While small-scale search engines in specific domains and languages are increasingly used by Web users, most existing search engine development tools do not support the development of search engines in languages other than English, cannot be integrated with other applications, or rely on proprietary software. A tool that supports search engine creation in multiple languages is thus highly desired. To study the research issues involved, we review related literature and suggest the criteria for an ideal search tool. We present the design of a toolkit, called SpidersRUs, developed for multilingual search engine creation. The design and implementation of the tool, consisting of a Spider module, an Indexer module, an Index Structure, a Search module, and a Graphical User Interface module, are discussed in detail. A sample user session and a case study on using the tool to develop a medical search engine in Chinese are also presented. The technical issues involved and the lessons learned in the project are then discussed. This study demonstrates that the proposed architecture is feasible in developing search engines easily in different languages such as Chinese, Spanish, Japanese, and Arabic.


*Keywords:* Search engine development; Multilingual search engines; Information retrieval

## 1. Introduction

With the growing popularity of the Web, search engines have been widely used in recent years. Most Web users begin their Web activities by submitting a query to a search engine such as Google or Yahoo. However, as the size of the Web is growing exponentially and the number of indexable pages on the Web has exceeded four billion, it has become more and more difficult for search engines to keep an up-to-date and comprehensive search index, resulting in low precision and low recall rates. Users often find it difficult to search for useful and high-quality information on the Web using general-purpose search engines, especially when searching for information on a specific topic or in a language other than English.

Many domain-specific or language-specific search engines have been built to facilitate more efficient searching in different areas. These search engines alleviate the information overload problem to some extent by providing more precise results and more customized features.

* Corresponding author.
*E-mail addresses:* mchau@business.hku.hk (M. Chau), jialun_qin@uml.edu (J. Qin), yzhou@gwu.edu (Y. Zhou), chunju@u.arizona.edu (C. Tseng), hchen@eller.arizona.edu (H. Chen).

0167-9236/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2007.07.006

Alternatively, some Web users prefer to address the problem by creating their own small-scale search engines in a specific domain or language for some special collections. This allows them to have their own customized, searchable digital libraries that can be accessed by other users.

However, much effort will be needed to construct such a search engine that provides effective and efficient search functionalities with a high-quality collection of documents. Manually creating the collection and indexing the terms for searching will require a lot of time and effort. Another alternative is to automate these task using software tools [35]. Although comprehensive software tools for creating search engines exist, most of them cannot work on non-English languages such as various European, Asian, and Middle East languages. In recent years, the number of non-English resources on the Web is growing rapidly and it has been estimated that more than 60% of Web users' native language is not English. A tool that can build specialized search engines in different languages is thus highly desired.

In this paper, we present our work in designing and implementing a software tool that addresses this problem. We will focus on the architectural design and the technical issues involved in developing such a tool. The rest of the paper is organized as follows. Section 2 reviews related work in search engine development. Section 3 discusses our research objective and suggests the criteria for an ideal search engine development tool. In Section 4, we present our proposed system, called "SpidersRUs," that we have developed to help users create specialized search engines in different domains and languages. Section 5 presents sample sessions demonstrating how users can use the tool in building specialized search engines. In Section 6, we present a case study describing how the proposed tool has been used to develop a medical search engine in Chinese. In Section 7, we discuss some of the technical and design issues involved in developing a multilingual search engine development tool. Finally, Section 8 concludes our work and provides some suggestions for future research.

## 2. Research background

Search engines and search techniques have been widely studied in information retrieval literature and have been used a lot in traditional information systems. These search engines allow users to retrieve information efficiently and effectively from a collection. As discussed earlier, due to the large size of the Web, it is often desirable to build domain-specific or language-specific collections that can be searched and managed more easily. Several components are involved in building a specialized collection and the corresponding search engines, namely (1) collection building, (2) indexing, and (3) searching. In this section, we review existing techniques in these three areas, as well as some existing tools that can be used for search engine development.

### 2.1. Collection building

Perhaps the most important part of any search engine is the content. No matter what indexing and retrieval techniques are used, a search engine would not be useful if the users do not like its content. Traditionally, information retrieval systems have been developed for materials that are available offline. In recent years, due to the rapid growth of the Internet and related technologies, increasingly more useful and valuable resources are available on the Web. These resources include corporate Web sites, personal homepages, academic research papers, online community resources, among others.

As Web pages link to each other by hyperlinks, "spiders" program are often used to traverse the Web to collect Web pages. Spiders, also known as Web robots, Web agents, crawlers, worms, or wanderers, are programs behind a search engine that retrieve Web pages by recursively following hyperlinks (URLs) in pages using standard HTTP protocols [3,5,11]. First, the spiders read from a list of starting seed URLs and download the documents at these URLs. Each downloaded page is processed and the URLs contained within are extracted and added to the queue. Each spider then selects the next URL from the queue and continues the process until the required number of documents have been downloaded or the local computing resources have been exhausted. This Web page collection process is often called "spidering" or "crawling". To improve speed, spiders usually connect simultaneously to multiple Web servers in order to download documents in parallel, either using multiple threads of execution [17] or asynchronous input/output [2].

In addition, a well-designed, "polite" spider should avoid sending multiple requests to a Web server within a short amount of time, which would otherwise overload the Web server [15]. Webmasters or Web page authors also should be able to specify whether they want to exclude particular spiders' access. There are two standard ways. The first one, called the Robot Exclusion Protocol, allows Web site administrators to indicate, by specifying a file named robots.txt in the Web site's root directory, which parts of their site should not be visited by a spider (robot) [21]. In the second method, usually

known as the Robots META Tag, Web page authors can indicate to spiders whether a document may be indexed or used to extract more links [23].

Spidering tools have been available since the early days of the Web. tueMosaic is an early example of personal Web spiders [13]. Using tueMosaic, users can enter keywords, specify the depth and width of search for links contained in the current homepages displayed, and request the spider to fetch homepages connected to the current homepage. WebRipper, WebMiner, and Teleport are some software tools that allow users to download massively from given Web sites files with particular types or attributes. Some open-source tools also have become available in recent years. For example, Heritrix, the crawler for the Internet Archive project, has been made available for download [19].

Some spiders were also designed to provide additional functionalities. For example, the Competitive Intelligence Spider performs breadth-first search on given URLs and applies linguistic analysis and clustering to the search results [7]. The hybrid simulated annealing spider supports "global" searching on the Web [38].

### 2.2. Indexing

Documents retrieved by spiders are stored in a repository of Web pages. To reduce the storage space needed, the pages are often compressed when they are stored. The repository is usually in the form of a database, but it is also common for small-scale search systems to simply store the documents as files. An indexer processes the documents in the repository and builds an underlying index of the search engine. The indexer tokenizes each document into words and records the occurrences of each word in the document. The indexer also calculates scores, such as the term frequency and document frequency of each word, that can be used for search result ranking or further processing.

The indexing results from the Indexer are then converted into an "inverted index." While the result of the original indexing process, often called the forward index, maps a document to a list of words contained within it, the inverted index maps a word to a list of documents containing the word. This allows fast retrieval of documents when a search query is submitted to the search engine. The resulting searchable indexes are usually stored into a database or a file system.

SMART (System for the Mechanical Analysis and Retrieval of Text), developed by Salton and his colleagues, probably was one of the earliest indexing tools that were widely used [29]. SMART creates an index for a set of documents and assigns a weight to each term–document relationship, based on the TFIDF formula (term frequency multiplied by inverse document frequency) [28]. *Lucene*, *Swish* and *Glimpse* [22] are three other tools that create indexes over a set of documents, especially Web documents. These tools scan through the documents and build an inverted, searchable index between each term and each document based on their occurrences. All three tools were later enhanced or combined with other tools to include Web page fetching (i.e., spidering) capabilities. For example, Lucene had a sub-project for spider called Nutch [24], Swish was enhanced to become *Swish-e* [32], *WebGlimpse* was a spider-enabled version of Glimpse.

### 2.3. Searching

A query engine accepts search queries from users and performs searches on the indexes [1]. After retrieving search results from the indexes, the query engine is also responsible for ranking the search results according to various content analysis and link analysis scores. It is also responsible for generating a summary for each search result, often based on the Web page repository. The query engine in some search engines is also responsible for caching the results of popular search queries. After all the processing, the query engine generates and renders a search result HTML page and sends it back to users through the user interface. The user interface allows users to submit their search queries and view the search results. When a user performs a search through the Web interface, the query is passed to the query engine, which retrieves the search results from the index database and passes them back to the user.

### 2.4. Existing tools

In addition to the spidering and indexing tools discussed earlier, there are many free software tools that provide all of the components of a search engine, i.e., collection building, indexing, searching, index storage structure, and user interface. Users can build their own search engines with such tools. Some popular examples of comprehensive search engine development tools are WebGlimpse [22], ht://dig [18], GreenStone [35–37] and Alkaline [34]. These tools take a list of Web sites from a user as seed URLs, collecting Web pages based on these seeds, index these pages, and set up a user interface for querying and browsing.

Although these toolkits provide integrated environments for users to build their own domain-specific search engines, most of these tools only work for

English documents and are not able to process non-English documents, especially for non-alphabetical languages. Only a few of them, such as GreenStone, support multilingual collection building. As a result, most of these tools cannot be used to create digital library for non-English collections.

Another problem is that many of these tools do not provide enough technical details, and their components and building steps are tightly coupled. As a result, users often find it difficult to customize the tools or reuse the intermediate results in other applications (such as document classification) even if they have strong technical skills. For example, in Alkaline, all the intermediate results of the spidering and indexing processes are hidden from users. In addition, most of these tools store the spidering results (Web pages) and indexing results (indexed terms and the document–term relationships) in binary format or other proprietary formats, often due to performance issues. This has made it very difficult, if not impossible, for users to use the results from a search tool for other purposes or in other applications.

## 3. Research objective

As discussed earlier, a tool that supports search engine creation in multiple languages is highly desired. An ideal tool should have the following properties:

- Platform-independent: The tool should be platform-independent such that users with different platforms and operating systems can use it without problems.
- Self-contained: Similar to the platform-independent requirement, the tool should be self-contained and should not rely on any underlying database system. This can ensure that users can use the tool easily without having to buy or install any database on their computers.
- Modular: The tool should be designed as a modular and object-oriented tool such that users can modify or plug-in other modules relatively easily.
- User-friendly: Users should be able to install the tool, customize the settings, build collections, and maintain the system easily.
- Integratable: The tool should be designed in a way such that it would be easy to integrate it with other existing tools and systems. Output of the tool (such as collected Web pages and index files) should be in plain text format such that they can be reused in other applications easily.
- Multilingual support: The tool should be able to process documents in different languages.
- Multiple format support: The tool should be able to process documents in popular formats in addition to HTML, e.g., MS Word and PDF files.

Most existing tools, however, do not satisfy these requirements because of various problems such as the inability to process non-English languages, the complexity involved in the tools, or the low interoperability and portability of the intermediate and end results. To address these problems, a tool, called SpidersRUs, has been developed to support specialized search engine building in different languages. This research aims to explore the various issues in designing and implementing such a software tool. In particular, we investigate the technical issues involved in a tool that supports collection building in various languages.

## 4. Proposed architecture

### 4.1. Choice of implementation platform

To explore the research issues, we designed and developed a software tool called SpidersRUs for creating specialized search engines in different languages (http://ai.bpa.arizona.edu/spidersrus/). After considering the requirements discussed in the previous section, the Java language was chosen as the programming language for our implementation. Java is an object-oriented language that is platform-independent. This allows the tool to run on different hardware platforms and operating systems with the Java virtual machine installed. Java also has been shown to be suitable for search engine development and has been used in several other spider applications [10,17]. In addition, Java has been well designed to support multiple languages. Each character is stored as a double-byte character in Unicode rather than a single-byte one. The Unicode standard is a character coding system that provides a unique double-byte number for every character in all languages [33]. Such standard is extremely useful for developing multilingual systems [12].

We also decided to store all intermediate files, including Web pages and index files, in their original formats or plain text formats. Although this may not be very space-efficient as the files are not compressed, it allows users to use the intermediate results for other purposes. More details will be discussed later in this section.

### 4.2. System architecture

The tool consists of five major modules — Spider, Indexer, Index Structure, Searcher, and the Graphical

User Interface (GUI). Fig. 1 shows an architecture diagram depicting how the components work together. Each component will be explained in detail in this section.

The architecture of SpidersRUs is similar to those of popular Web search engines [1,2]. The Spider module collects documents from the Web. These documents are stored as files in their original form in the local repository. The files are then indexed by the Indexer module. Terms are extracted from the documents and the indexes are created and saved as files. The users can then start the Searcher module which will load the index files. This will allow the users to search the collection through the GUI. In the following, we will discuss the design of each component and the technical issues involved. In particular, we will discuss how they have been implemented to work with multiple languages.

*4.2.1. Spider*

The task of the Spider module is to fetch Web pages and other Web documents from the Internet to form the collection that the users want to create. Designing a Spider for a search engine is a challenging task. First, a search engine spider needs to be scalable and be able to download millions of Web pages within a reasonable amount of time and limited memory usage. Second, the spider needs to avoid duplicated URLs as well as duplicated content. Many Websites have mirror sites containing exactly the same content and different URLs often refer to the same document. For instance, the four URLs http://www.arizona.edu, http://arizona.edu, http://128.196.133.81, and http://www.arizona.edu/index.html all point to the same Web page. An efficient spider should only fetch and store the content of one of these pages to avoid having duplicated documents in the search results list. Third, a search engine spider should not cause problems to other Web users. As reviewed in Section 2, a polite spider should provide Robot Exclusion support and avoid overloading any Web servers. Lastly, because our system is designed for specialized search engine development, the spider also should allow users to specify customizable filters on what URLs and what contents they would like to collect. Unwanted content should be filtered by the spiders. Our design of the Spider module has partly followed the design of Mercator [17] and is shown in Fig. 2.

The core of the Spider module is the small spiders – the search agents that actually connect to Web servers and download documents. These spiders are written in multiple threads such that they connect to multiple Web servers simultaneously. The multiple threads ensure that the bandwidth between the computer and the Internet can be utilized effectively as the download speed will be least affected by the slow response time of one particular Web server. Each spider will get the first URL from its corresponding URL queue (to be discussed below) and try to download the page.

The collected files are sent to a SpiderMaster object that controls all the spiders. These documents are then passed to the ContentHandler. The ContentHandler is responsible for checking the documents downloaded and saving them to the local disk. The ContentHandler will first check to see whether the document has been downloaded before, possibly from a mirror Web site, using a hashtable called Content Seen. After checking with the hashtable, the document will be checked by the

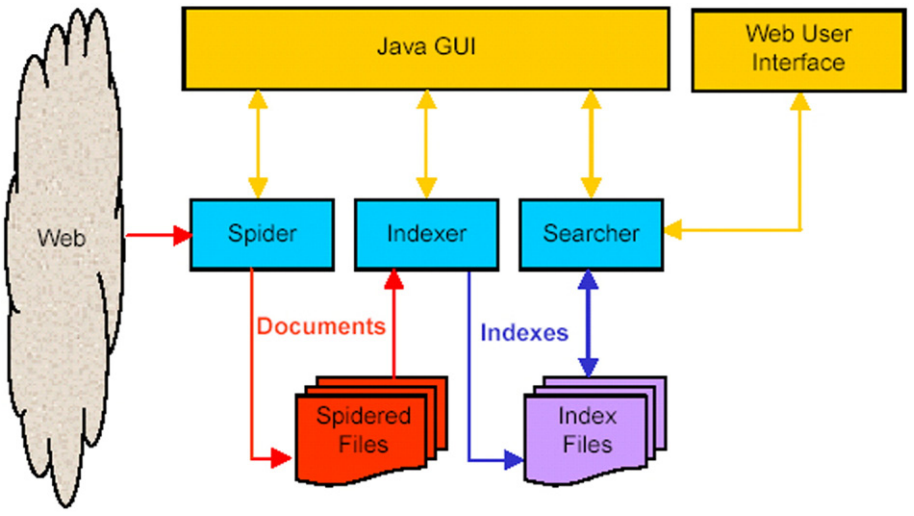


Fig. 1. System architecture.

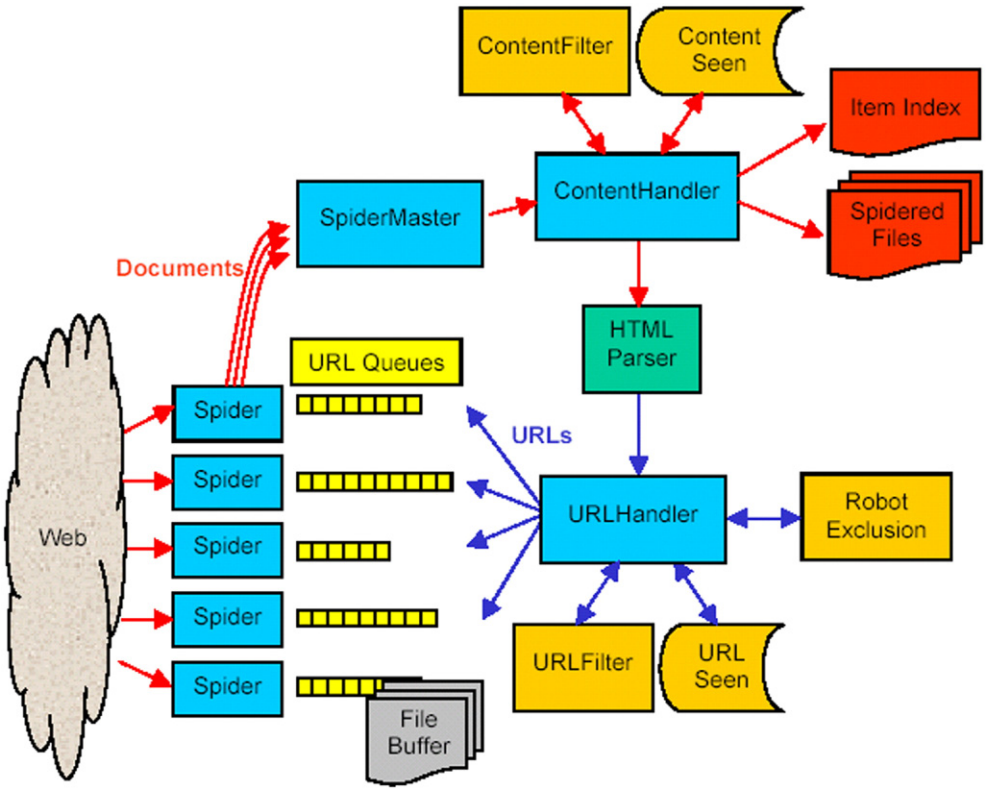


Fig. 2. Design of the Spider module.

ContentFilter. The filter contains two user-specified lists: a good term list and a bad term list. This process allows users to control the quality and relevance of the collection. After that, the document will be saved to the local disk. A unique item id also will be assigned to the document and an index will be stored in the Item Index.

If the document is an HTML file, the HTMLParser will extract all the URLs from the file. These URLs will be passed to the URLHandler for processing. First, they will be checked against the Robot Exclusion Protocol. If a robots.txt file is found on that Web server specifying that the current URL should be excluded, then the URL will be discarded. If the URL passes the checking, it will then be checked against the URL Filter. The URL Filter also contains two user-specified lists: a Good URL list and a Bad URL list. These two lists can be specified using regular expression [20]. After checking with the URL Filter, the URL will be checked to see whether it has been seen before using the URL Seen hashtable.

If a URL passes all the checking, it will be added to the URL queues. While our spiders are designed to download Web documents in a breadth-first search order, our spiders also need to be "polite" and do not all send requests to the same Web server at the same time. Following the design of Mercator, a set of multiple first-in–first-out queues is used in our system instead of a single queue. Each queue is assigned to exactly one spider, and URLs from the same host are always added to the same queue. This makes sure that there will not be more than one spider sending request to the same Web server simultaneously, thus preventing our system from overloading any Web servers.

#### 4.2.2. Indexer

After the spiders have finished downloading the required number of documents or have been stopped manually, the user can proceed to the next step to start the Indexer module. The Indexer creates a searchable index for the documents collected by extracting terms from the documents and recording the relationships between these terms and the documents.

As the main objective of our research is to develop a tool to support multilingual search engines creation, our Indexer should be able to process documents in different languages. Also, because of the diverse formats of documents on the Web, the Indexer also should be able to handle documents in different formats, such as HTML, MS Word, PDF, and so on. The design of our

Indexer module and our Index Structure is shown in Fig. 3.

First, the IndexerMaster checks the Item Index created by the Spider module to see how many documents have been collected and their locations in the local disk. The information for each document is then sent to the appropriate parser to convert the file to a uniform document object for indexing. The document object models a document in different elements such as document title and document body in plain text formats. The file is loaded by the Indexer using the encoding specified by the user. Because Java supports the reading of byte streams based on different character sets, this allows our program to read the file correctly using the specified language encoding.

Each document object is then passed to the DocIndexer for indexing. Specifically, the DocIndexer will tokenize the document into words and then create an index recording that these words appear in this document. The relative positional information and the frequency for each word are also recorded. This process is relatively easy for most Western languages in which spaces are used to separate words. However, for ideographic languages such as Chinese or Japanese, in which there is no space between words, it would not make sense to use only spaces and punctuations in tokenization because this would result in very long phrases which would be difficult to be matched by users' search queries. To address the problem, the DocIndexer first tries to detect what language is used in the document. If a non-space-delimited language is used, the DocIndexer will index each double-byte character found in the document. Otherwise, it will simply index each word. One should note that single-character-based indexing for Asian languages may not achieve retrieval precision as good as that of other advanced indexing methods such as multiple-character-based indexing (e.g., bigrams or n-grams) or phrase-based indexing. However, a single-character index is often smaller in size and can achieve higher recall. To maintain simplicity and smaller index size, we have adopted single-character-based indexing in our tool. Nonetheless, the DocIndexer can be easily extended to adopt other indexing methods such as multiple-character-based indexing or stemming (e.g., [26]), which is currently not available in our system.

After the index is created, it is stored in the Index Structure and converted into an "inverted index". While the original index maps a document to a list of words contained within, an inverted index is designed to map a word to a list of documents containing the word. This allows fast retrieval of documents when a search query is passed to the search engine. The inverted index is also stored in the Index Structure.

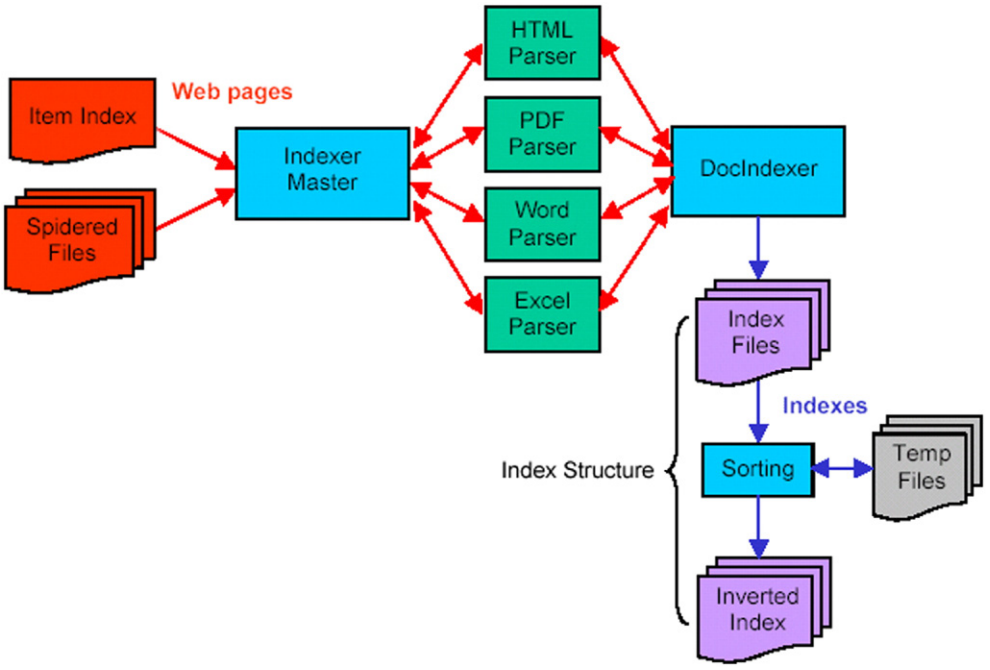


Fig. 3. Design of the Indexer module and the Index Structure.

#### 4.2.3. Index Structure

The Index Structure is the underlying representation of the documents and the search index in any search engine created by our tool. One of the major requirements of a search engine is to support fast retrieval. Users will lose patience on search engines that have slow response time. Therefore, a well-designed index structure and retrieval algorithm is core to a search engine. As we do not make use of any proprietary database in our tool, we have developed our own file structure and retrieval mechanisms for our indexes. This file-based index allows our tool to run independently without relying on any database systems.

Because we would like the index to be easily read by users or reused in other applications, we have aimed to design the Index Structure in a simple format. Currently, the Index Structure consists only of five files, and these files are stored in plain text format which is easily readable by users. Although a binary file format will allow faster retrieval and more efficient use of storage space, such format is difficult to be read by humans or reused in other applications. While it is also possible to store the files in binary formats and provide simple API for users to access these files, in this case, however, the files are still not immediately reusable, especially by users with little computer background. Since the performance difference in retrieval time and storage space is not large, we decided to use the plain text format. As a result, our index is in a more readily reusable format than other similar applications which store their indexes in binary forms.

Our Index Structure consists of five files, namely Items.dat, ItemDetails.dat, Terms.dat, Relations.dat, and SortedRelations.dat. These files store the item (document) index, the item details (e.g., document titles), the term index, the document–term index, and the inverted index, respectively. The files are designed based on the table design in a relational database.

In order to support different languages, the files, particularly the terms and the document titles, have to be stored in their respective encoding. One alternative is to store such information using UTF-8, a compressed storage format for Unicode characters. However, because UTF-8 is still not very popular for Web pages in some languages, we decided to store this information in their original encoding for easier processing and display to the users.

#### 4.2.4. Searcher

After the index is created for a collection, the Searcher module will allow users to perform searches. The module is a query engine responsible for parsing user queries and retrieving results from the index. The Search module supports both Boolean searching (specified using keywords “AND”, “OR” and “NOT”) and phrase searching (specified using double quotes). The phrase searching is possible because we stored the position information in our index. Search results are then ranked by the frequencies of the search terms and returned to the user.

When the Search module is started, it loads some information about the Index Structure in order to support retrieval. The Search module contains a simple version of a Web server, which will create listen on a port for user query. For example, after starting the Search module using the default port number 9999, the search engine developer can access the search service through the URL http://localhost:9999/. For other users, the search service can be accessed by replacing “localhost” with the domain name or IP address, e.g.,http://ai.arizona.edu:9999/. When a search query is inputted by the user, it will be sent to the Search module and processed according to the given encoding. The Search module will then retrieve the search results which will be rendered in an HTML template and returned to the user through the given language encoding. The HTML template and the image files used in the search result page can be customized by users in different languages.

#### 4.2.5. GUI

The Graphical User Interface (GUI) module is writõten in Java and allows users to interact with the other modules in the system. Users can create new projects, maintain their collections, or delete existing ones through the GUI. They can also specify the options for the other modules (e.g., the number of pages to be collected by the Spider) through the interface. In the next section, we will show how users can use the tool through the user interface.

## 5. Sample user sessions

The SpidersRUs toolkit allows users to build, access, and maintain multiple document collections in multiple languages. In this section, we demonstrate the process of building multilingual document collections and Web-based search engine services with the toolkit.

### 5.1. Creating a new collection

The first step of building a new document collection is to create a new project in the toolkit. There are two different methods to create a new project: a simple method and an advanced method. A user can choose the

desired method by clicking the "File" menu in the toolbar and selecting either "New" or "Advanced New" option in a drop-down menu.

After selecting the method of creating a new project, the user can specify the name of the project, the location to store the project files, the language encoding of the collection, and the description of the project (see Fig. 4). In the advanced method, the user can modify and implement certain components in the toolkit such as HTMLParser, HTMLFilter and Searcher. This allows users to customize the toolkit for specific usage.

After all the parameters are specified, a new project is created. The current status of the new project will be displayed in the main window. Other existing projects that were created in the past also will be shown on the panel on the left-hand side.

### 5.2. Spidering

After a project is created, the user can interact with the Spider module by clicking on the "Spidering" tab (see Fig. 5). The "Add Seeds" button allows the user to add seed URLs (starting URLs) to start the spidering process. The user can add seeds by typing the URLs one by one in the text box or by loading the URLs from a text file which contains a list of seed URLs. The seed URLs added will be displayed in the interface.

The "Advanced" button allows the user to set parameters of the Spider module such as the number of spiders (multiple threads) to be used, the maximum level from the seed URLs that the spiders should visit, the number of Web pages to be collected, the maximum waiting time that a spider should wait before giving up on fetching a Web page, and whether any proxy server should be used. The user can also limit the spiders to fetch pages only in the same Web domain, or specify whether the Robot Exclusion Protocol should be followed during the spidering process.

After all the parameters are specified, the user can start the spidering process by clicking the "Start" button. During the spidering process, status messages, including errors encountered, will be displayed in a message window. This allows the user to monitor the spider process closely. The user also can stop the process by pushing the "Stop" button. Clicking on the "Resume" button will resume a paused spidering process.

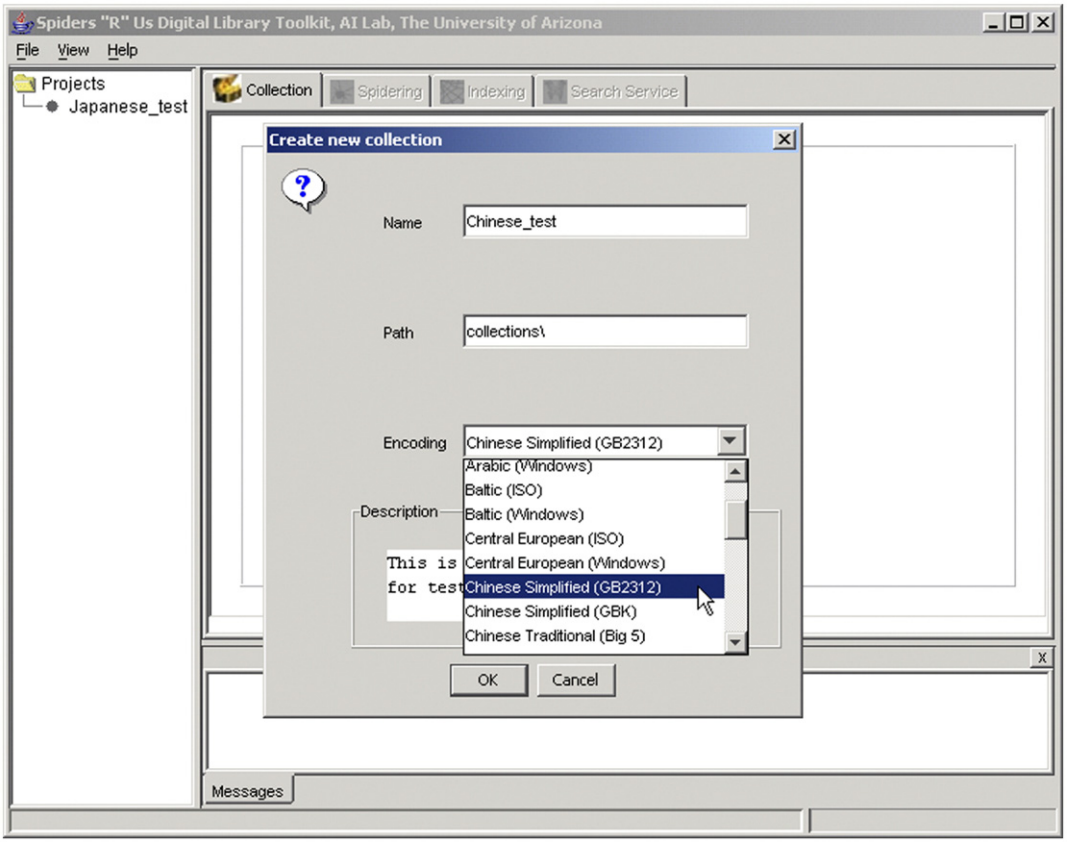


Fig. 4. Creating a new project.

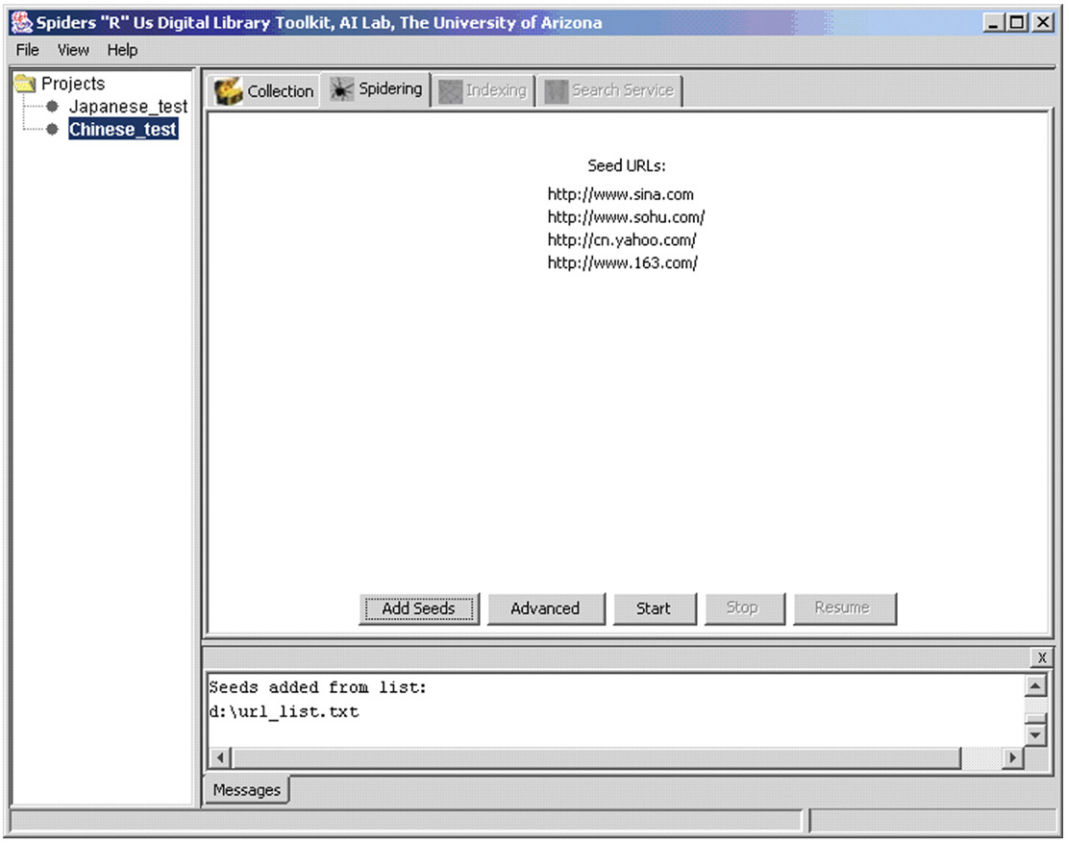


Fig. 5. The user interface for the Spider module.

### 5.3. Indexing

After the spidering process is completed, the user can go to the interface for the Index module by clicking on the "Indexing" tab (see Fig. 6). The user can start the indexing process by pushing the "Start Indexing" button. During the indexing process, status messages and error messages are displayed in the message window. The default indexer in the toolkit will automatically select word-based index or character-based indexing method according to the language of the document collection specified. When the forward indexing process is finished, the user can then start the sorting process to create the inverted index by clicking the "Starting Sorting" button. The indexing process and sorting process are separated in order to make the program more flexible.

### 5.4. Search service

After the indexing and sorting processes have been finished, the user can go to the search service interface. The user can assign a port number for the search service or simply accept the default port number "9999" and click the "Start service" button. When the service starts, the user can launch a browser window (e.g., Microsoft Internet Explorer) to search the collection by clicking the "Launch Browser" button. Any search results will be displayed to the user in the proper language encoding (see Fig. 7). In the example, the user enters the search term "贸易" ("trade") in Simplified Chinese in the search box in the Web browser and the results are displayed to the user. As mentioned earlier, the HTML template and the image files used can be customized by the users.

### 5.5. Maintaining previous collections

The SpidersRUs toolkit can build and maintain multiple projects or collections. To work on a previously built collection, the user can simply choose the project name on the left-hand side window and click the "Load Project" button (see Fig. 8). After the project is loaded, all the information related to the project will be retrieved and displayed. The user can make modifications to the collection or simply start the search service for this collection. The user also can delete an existing collection by clicking the "Delete" button.

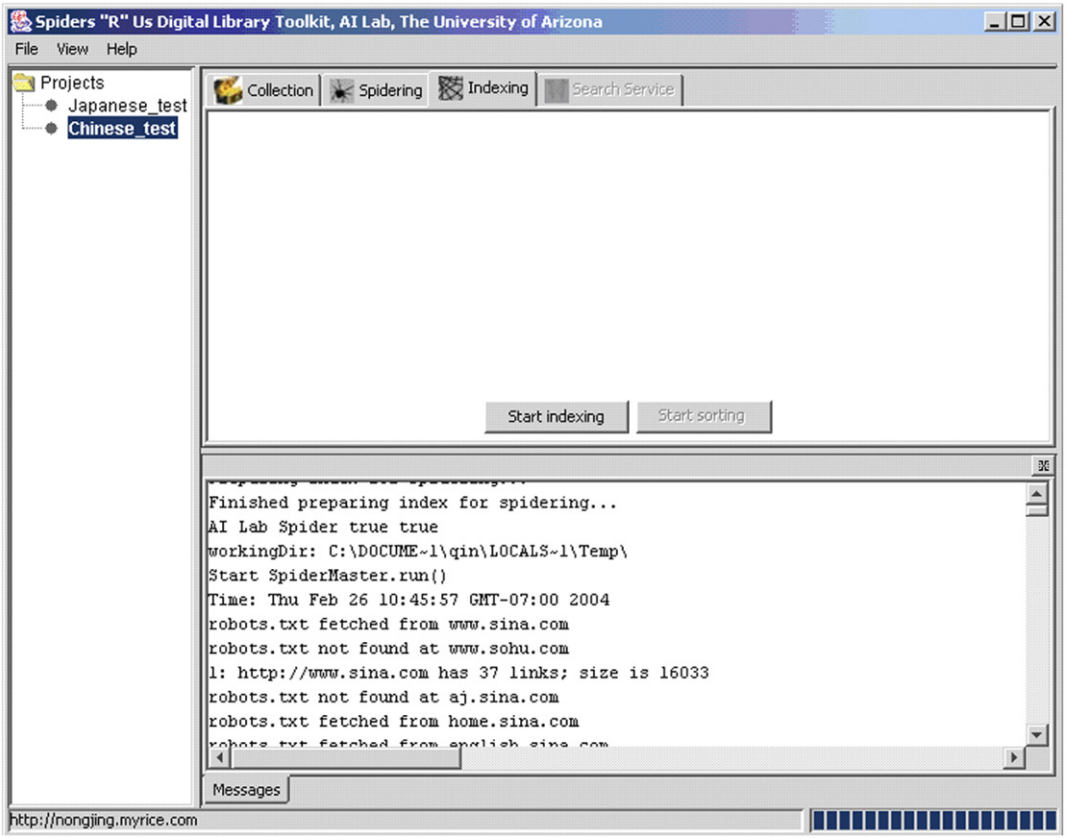


Fig. 6. The user interface for the Indexer module.

### 5.6. Examples of collections in other languages

The SpidersRUs Toolkit is designed to deal with different languages. Fig. 9 shows some examples of collections built by the toolkit in Japanese, Spanish, and Arabic. As discussed earlier, the support for multiple languages, which is not available in most current search engine tools, will be very useful for specialized search engine development. Currently, the Web user interface is designed in English. However, this can be easily customized for different languages.

## 6. Case study

SpidersRUs has been designed to help users develop specialized search engines in different languages easily and effectively. In this section, we present the use of the SpidersRUs toolkit to develop a Web search engine that specializes in medical information in the Chinese language. We will describe the background of the case study, how the toolkit was used in the development of the search engine, and an evaluation study of the collection built.

### 6.1. Background

The search engine was developed by our research group to address the need for medical information of Chinese-speaking users [39, 40]. A tremendous number of Chinese medical information resources have been created on the Web, ranging from scientific papers and journals to general health topics and clinical symposia. However, Web users are often frustrated when they try to look for Chinese health information online. There are few medical domain-specific search engines built for Chinese users. Compared to the wide availability of English medical information services such as MEDLINE and CANCERLIT, current Chinese medical information services cannot satisfy the growing medical information needs of Chinese users.

Several factors contribute to the difficulties of supporting Chinese information-seeking in the medical area. One major problem is the regional differences between mainland China, Hong Kong and Taiwan. Although the populations of all three regions use Chinese, they use different Chinese characters. People from mainland China where Simplified Chinese is used usually find it difficult

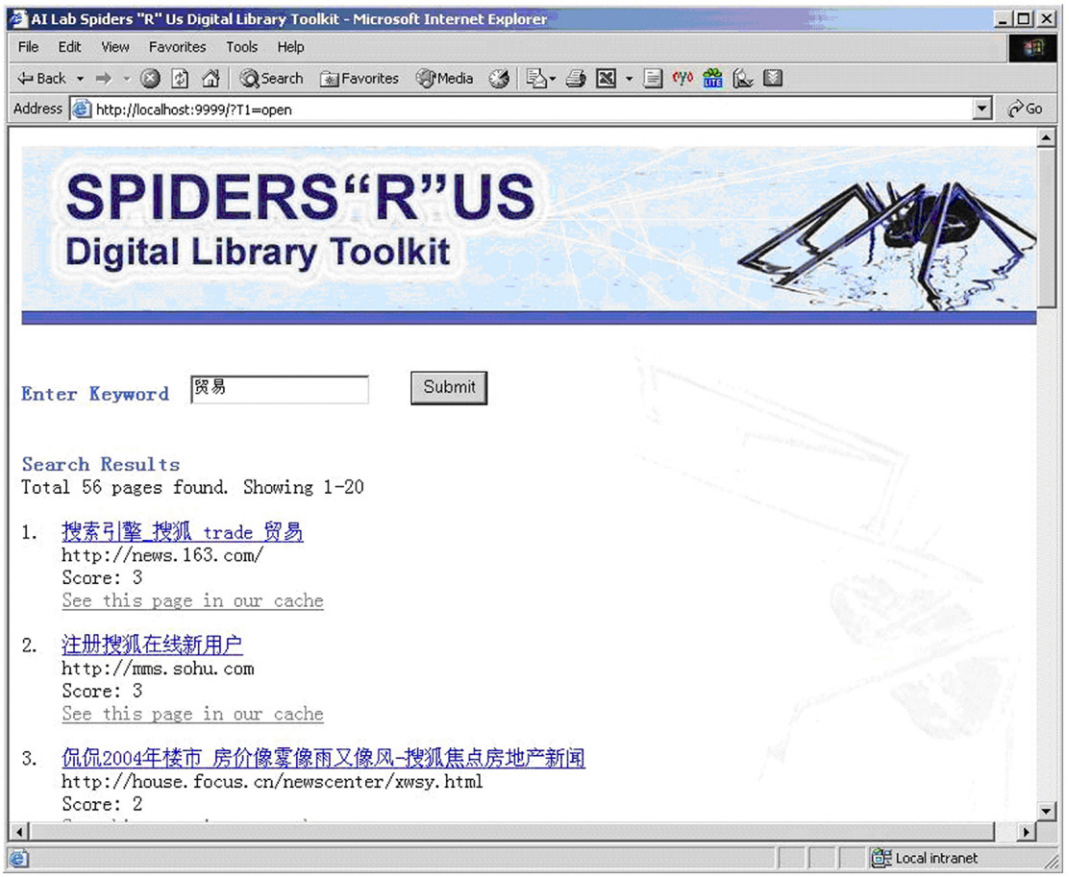


Fig. 7. Search results for a Chinese collection.

to read Traditional Chinese that is used in Hong Kong and Taiwan, while people from the latter two areas also have similar problems in searching for information written in Simplified Chinese. Moreover, Simplified Chinese and Traditional Chinese are encoded differently in computer systems. Simplified Chinese is usually encoded using the GB encoding while Traditional Chinese is usually encoded using the Big-5 encoding. When searching in a system encoded one way, users usually cannot get information encoded in the other. Building a Chinese medical Web portal that supports information searching for users in the three regions is, therefore, a challenging task. These issues make the medical domain in Chinese an ideal testbed to study the use of the SpidersRUs toolkit to build specialized search engine in non-English languages.

### 6.2. Collection building and indexing

The SpidersRUs toolkit was used to build the searchable collections from the three regions for the medical portal. After getting the suggestions from medical domain experts in these regions, 210 starting URLs were manually selected, including 87 from mainland China, 58 from Hong Kong and 65 from Taiwan. These URLs cover a large variety of medicine-related topics, such as public clinics, drug information, and hospital information. Staring with these medically related URLs, the SpidersRUs toolkit searched the Web to download documents. We assumed that medical pages included in the list were likely to point to sites that were considered useful [6,9].

Web documents from mainland China, Hong Kong and Taiwan were collected separately in order to differentiate among sources and identify encoding schemes. Therefore, the toolkit was executed three times to build three separate collections. The toolkit was first run using the URLs from mainland China, and collected around 150,000 pages in around 3 h. This collection consisted of pages written in Simplified Chinese encoded in the GB2312 format. Then the toolkit was used to build the Taiwan collection, which also downloaded approximately 150,000 pages in three hours' time. Finally, a Hong Kong

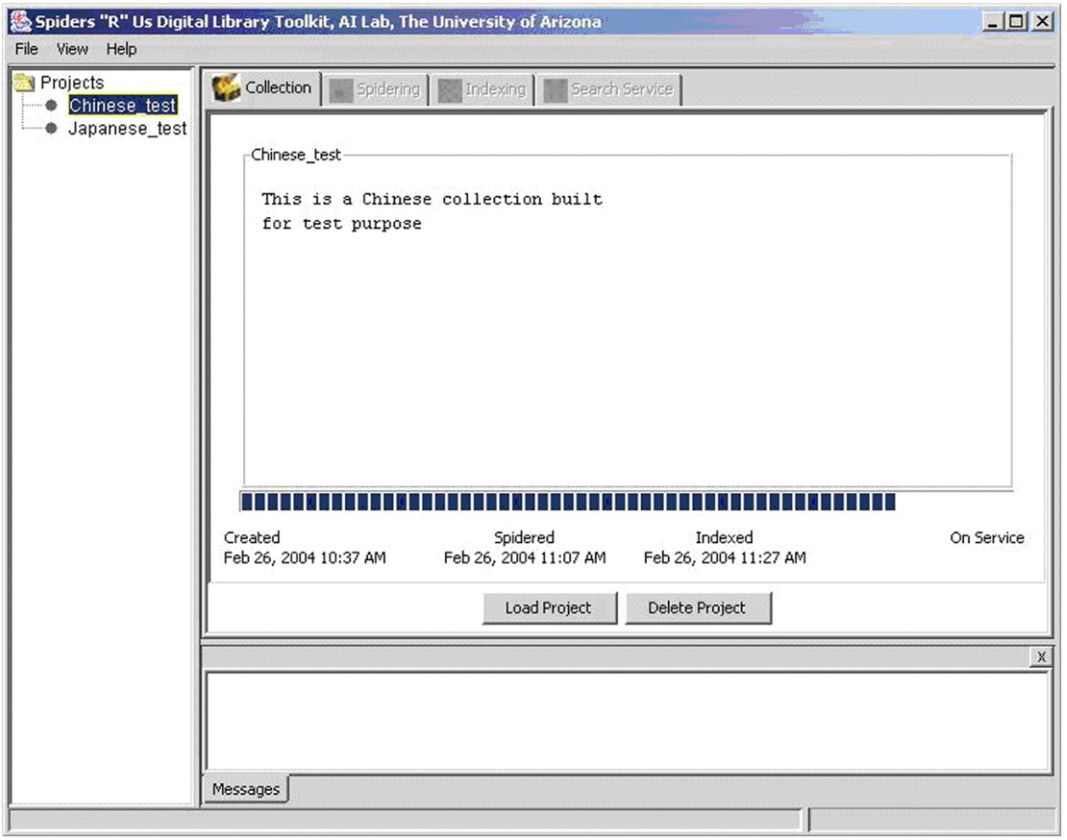


Fig. 8. Loading an existing collection.

collection was built, consisting of around 5000 pages downloaded in about 10 min. Both the Taiwan and the Hong Kong collections consisted of Web pages written in Traditional Chinese encoded using Big-5.

The spidering process was not as fast as other tests that we have performed previously on English pages, in which we were able to download about 100,000 Web pages in an hour. One of the main reasons is that both tests were run in the United States. As a result, connecting to and downloading pages from Web servers in China, Taiwan, and Hong Kong would take significantly more time than from those in the US. Nonetheless, the reported times were still within a reasonable limit for specialized search engine development and for updating (refreshing) the collection.

The collected documents were then indexed by the indexer in the toolkit. Because the toolkit provides multilingual support, no special processing was needed from the users. The indexing for the mainland China collection and the Taiwan collection each took around 2 h, while the Hong Kong collection only took a few minutes. The mainland China collection was indexed based on the GB2312 encoding while the Taiwan and the Hong Kong collections were indexed using the Big-5 format.

In the system, users could access the collections built by the SpidersRUs toolkit as well as other popular Chinese search engines such as Sina (http://www.sina.com), Yahoo Hong Kong (http://hk.yahoo.com), and Openfind (http://www.openfind.com.tw). A sample screenshot is shown in Fig. 10. The SpidersRUs collections and user interface were integrated into the interface. Users could choose from the list of search engines shown (including the ones built by SpidersRUs) and perform "meta-searching" on the collections [3,8,30]. The search results were then combined and displayed to users.

To enable cross-searching between Traditional Chinese and Simplified Chinese among the three regions, an encoding conversion program is used. The encoding converter uses a dictionary that maps between Simplified Chinese characters (in GB2312) and Traditional Chinese characters (in Big-5). In the Simplified Chinese version, when a user enters a query in Simplified Chinese, the query is sent to search the mainland China collection using

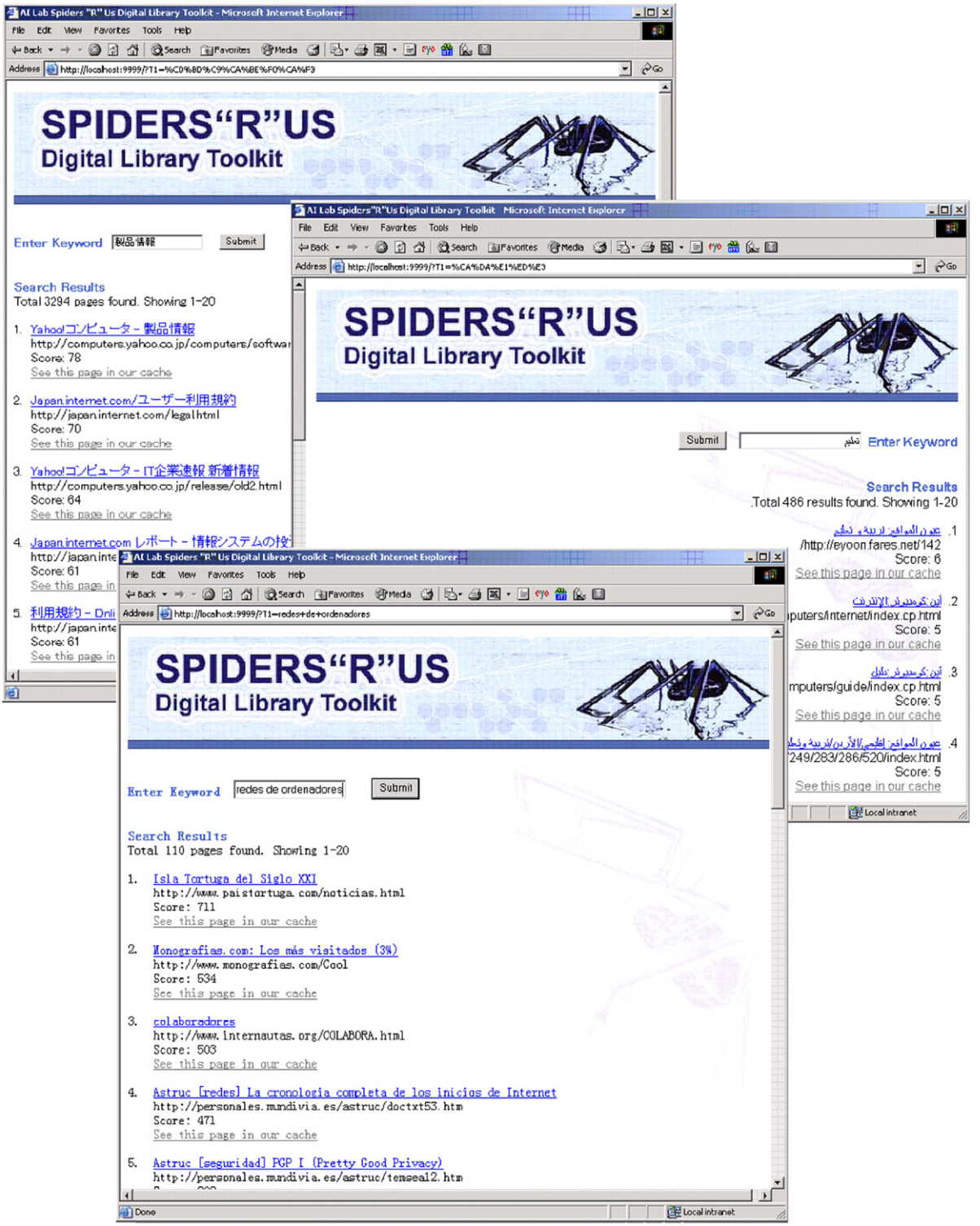


Fig. 9. Examples of collections in Japanese, Spanish and Arabic built by the toolkit.


Fig. 10. Accessing the collections built by SpidersRUs.

Simplified Chinese. At the same time, the query is converted into Traditional Chinese and used to search the Hong Kong and Taiwan collections that use traditional Chinese. After the results are retrieved, the encoding conversion program is invoked again to convert the search results from Traditional Chinese into Simplified Chinese. The whole process is transparent to the user. This encoding conversion program enables cross-regional search and addresses the problem of different Chinese characters and encodings. It also demonstrates that it is easy to integrate our toolkit with other applications.

Because documents collected were stored in their original formats, they could be easily further processed for other purposes. For example, we were able to use a Mutual Information program [25] to extract meaningful Chinese phrases from the document collections. These phrases were then saved into a separate index file and used for document summarization and document clustering [4,39].

### 6.3. Evaluation

An evaluation study was conducted to evaluate the performance of the search system. In the study, the medical search system was compared with existing Chinese Search Engines for their performance in Chinese medical information browsing and searching [39]. Forty-five subjects from mainland China, Taiwan, and Hong Kong (15 from each region) participated in the experiment. Each subject was asked to perform four search tasks. In a search task, a subject would be asked a short question that required a specific answer which could be found in the documents in the collection.

Yahoo HK, Openfind, and Sina, are chosen as our benchmarks as they are one of the most widely used search engines in Hong Kong, Taiwan, and China, respectively. Efficiency was measured by the average time that subjects needed to perform a task. Effectiveness was represented by accuracy, which was measured by the

number of correct answers given by a subject divided by the total number of questions. The evaluation results showed that our system achieved higher efficiency and effectiveness than the benchmark search engines. In terms of effectiveness, our system performed significantly better than Sina (paired $t$-test $p$-value = 0.008) and comparably to Yahoo HK and Openfind in search tasks. In terms of efficiency, our system was significantly better than Sina ($p = 0.040$) and Openfind ($p = 0.045$) and comparable to Yahoo HK. On average, our system achieved 20% higher effectiveness and was 30% more efficient than the benchmark systems. The results also showed that the document clustering tool could improve users' performance when browsing. More details about the evaluation can be found in [39,40].

The case study has shown that it is easy and convenient for search engine developers to use the SpidersRUs tool to create a specialized search engines in non-English languages. In the case study, the collections created from SpidersRUs were combined with other search engines and were able to improve users' searching and browsing performance. It also demonstrated that the intermediate results of the process, including the documents downloaded from the Web and the terms extracted from these documents, could be used to integrate with other applications to provide advanced functionalities such as document summarization and document clustering.

## 7. Discussions

We have presented the architecture design of the SpidersRUs toolkit, a sample user session of the system, and a case study on a Chinese medical search engine. In this section, we will discuss the performance of our system and some lessons learned.

### 7.1. Performance

Speed is an important issue in search engine development. Because the size of the Web keeps growing, tools for search engine creation, even only for personal or domain-specific search engines, need to scale up to thousands or even millions of pages. When we developed the SpidersRUs toolkit, we designed it with the aim to build collections up to around half to one million documents. To achieve this objective, we spent a lot of time in optimizing the codes for each module in order to improve performance, both in terms of speed and memory usage. One major bottleneck we encountered during the early stage of our project is the I/O operations. In order to improve performance by minimizing disk seek time and read/write time, all I/O operations had to be buffered. We also ensured that the different modules did not have any memory leakage. As the tool has to process one million documents, any small leakage in memory would be multiplied by a million times, which would significantly affect the tool's performance.

For the Spider module, we had to make sure that each spider agent was light-weighted and would not consume too many resources. Another problem with the Spider module was that some working memory, such as the spider queues, was too large to be stored entirely in memory. However, it would also be very inefficient to store them completely in the local disk because the reading time would be too slow. To address the problem, we stored part of the information in the memory and part in the local disk. The information that was more likely to be used (e.g., the head elements of a queue) would be cached in the program's memory. Less frequently used information would be stored in the local disk, and read into memory only when needed.

Sorting the forward index to become the inverted index is not an easy task when the index is large. To minimize the memory usage of this process, the forward index is first split into buckets based on the term id. For example, the first bucket will contain the relations for the terms whose term id is in the range of 1 to 100, the second bucket will contain those between 101 and 200, and so on. Each bucket is then ordered individually based on term id. After all buckets are sorted, they are combined to form the final inverted index.

As can be seen from our discussion, there is always a tradeoff between speed and memory. We tried to design our toolkit to run on high-end personal computers (e.g., Intel Pentium 4 at the time of our design) rather than powerful computer servers or low-end personal computers. In an evaluation test, the final tool was able to develop a collection of one million pages within two days (approximately one day for spidering and one day for inverted indexing). While this is not as fast as some sophisticated applications (e.g., the Mercator which uses an optimized version of the Java Virtual Machine [16,17]), it is well suited for specialized search engine development.

### 7.2. Developing multilingual systems

There are several major issues in developing multilingual systems. First, each language has its own characteristics for text tokenization, so special attention need to be paid to the tokenization method in the indexing process. As discussed earlier, there is no space between words for some languages. Second, different encodings

exist even for the same language. For example, Japanese can be encoded in EUC or Shift-JIS. Also, every language can be encoded in its specific encoding as well as Unicode. Third, it is important to know what encoding is used when reading a series of bytes from the Web or from the local disk. If the correct encoding was not known, it would be difficult to convert the bytes back to its original characters.

To address the first problem, our toolkit index documents in different languages in different ways. For languages where words are separated by spaces, spaces are used to indicate word boundary for the Indexer. For other languages, the indexer will assume that a word boundary exists between every pair of double-byte characters and will index each of them. While this approach may not be very efficient for some languages as some characters may appear very frequently, it would apply to any languages easily.

To address the last two problems, the encoding of a document must be known. While it is possible to let the system automatically detect the encoding of a document, either by checking the characters used in the document or by looking at the meta-information of a Web page, such methods do not always give the correct encoding. It is not uncommon that Web page authors put the wrong encoding information in the meta-data of the HTML file. Therefore, we allow the user to specify manually the encoding to be used for each collection. Currently, our system can only handle a set of documents in the same encoding. This includes handling English plus an Asian language (e.g., English and Traditional Chinese as both can be represented in Big-5 encoding), but not two languages with different encodings like Japanese and Chinese. If the documents collected are encoded using Unicode (which can be used to code all languages), our system can handle multiple languages. One way of doing this is to detect the encoding of the source documents and convert all documents into Unicode. Such a detection component will be a possible future direction for our development.

Java, as mentioned earlier, is a programming language designed for multiple languages. Each character is treated as a double-byte Unicode character in Java. When reading a stream of bytes from the Web or from disk, a Java program can easily convert the bytes to the Unicode characters of a particular language according to the encoding specified by the user. When the program needs to save a character string (e.g., a term or the title of a document) to the local disk, the Unicode characters, be it Chinese, English, or Arabic, can be converted back to the byte streams based on the encoding specified by the user. This has made it very convenient to develop multilingual information systems in Java.

### 7.3. Implications for system developers and users

New collections can be created easily using software tools such as SpidersRUs. With a set of URLs defined by the users, a new collection can be built in a few minutes to a few hours. System developers and Web site administrators can use the tool to build a specialized Web search engine in a language they choose. In addition, researchers can use the tool to collect documents from the Web for other purposes. For example, Web documents can be used as a linguistic resource for various linguistic and information retrieval research (e.g., [14]). SpidersRUs can help users download a selected set of documents from the Web conveniently. Also, as demonstrated in our case study, the tool can be used to create a forward index and an inverted index in any languages that can be used in other applications such as document clustering. While many other researchers have used SMART [29] for such purposes for English documents, SpidersRUs can be used for any languages. As the files and the indexes created by SpidersRUs are in plain text format, they can be easily integrated into any other applications.

The tool also encourages users to create their collections and own search engines. Similar to Greenstone [37], SpidersRUs supports collection building in multiple languages and we plan to promote it to different parts of the world. The tool can be used to build a search engine for a single Web site, a specific domain, a digital library, an archival, or multimedia documents (with minor modifications). However, we also have another intention. In addition to allowing users to create their own searchable collections in their own languages, SpidersRUs essentially provides the tools for users to spider, index, and search a collection in their own languages. This gives system developers and users more power over other tools such as Greenstone (which store their information in binary format), as users can reuse the collected files and the search indexes in any way they like. They can combine them with any of their own applications, such as an information retrieval system, an e-commerce system, or other Web applications.

One limitation of our tool is that we currently do not support incremental update. A complete re-run is needed for refreshing the search engine index. However, due to the small size of the collections intended for our tool, this would only take a reasonable amount of time. We are currently working on revising the tool to support incremental refresh and update.

## 8. Conclusion and future directions

In this paper, we have reviewed existing research in specialized, multilingual search engines and proposed an architecture for search engine development. We have presented the design and a sample user session of the SpidersRUs toolkit, and have shown several search engines developed by the toolkit in different languages. A case study on using the toolkit for developing a Chinese medical search engine also has been discussed. Our result has demonstrated that toolkit can be used for developing search engine effectively and efficiently in multiple languages. Some technical issues involved in our study, such as the optimization of the toolkit and the use of Java in developing multilingual systems, are also discussed.

Our future work has several directions. First, we would like to further enhance the performance of our system. While the current performance level is satisfactory, it is possible to further improve the speed of the spidering and indexing processes and to make the system more stable. We also plan to make the system more customizable by allowing users to have more control over the options that can be selected in the system.

Second, we would like to expand our toolkit for multimedia information. In recent years, more and more multimedia data, such as music, images, and movies, are available on the Web. It is important to create a toolkit that allows users to develop easily search engines for multimedia materials. As most current tools rely on meta-data (e.g., [35]), it would be useful to have a toolkit that can help users collect and index multilingual, multimedia data automatically.

Third, we are improving our toolkit such that more language-specific or document-specific features can be set. For example, specific indexers (such as one that employs phrase-based indexing) can be defined for different languages according to their characteristics such that the indexing process can be more effective. Customized indexers can also be used to handle semi-structured documents such as XML and fielded-texts in database by extracting text from every field in the data source and storing them into the Index Structure. We also plan to implement cross-language information retrieval (CLIR) functionality into the toolkit [27]. For example, users would be able to retrieve Japanese documents by inputting an English query, and vice versa.

Lastly, we are working on developing a version of our toolkit that is OAI-compliant [31]. OAI, or the Open Archive Initiative, provides some standards for search engine and digital library systems. A toolkit that can collect information from OAI-compliant Web sites as well as provide information to other users according to the OAI standard would be very useful for the exchange and sharing of data among search engines and digital libraries. This would be especially useful for communities where a large number of domain-specific digital libraries are involved, such as the National SMETE Digital Library (NSDL) community.

## Acknowledgements

This project has been supported in part by the following grants:

- NSF Digital Library Initiative-2, "High-performance Digital Library Systems: From Information Retrieval to Knowledge Management," IIS-9817473, April 1999–March 2002.
- NSF National SMETE Digital Library: "Intelligent Collection Services for and about Educators and Students: Logging, Spidering, Analysis and Visualization," DUE-0121741, September 2001–August 2003.

We would like to thank Chia-Jung Hsu for his contribution to this project and the user manual, and Maryan Mikhael and Shingka Wu for the graphic design. We would also like to thank other members of the Artificial Intelligence Lab at the University of Arizona who have tested the toolkit and shared with us their ideas and comments.

## References

[1] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, S. Raghavan, Searching the Web, ACM Transactions on Internet Technology 1 (1) (2001) 2–43.
[2] S. Brin, L. Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, Proceedings of the 7th WWW Conference, Brisbane, Australia, Apr 1998.
[3] M. Chau, D. Zeng, H. Chen, Personalized spiders for web search and analysis, Proceedings of the First ACM/IEEE-CS Joint Conference on Digital Libraries, Roanoke, Virginia, USA, 2001, pp. 79–87, June 24–28, 2001.
[4] M. Chau, H. Chen, J. Qin, Y. Zhou, W.K. Sung, Y. Chen, Y. Qin, D. McDonald, A. Lally, M. Landon, NanoPort: a web portal for nanoscale science and technology, Proceedings of The Second ACM/IEEE-CS Joint Conference on Digital Libraries, Portland, Oregon, USA, July 14–18, 2002, 2002, p. 373.
[5] M. Chau, H. Chen, Personalized and focused web spiders, in: N. Zhong, J. Liu, Y. Yao (Eds.), Web Intelligence, Springer-Verlag, 2003, pp. 197–217, February 2003.
[6] M. Chau, H. Chen, Comparison of three vertical search spiders, IEEE Computer 36 (5) (2003) 56–62.
[7] H. Chen, M. Chau, D. Zeng, CI Spider: a tool for competitive intelligence on the web, Decision Support Systems 34 (1) (2002) 1–17.
[8] H. Chen, H. Fan, M. Chau, D. Zeng, MetaSpider: meta-searching and categorization on the Web, Journal of the American Society of Information Science & Technology 52 (13) (2001) 1134–1147.

[9] H. Chen, A. Lally, B. Zhu, M. Chau, HelpfulMed: intelligent searching for medical information over the Internet, Journal of the American Society for Information Science and Technology 54 (7) (2003) 683–694.
[10] H. Chen, M. Ramsey, P. Li, The Java search agent workshop, in: P. Gabriella, F. Crestani (Eds.), Soft Computing in Information Retrieval, Physica-Verlag, 2000, pp. 122–140.
[11] F.C. Cheong, Internet Agents: Spiders, Wanderers, Brokers, and Bots, New Riders Publishing, Indianapolis, Indiana, USA, 1996.
[12] D. Czarnecki, A. Deitsch, Java Internationalization, O'Reilly & Associates, Sebastopol, California, USA, 2001.
[13] P. DeBra, R. Post, Information retrieval in the World-Wide Web: making client-based searching feasible, Proceedings of the First International World Wide Web Conference, Geneva, Switzerland, 1994.
[14] F. Duclaye, F. Yvon, O. Collin, Using the Web as a linguistic resource for learning reformulations automatically, Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-2002), Las Palmas, Canary Islands, Spain, May 2002, 2002, pp. 390–396.
[15] D. Eichmann, Ethical web agents, Proceedings of the 2nd International World Wide Web Conference, Chicago, Illinois, USA, 1994.
[16] A. Heydon, M. Najork, Performance limitations of the Java core libraries, Proceedings of the 1999 ACM Java Grande Conference, Jun 1999, 1999, pp. 35–41.
[17] A. Heydon, M. Najork, Mercator: a scalable, extensible Web crawler, World Wide Web 2 (4) (1999) 219–229.
[18] ht://dig Group. "htdig Reference." [Online], Available at http://www.htdig.org/htdig.html.
[19] Internet Archive, Heritrix Wiki, [Online], Available at http://crawler.archive.org/cgi-bin/wiki.pl, 2006.
[20] S.C. Kleene, Representation of events in nerve nets and finite automata, Automata Studies, Annals of Mathematics Studies 34 (1956) 3–42.
[21] M. Koster, A Standard for Robot Exclusion, [Online], Available at http://www.robotstxt.org/wc/norobots.html, 1994.
[22] U. Manber, M. Smith, B. Gopal, WebGlimpse: combining browsing and searching, Proceedings of the USENIX 1997 Annual Technical Conference, Anaheim, California, 1997, Jan 1997.
[23] M.L. Mauldin, Spidering BOF report, Report of the Distributed Indexing/Searching Workshop, Cambridge, Massachusetts, USA, May 1996, 1996.
[24] Nutch, Welcome to Nutch! [Online], Available at http://lucene.apache.org/nutch/index.pdf, 2006.
[25] T. Ong, H. Chen, Updatable PAT-Tree approach to Chinese key phrase extraction using mutual information: a linguistic foundation for knowledge management, Proceedings of the Second Asian Digital Library Conference, Taipei, Taiwan, Dec 1999, pp. 63–84.
[26] M.F. Porter, An algorithm for suffix stripping, Program 14 (3) (1980) 130–137.
[27] J. Qin, Y. Zhou, M. Chau, H. Chen, Multilingual web retrieval: an experiment in English–Chinese business intelligence, Journal of the American Society for Information Science and Technology 57 (5) (2006) 671–683.
[28] G. Salton, Automatic Text Processing, Addison-Wesley, Reading, Massachusetts, USA, 1989.
[29] G. Salton, M.J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, New York, USA, 1983.
[30] E. Selberg, O. Etzioni, Multi-service search and comparison using the MetaCrawler, Proceedings of the 4th World Wide Web Conference, Boston, Massachusetts, USA, Dec 1995.
[31] H. Suleman, E.A. Fox, A Framework for Building Open Digital Libraries, D-Lib Magazine 7 (12) (2001), Available at: http://www.dlib.org/dlib/december01/suleman/12suleman.html.
[32] Swish-e Development Team, The Swish-e Documentation, [Online], Available at http://swish-e.org/current/docs/index.html, 2002.
[33] The Unicode Consortium, The Unicode Standard. Worldwide Character Encoding, Addison-Wesley, 1992.
[34] Vestris Inc., Alkaline: A UNIX/NT Search Engine, [Online], Available at http://alkaline.vestris.com/docs/pdf/alkaline.pdf, 2001.
[35] I.H. Witten, R.J. McNab, S.J. Boddie, D. Bainbridge, Greenstone: a comprehensive open-source digital library software system, Proceedings of the ACM Digital Libraries Conference, San Antonio, Texas, USA, 2000.
[36] I.H. Witten, D. Bainbridge, S.J. Boddie, Greenstone: open-source DL software, Communications of the ACM 44 (5) (2001) 47.
[37] I.H. Witten, D. Bainbridge, S.J. Boddie, Power to the people: end-user building of digital library collections, Proceedings of the Joint Conference on Digital Libraries, Roanoke, Virginia, USA, June 2001, 2001, pp. 94–103.
[38] C.C. Yang, J. Yen, H. Chen, Intelligent Internet searching agent based on hybrid simulated annealing, Decision Support Systems 28 (3) (2000) 269–277.
[39] Y. Zhou, J. Qin, H. Chen, CMedPort: an integrated approach to facilitating Chinese medical information seeking, Decision Support Systems 42 (3) (2006) 1431–1448.
[40] Y. Zhou, J. Qin, H. Chen, Z. Huang, Y. Zhang, W. Chung, G. Wang, CMedPort: a cross-regional Chinese medical portal, Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries, Houston, Texas, May 27–31, 2003, 2003, p. 379.

**Michael Chau** is an Assistant Professor and the BBA(IS)/BEng(CS) Coordinator in the School of Business at the University of Hong Kong. He received his Ph.D. degree in management information systems from the University of Arizona and a Bachelor Degree in Computer Science and Information Systems from the University of Hong Kong. His current research interests include information retrieval, Web mining, data mining, knowledge management, electronic commerce, security informatics, and intelligent agents. He has published more than 60 research articles in leading journals and conferences, including *IEEE Computer*, *Journal of the America Society for Information Science and Technology*, *Decision Support Systems*, and *Communications of the ACM*. More information can be found at http://www.business.hku.hk/~mchau/.

**Jialun Qin** is an Assistant Professor in the Department of Management at University of Massachusetts Lowell. He received his Ph D. degree in Management Information Systems from the University of Arizona. His research interests include knowledge management, data and Web mining, digital libraries, and human computer interaction. His publications have appeared in *Decision Support Systems*, *Journal of the American Society for Information Science and Technology*, and *IEEE Intelligent Systems*. Contact him at Jialun_qin@uml.edu.

**Yilu Zhou** is an Assistant Professor in the Department of Information Systems and Technology Management at George Washington University. Her current research interests include multilingual knowledge discovery, Web mining, text mining and human computer interaction. She received a Ph.D. in Management of Information System from the University of Arizona, where she was also a research associate of the Artificial Intelligence Lab. She received a B.S. in Computer Science from Shanghai Jiaotong University. Contact her at yzhou@gwu.edu.

**Chunju Tseng** is an Assistant Research Scientist in the Department of Management Information Systems at the University of Arizona. His research interests include Web mining, Infectious Disease Surveillance and human computer interaction. He received his B.S. degree from the National Taiwan University and M.S. in Management Information System from the University of Arizona. He has published articles in Joint Conference on Digital Libraries and presented in many conferences including International Conference on Digital Government Research and Conference on Human Factors in Computing Systems.

**Hsinchun Chen** is McClelland Professor of Management Information Systems at the University of Arizona and Andersen Consulting Professor of the Year (1999). He received the B.S. degree from the National Chiao-Tung University in Taiwan, the MBA degree from SUNY Buffalo, and the Ph.D. degree in Information Systems from the New York University. Dr. Chen is a Fellow of IEEE and AAAS. He received the IEEE Computer Society 2006 Technical Achievement Award. He is author/editor of 13 books, 17 book chapters, and more than 130 SCI journal articles covering intelligence analysis, biomedical informatics, data/text/web mining, digital library, knowledge management, and Web computing. Dr. Chen was ranked #8 in publication productivity in Information Systems (CAIS 2005) and #1 in Digital Library research (IP&M 2005) in two recent bibliometric studies. He serves on ten editorial boards including: *ACM Transactions on Information Systems*, *IEEE Transactions on Systems, Man, and Cybernetics*, *Journal of the American Society for Information Science and Technology*, and *Decision Support Systems*. Dr. Chen has served as a Scientific Counselor/Advisor of the National Library of Medicine (USA), Academia Sinica (Taiwan), and National Library of China (China). He has been an advisor for major NSF, DOJ, NLM, DOD, DHS, and other international research programs in digital library, digital government, medical informatics, and national security research. Dr. Chen is the founding director of Artificial Intelligence Lab and Hoffman E-Commerce Lab. He is conference co-chair of ACM/IEEE Joint Conference on Digital Libraries (JCDL) 2004 and has served as the conference/program co-chair for the past eight International Conferences of Asian Digital Libraries (ICADL), the premiere digital library meeting in Asia that he helped develop. Dr. Chen is also (founding) conference co-chair of the IEEE International Conferences on Intelligence and Security Informatics (ISI) 2003–2007. Dr. Chen has also received numerous awards in information technology and knowledge management education and research including: AT&T Foundation Award, SAP Award, the Andersen Consulting Professor of the Year Award, the University of Arizona Technology Innovation Award, and the National Chaio-Tung University Distinguished Alumnus Award. Further information can be found at http://ai.arizona.edu/hchen/.