Decision Support Systems 48 (2010) 369–382

Contents lists available at ScienceDirect

Decision Support Systems

journal homepage: www.elsevier.com/locate/dss

Designing the user interface and functions of a search engine development tool

Michael Chau *, Cho Hung Wong

School of Business, The University of Hong Kong, Pokfulam, Hong Kong

ARTICLE INFO

Article history:
Received 29 January 2009
Received in revised form 27 August 2009
Accepted 14 October 2009
Available online 25 October 2009

Keywords:
Web search
Search engine development tools
User evaluation

ABSTRACT

Search engine development tools have been made to allow users to build their own search engines. However, most of these tools have been designed for advanced computer users. Users without a full understanding of topics such as Web spidering would find these tools difficult to use due to different issues in terms of user interface, performance, and reliability. In view of these issues, we presented a tool called SpidersRUs to strike a balance between usability and functionality. On one hand, beginners should be able to operate the tool by using the basic functions needed to build a search engine. On the other, advanced users should be given the options to exert a higher level of customization while working on the tool. To study the interface design of SpidersRUs, we compared its usability and functionality from the users' perspective with two other development tools, namely Alkaline and Greenstone, in an evaluation study. Our study showed that SpidersRUs was preferred over the other two, particularly in areas of screen layout and sequence, terminology and system information, and learning to use the system.

1. Introduction

The rapid development of the Internet and the large amount of information on the World Wide Web have led to the problem of information overload [3]. This motivated the development of search engines such as Google (www.google.com) and AltaVista (www.altavista.com), many of which have become a commercial success. While they can satisfy users' queries for general purposes, most of these search engines do not allow users to specify the intended search topic, thereby generating a large number of results which the user may consider irrelevant. For instance, searching for the keyword 'option' may retrieve results related to finance (as in "call option") or interface design (as in "option button"), among other topics. In view of the problem, most general-purpose search engines try to offer some solutions. For example, a search engine can suggest a list of more specific keywords, which leads to search results in a particular topic. For example, upon searching for "zodiac", Yahoo! (www.yahoo.com) suggested a list of terms including 'zodiac movie' and "zodiac signs", so users would get results related to either the movie or the star signs. Another method is to allow users to specify a domain (Web host) address so all search results are obtained from that domain. For example, searching for the movie "Zodiac" from The Internet Movie Database (www.imdb.com) can be entered in Google as "zodiac site:www.imdb.com".

The two methods above, however, do not offer the complete remedy. In the first one, adding another keyword as the topic can result in Web pages that are still too general. The second solution allows domain-specific searches, but the search results can be limited, and in some cases users may not find a good Web domain for their searching.

Vertical search engines address the potential needs for topic-specific searching by restricting the search collection and results to a specific topic [7]. Healthline (www.healthline.com), for instance, supports searching for only health-related topics. LawCrawler (lawcrawler.findlaw.com) is a vertical search engine for legal information. The development of vertical search engines is useful for decision support in particular topics because they can provide customized information and analyses in these topics. By collecting only Web pages in a given topic, it is often possible to collect documents more accurately and comprehensively. Analyses that are specific to the topic also can be performed more easily in vertical search engines. For example, medical term suggestions, ontology-based analysis and clustering have been developed for a medical search engine called HelpfulMed in the medical area [12]. These search engines often serve as useful decision support tools [19].

To help the development of vertical search engines, a number of development tools for building vertical search engines have been created. Building a search engine involves a number of technical details. While the development tools can be designed to handle everything automatically so users can build their own search engines easily, the lack of user involvement may result in poor collections and a low level of customization. On the other hand, development tools that assume users' possession of expert knowledge in programming may render the tools unusable. Therefore, it would be interesting to study how to design a good search engine development tool to strike the balance.

In this paper, we present our work in designing a vertical search engine tool, called the SpidersRUs Digital Library Toolkit, with a focus on user evaluation. We describe in detail the design and the user

* Corresponding author.
E-mail addresses: mchau@business.hku.hk (M. Chau), joewch@graduate.hku.hk (C.H. Wong).

0167-9236/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.dss.2009.10.001

interface of the tool and report a study that compares the tool with two widely used vertical search engine development tools, namely the Greenstone Digital Library Software and the Alkaline Search Engine Server. Instead of looking at the effectiveness of the collection quality and search performance of these tools, we focus on evaluating the user satisfaction when using these tools in the development process. In particular, we emphasize on areas of the tools such as user interface, documentations and support.

The rest of this paper is structured as follows. Section 2 reviews the major components of a search engine, including the Web spider, indexer and query engine, and existing vertical search engine development tools. Section 3 specifies the research focus and value of this study. Section 4 presents in detail the system design of SpidersRUs Digital Library Toolkit. Section 5 discusses the methodology to study the comparative usability of the three tools, and reports and discusses the results from the study. In Section 6, we provide some guidelines for user interface design for search engine development tools. We conclude the paper with some discussions of future research in Section 7.

2. Related work

This section starts by presenting the major components commonly seen in vertical search engine development tools. We then present two existing tools that are available and have been discussed in previous research.

2.1. Components of a search engine development tool

A common search engine development tool consists of a number of Web spiders, an indexer and a query engine, which help users in collecting, indexing, and querying Web pages, respectively [1]. Fig. 1 shows the overall architecture of a typical vertical search engine development tool.

A *Web spider* (also known as Web crawler) is a program that automatically retrieves documents (such as HTML and PDF) in the World Wide Web, which is a process called Web spidering or Web crawling [5,13]. In case of building a vertical search engine, a list of URLs (known as seeds) that link to documents in the intended topic should be specified as starting points for the Web spiders. Building a search engine about sports, for example, will need URLs pointing to Web sites such as Yahoo! Sports and ESPN.com. Each spider will then fetch the documents from the Web, identify the hyperlinks stored in them and visit these documents in turn by following the hyperlinks. As the Web spider makes a visit, it retrieves the document and stores a copy in the executing computer. Using some filtering and selection techniques during the process of Web spidering, a collection of documents is created. For example, URLs pointing to invalid or irrelevant documents can be removed [10,11,18]. As the seeds are related to a specific topic, it is likely that the majority of documents in the collections are topic-specific. Some advanced focused crawling techniques may be used in the spidering process [4,6,20], though these techniques are generally not available in popular search engine development tools.

An *indexer* creates a search index (in a process called indexing) for the collection of documents obtained during Web spidering. The first step is to extract a list of key words from the collection of topic-specific documents so as to create an initial index, which maps each document to the list of words. The next step is to create an inverted index which maps a word to a list of documents containing that word [21]. The aim of indexing is to enable efficient searching. Without an index, a search query will have to be handled by traversing through all the documents, which is not time-efficient.

A *query engine* is the main program that handles users' queries [1]. It takes search queries (usually as one or more keywords) and finds relevant documents by looking up in the inverted index. Although the query engine is the only program that accepts search queries directly from users, the performance of searching (e.g., relevance of search results, searching time) largely depends on the implementation of the Web spiders and indexer, in addition to hardware performance of the server machines running the query engines. In most cases, users access the query engine using a Web browser, which displays the search results as a Web page.

2.2. Existing tools

In this subsection, we review two popular building tools that have been widely used in previous research, namely the Greenstone Digital Library Software (Greenstone) and the Alkaline Search Engine Server (Alkaline) [8,23,24].

Greenstone is an open-source multilingual software application for building digital libraries. The application was developed by the New Zealand Digital Library Project at the University of Waikato [23–26]. While it fully supports four different languages (English, French, Spanish and Russian), it also offers a graphical interface in more than 40 languages, created by a large number of volunteers. In terms of multi-platform support, Greenstone is available for Microsoft Windows, PowerPC Mac OS X, and Linux. The application also comes with a lot of online recourses and documentation. In addition to HTML, Greenstone allows users to build digital libraries supporting file formats such as Word, PDF and PowerPoint, as well as multimedia documents including JPEG, MP3 and MPEG. The tool was designed for users with different levels of computer literacy [2]. While general users will be able to build their own search engines, advanced users with knowledge in HTML or Perl are allowed to exert greater customization. Figs. 2–4 show the screenshots of Greenstone during the process of building a search engine.

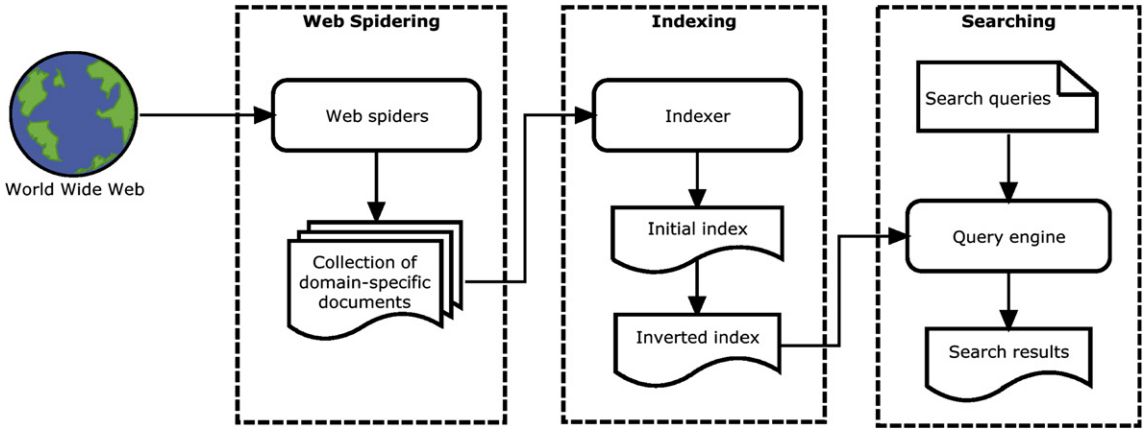

Fig. 1. Overall architecture of a common vertical search engine development tool.

Fig. 2. Greenstone – creating a new collection.

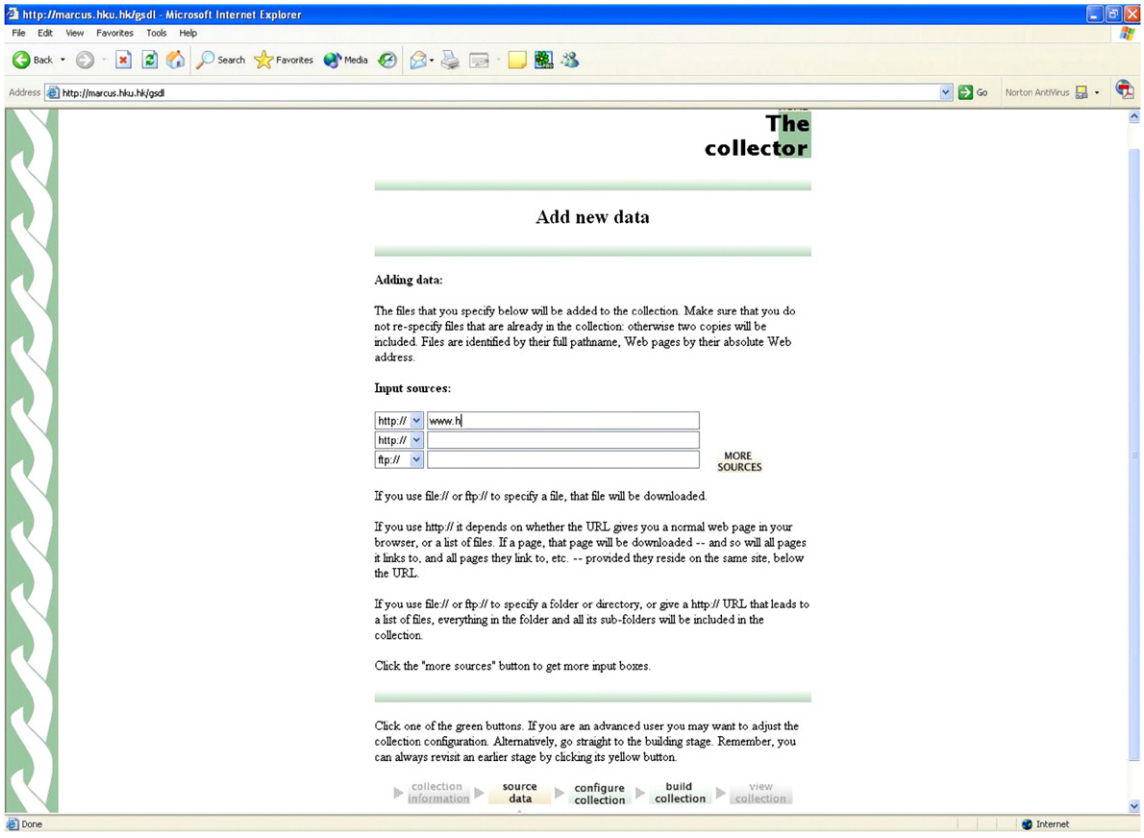

Fig. 3. Greenstone – adding seeds.

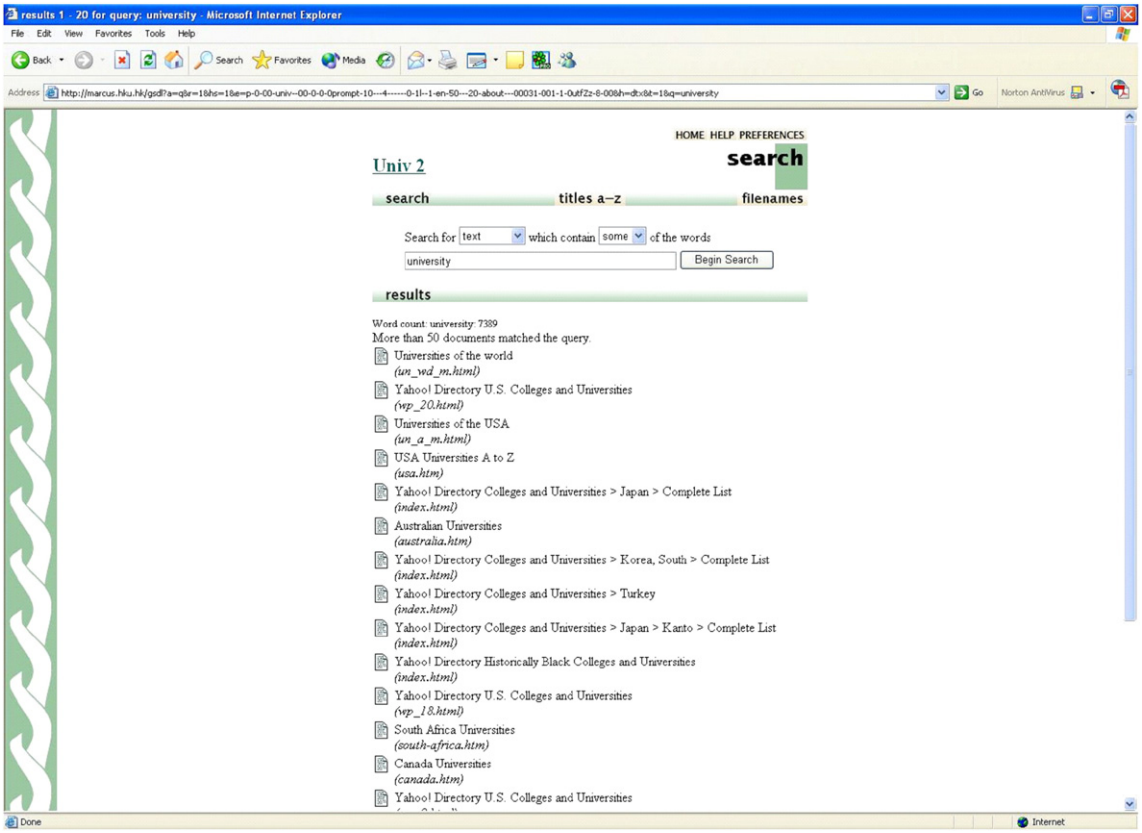

Fig. 4. Greenstone – search results.

Alkaline is a freeware developed by Vestris Inc. in Switzerland. The application was designed for non-commercial use, although purchase of licenses is also available for commercial users. The multi-platform tool supports most popular operating systems. The latest release (version 1.9) is available for use on six operating systems (including UNIX, Microsoft Windows NT/2000/XP and Sun Solaris). As for functionality, Alkaline supports a wide range of searching including Simple Search, Boolean Search, Meta Data Search, and Numeric Data Search. Similar to Greenstone, Alkaline supports the searching of HTML, PDF, Word and multimedia files. Alkaline was designed for users with more advanced computer skills. In order to enable case-sensitive searching, for example, users are required to have some knowledge in HTML and Perl. The official user guide (http://alkaline.vestris.com/docs/alkaline/index.html) and FAQs (http://alkaline.vestris.com/faq.html) are the major official resources available online. The majority of resources available on the official Web sites are devoted to technical specifications and support, such as installation, upgrade, server configuration, performance optimization, customization, indexing, searching and troubleshooting. Materials suitable for general users, on the other hand, are relatively limited.

Alkaline is based on text command prompt and does not provide a graphic user interface during the search engine development process. To create a new collection in Alkaline, users need to create a new directory under the Alkaline main directory. The new directory should contain the configuration file "asearch.cnf" and the search engine main page "search.html". Two screenshots of what users can see in the command prompt are shown in Figs. 5 and 6. Fig. 7 shows the search results page that users can see when performing a search.

While much extant research has been devoted to the study of general-purpose search engines, very little has been done on the development tools and their usability. A few examples include the Greenstone User Survey conducted by the School of Library and Information Science at the University of North Carolina in late 2004 [22]. The objective of the study was to obtain users' feedback about "adequacy of current support structures and mechanisms" of the Greenstone Digital Library Software. In particular, a survey was conducted among members of the Greenstone community. Questions of the survey were mostly related to user manuals, availability of training and support, as well as suggestions for future improvement. The study had a greater emphasis on external aspects of the software, while users' opinions about the software itself were omitted.

3. Research questions

Although vertical search engine development tools have been available for general users for many years, there has not been an adequate study on the users' satisfaction towards the tools. In particular, we believe that areas such as user interface and functionality are important parts of the user experience. In this paper, we report our design and evaluation of a vertical search engine development tool, with a focus on the evaluation in five areas: users' overall reaction, screen layout, terminology and system information, learning to use the system, and system capabilities. We compare SpidersRUs against Greenstone and Alkaline as the benchmarks. In general, we aim to evaluate the relative usability and user satisfaction of the three tools in order to compare their quality from the users' point of view.

Greenstone and Alkaline were chosen because of several reasons. First, these two tools were widely used and reported in previous studies [8,24]. Second, they were among the few search engine development tools that were freely available at the time of the study. We believe these two are the best tools that are most suitable for our study. In addition, the two tools cover the whole search engine development process. Some other tools focus only on a part of the process (either spidering or indexing) and are therefore not suitable.

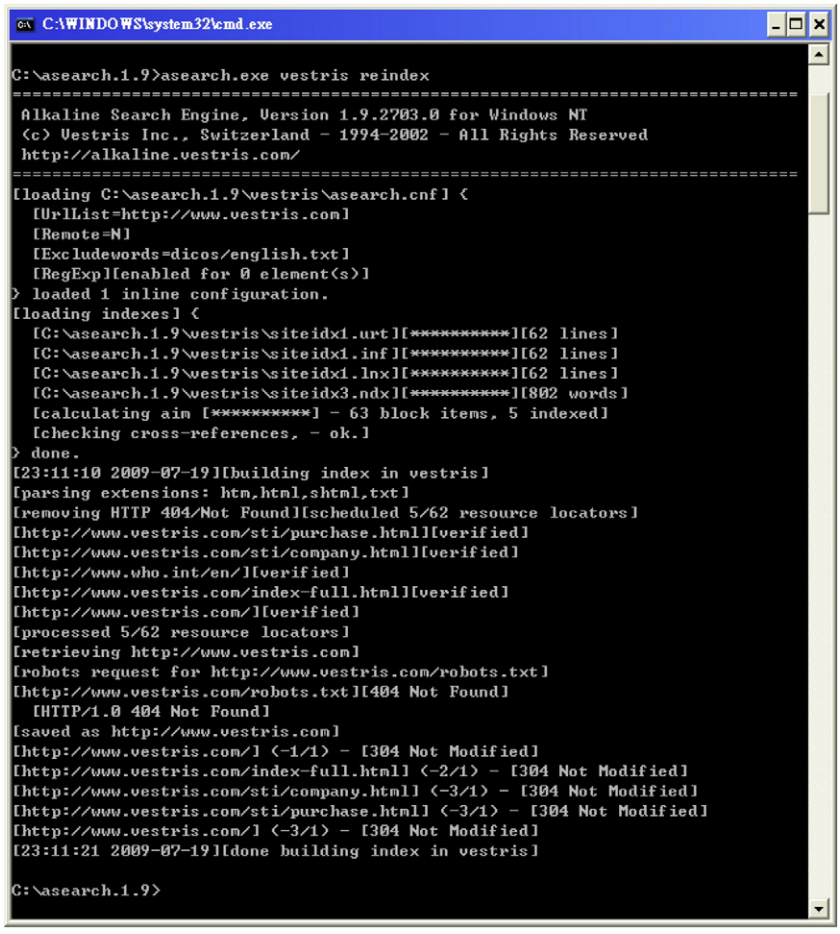

Fig. 5. Alkaline – spidering.

Lastly, the two tools have a similar set of features as SpidersRUs. This allows for better comparison among the three tools.

4. System design

This section discusses the system architecture of the search engine development tool SpidersRUs that we designed. SpidersRUs is a collection of modular tools designed for different functions (e.g., spidering, indexing, searching). The tool can be used to build search engines in multiple languages (e.g., European, Middle East and Asian languages). Same as the two applications discussed above, SpidersRUs can be run on multiple platforms, such as Windows and Linux. SpidersRUs was written in Java, a platform-independent programming language. Java adopts double-byte for character storage, which is useful for multilingual systems [15]. Developed as a collection of several components, the application stores intermediate results as text files after each step of the development process, thus offering users a higher degree of customization. SpidersRUs was designed to suit the need of both general and advanced users. While any users can build their search engines using the default step-by-step approach, advanced users can modify existing components to meet their own needs.

In general, the design of SpidersRUs resembles the architecture presented in Section 2.2 and a design diagram is shown in Fig. 8. The detailed architecture of SpidersRUs was presented in [8,9]. In this paper, we focus more on areas such as the user interface, that are relevant to using the system from a user perspective.

The system is comprised of four components, three of which correspond to the three main functions of the tool, while a graphical user interface (GUI) is used to access these functions. In the following, we will discuss each component in details as well as the unique features offered by the components while we go through the process of developing a search engine.

4.1. Spidering

To start with, the users have to specify a list of topic-specific URLs known as seeds, which are then assigned to the spiders. Each spider is initially assigned exactly one URL from the seeds and tries to fetch the document specified by the URL. To maximize efficiency, parallel fetching of documents is made possible by implementing each spider as a single thread. A component called ContentHandler is used to check if there are any duplicated documents collected (e.g., due to mirror sites). This is implemented by maintaining a hash table called ContentSeen. In addition, in order to improve the relevance of search results, irrelevant documents can be filtered out from the collection at same time. This function is handled by the ContentFilter, which checks to see if the documents contain some key words that the users have included in the bad-term list. Each document in the filtered collection is assigned an identifier (id) in the Item Index, which keeps the list of all documents and their corresponding ids. This completes the first pass of spidering.

In order to build a larger collection, any HTML documents, which most commonly contain URLs pointing to other documents of the same topic, are passed to the HTMLParser. After receiving each HTML document, the HTMLParser extracts all the URLs inside and passes them to the URLHandler, which decides whether to add the target document to the collection by considering a number of factors.

Firstly, the URLHandler maintains a hash table called URLSeen, which is similar to the ContentSeen described above. If a URL is found

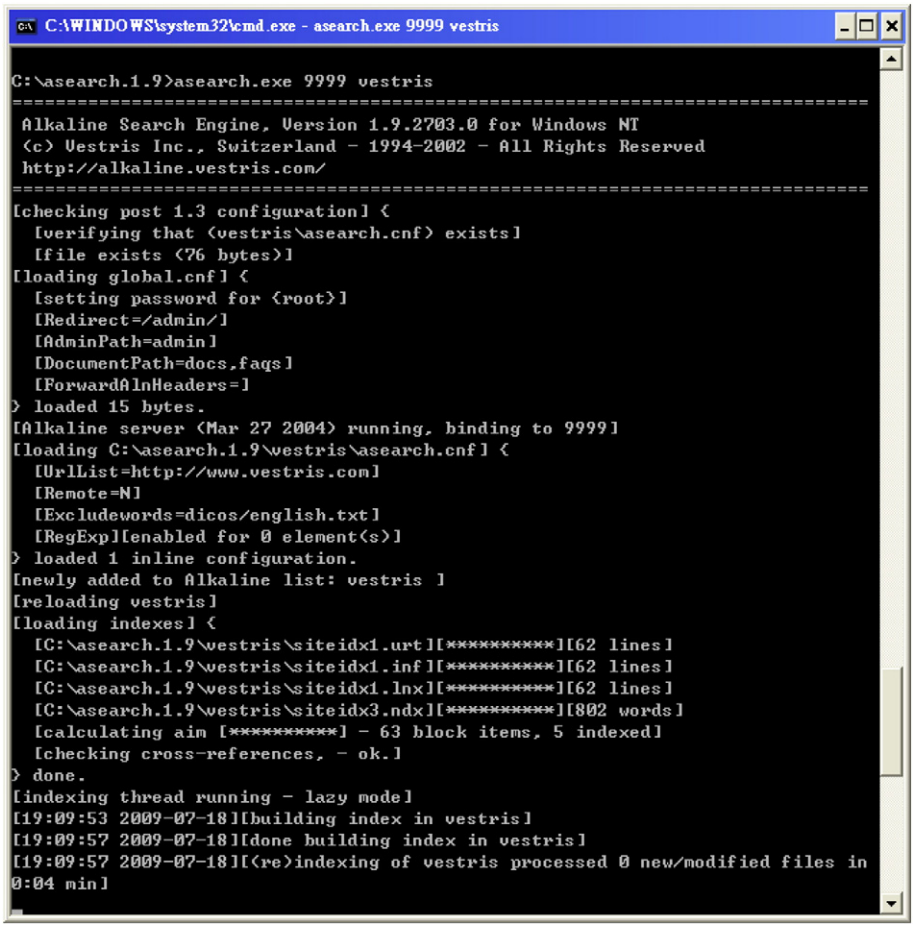

Fig. 6. Alkaline – starting search service.

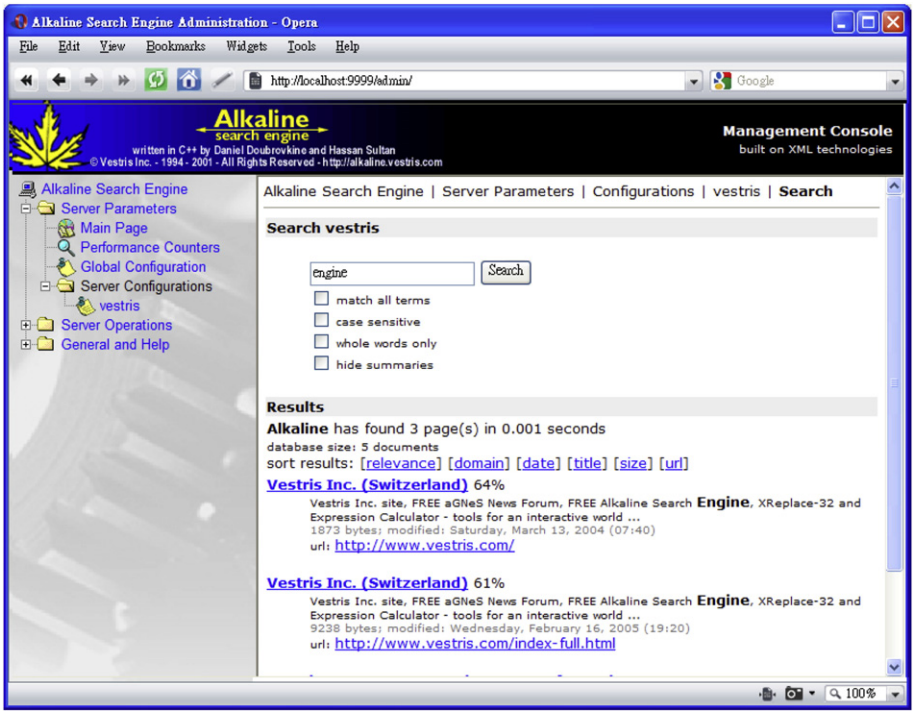

Fig. 7. Alkaline – search results.

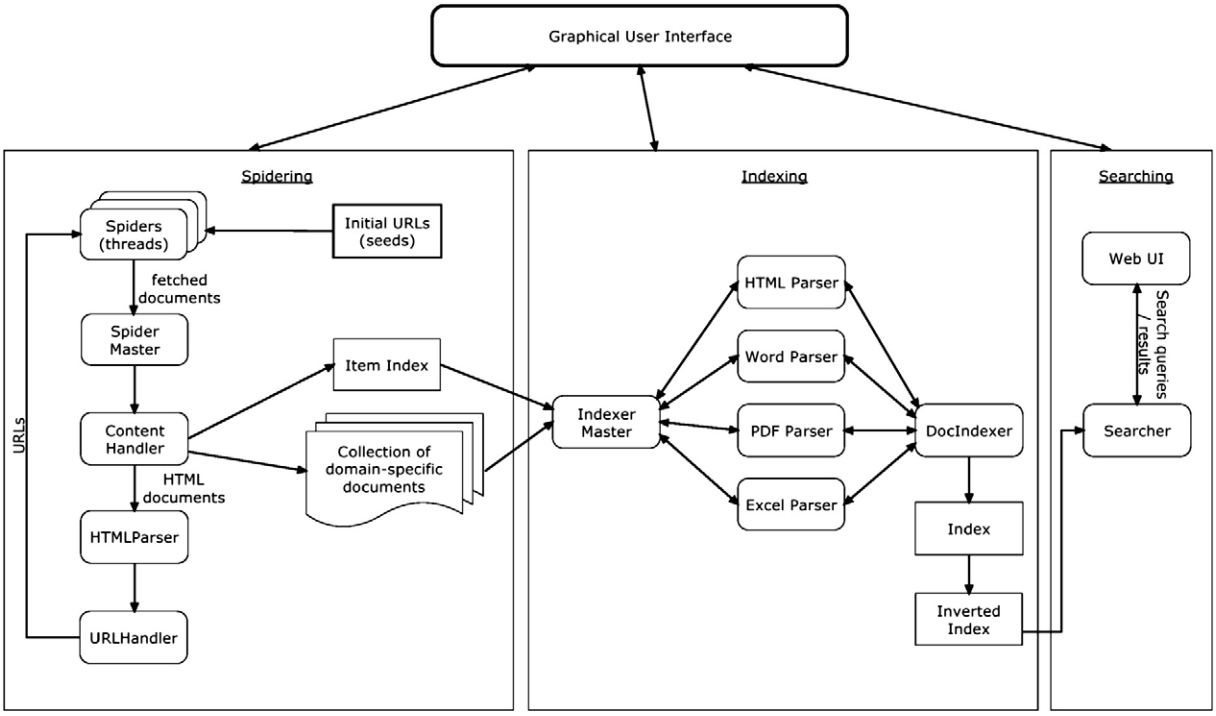

Fig. 8. Overall architecture of SpidersRUs.

in the URLSeen, it suggests that the URL has been accessed before and should thus be ignored. Otherwise, the URL is passed to one of the spiders for fetching, and it is also added to the URLSeen to avoid duplicate fetching. One point worth mentioning is that a URL of a particular Web domain will always be passed to the spider that has been fetching from that domain. This restriction ensures that no two spiders can access the same Web server at the same time, which avoids overloading the server.

Secondly, a component called URLFilter is used to filter out "bad URLs". In order for this function to work, users can specify a list of unwanted URLs using regular expression. Any URL handled by the URLHandler that matches the unwanted list will be ignored.

Thirdly, the Robot Exclusion Protocol is used to determine if the Web server welcomes robots to index its documents. In practical terms, if the file "robots.txt" is found on a Web server, the URLs listed in the file will be ignored by the URLHandler.

Subsequently, any new URLs from the URLHandler will be added to the URL queue and fetched by the spiders. The document collection as well as the Item Index will grow in size. The process can be terminated based on a number of factors. For instance, users can set a limit on the total number of documents to be fetched.

4.2. Indexing

The Item Index and the collection of documents obtained after spidering are passed to the IndexerMaster. The IndexerMaster obtains information about each document in the collection and decides if it is an HTML, Word, PDF or Excel document. The document is then passed to the respective document parser (HTML Parser, PDF Parser, Word Parser, or Excel Parser), where it is converted to plain text for easier indexing.

After that, an Initial Index is created by the DocIndexer. The Initial Index is essentially a list of <document, word, frequency of occurrences> tuples. As multilingual support is a highlighted feature of SpidersRUs, the decision for defining a word is crucial at this step. Apparently English words are sequences of characters separated by spaces, but when it comes to some Asian languages such as Chinese and Japanese, which do not have spaces, the definition of a word can become controversial. While one can define a word as a sequence of characters separated by punctuations, each "word" will be excessively long and impossible to be matched by end-users' search queries. Here we adopt a slightly different definition between space-delimited (most European languages including English) and non-space-delimited (most Asian languages) languages. For the former, a word is separated by either punctuation or a space; for the latter, each character is taken as a word. With these measures taken, both types of languages can be indexed effectively.

The next step following the creation of the initial Index is to build the Inverted Index, as mentioned in Section 2.2. Technically the Inverted Index contains the exact same list of words as the initial Index. To create an Inverted Index, we should first traverse through the Index to get the list of words (with duplicates removed). Then for each word in the list we need to find the corresponding documents and frequency. In other words, the Inverted Index is a list of <word, list of documents, frequency> tuples for the entire document collection.

4.3. Searching

The Searcher is the component that allows users to perform searching through a Web interface. Besides focusing on searching efficiency and accuracy, it is very important for the Searcher to have a user-friendly and easy-to-use Web interface for getting queries from and displaying search results to the end-users. In SpidersRUs, users can specify customized HTML files to be used as templates for the search interface and for result display.

4.4. User interface

4.4.1. Creating a new project

When the tool first starts, the users have to create a new project. A new project may be created as "New" or "Advanced New". While the first option appeals to general users with a simple interface (see Fig. 9), the "Advanced New" option offers more options for advanced users to exert a higher degree of customization (see Fig. 10). In particular, the "Advanced New" option allows users to specify the

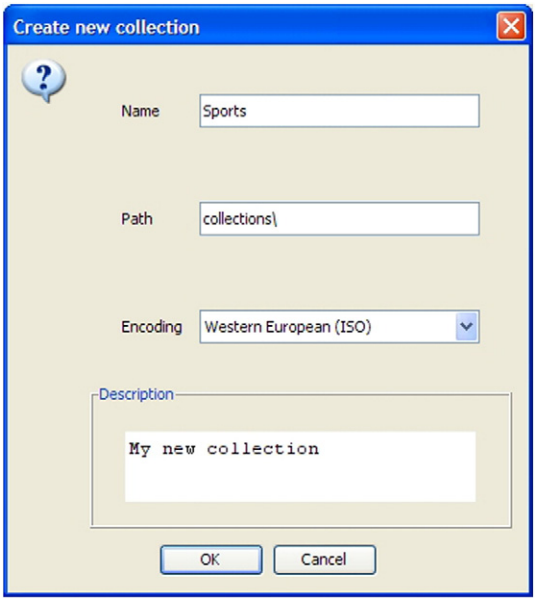

Fig. 9. Creating a "New" project.

location of each of the components of the tool, which include IndexerLoader, SpiderMaster, IndexerMaster, Searcher and so on.

Any newly created projects are added to the projects list on the left of the main window. This feature allows users to work on multiple projects at the same time. As shown in Fig. 11, a list of parameters such as "Items collected" and "Search index size" can be seen on the right in the "Collection" tab after selecting a project on the left.

4.4.2. Spidering

In the Spidering tab (Fig. 12), users can press the "Add Seeds" button to specify the seeds for spidering. They may choose to insert the URLs one by one, or to import them from a text file.

As shown in Fig. 13, the "Advanced" option allows users to enter some spider parameters such as "no. of spiders" and "timeout".

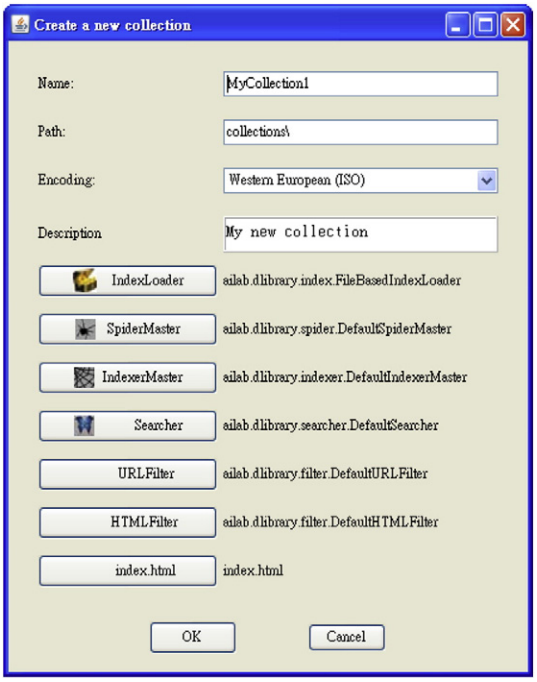

Fig. 10. Creating an "Advanced New" project.

The spidering process can then be started. We recognized that it is important that users be informed about the progress of their collection. During the process, a progress bar and the number displays (e.g., number of pages fetched) will be continuously updated. Detailed messages can also be seen from the text box at the bottom (see Fig. 14). These messages are useful for debugging purpose for advanced users.

4.4.3. Indexing

After the documents have been fetched, users can start the indexing process to create the initial Index and then the Inverted Index. These steps are carried out under the Indexing tab (Fig. 15).

4.4.4. Search service

After the vertical search engine has been built, users can start the search engine server by clicking on the "start service" button (Fig. 16). The default port of 9999 will be used, but it can be changed by the users if needed. The user interface of the search engine can be shown by launching a Web browser (Fig. 17).

5. Evaluation

In order to test the usability of SpidersRUs in facilitating users in search engine development, we conducted a user study to compare the usability and user satisfaction of SpidersRUs with Greenstone and Alkaline as benchmarks. We discuss the study in detail in this section.

5.1. Overview of the study

The evaluation study was conducted in an undergraduate business course called Internet Applications Development, which was taught at the University of Hong Kong (HKU). Participants were asked to build topic-specific search engines in the course as part of their coursework. Each group (with three students) was assigned a specific topic (e.g., basketball) and they had to create search engines based on that topic. In order to compare among the three tools mentioned above, each group had to use all the three tools. In other words, each group would produce three search engines on the same topic.

Although most participants did not have any experience in building search engines, they were given only 4 weeks to finish their work as a group. The time constraint was set so that we could evaluate the level of difficulty in learning to use the tools.

After the participants finished the project, each of them completed a questionnaire. Details of the questionnaire are discussed in the following sections.

5.2. The Questionnaire for User Interaction Satisfaction (QUIS)

Our questionnaire was designed based on a usability testing tool called Questionnaire for User Interaction Satisfaction (QUIS). Developed by the Human–Computer Interaction Lab at the University of Maryland at College Park, QUIS is a measurement tool for evaluating computer users' subjective satisfaction with the human–computer interface [14]. It has been suggested that QUIS is widely applicable for accessing many different types of interfaces [16].

While a lot of other measurement tools also serve a similar purpose, they usually suffer from problems such as validation and reliability [17]. It has also been suggested that the selection of question types is also an important factor to make a questionnaire effective in revealing users' opinion towards usability of software systems [14].

QUIS was designed as a result to provide reliable measurement methods. There had been a number of versions of QUIS, with each version adding more evaluation criteria to the previous one. Most QUIS-based questionnaires are arranged in a hierarchical layout — it starts with a demographic questionnaire, which aims to determine

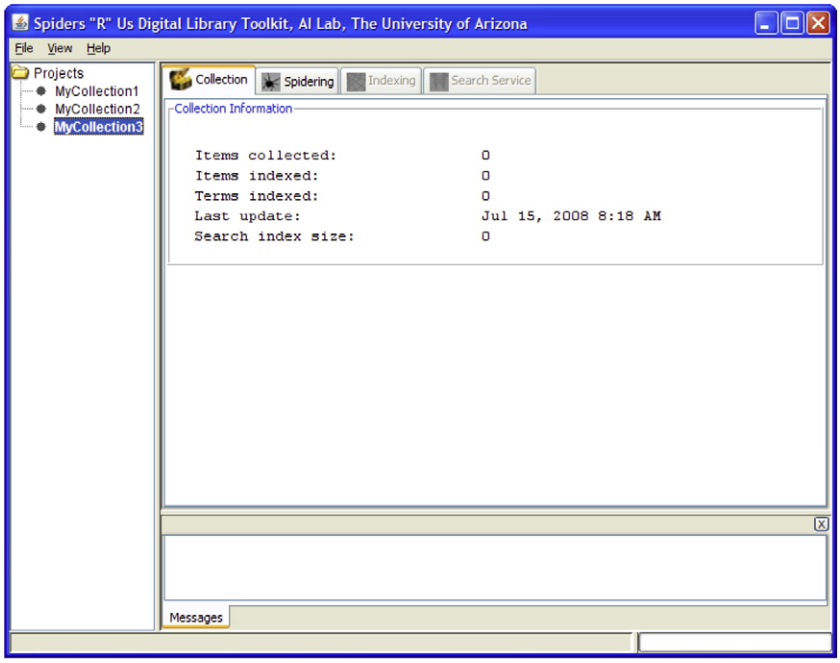

Fig. 11. Working with multiple projects.

user background information such as level of computer literacy. This is followed by measures of overall reaction towards the system. Finally, there are several specific interface sections.

We adapted our questionnaire from QUIS and it consists of:

(1) A background information section with questions relating to experience of using computers and building search engines;
(2) An overall-reaction section with six measures (e.g., level of difficulty and satisfaction of using each of the search engine building tools);
(3) Four standard measures on screen layout and sequence, terminology and system information, learning to use the system and system capabilities; and
(4) A comments section containing open-ended questions, which allow subjects to provide comments that are possibly related to areas of improvement of each tool desired by the subjects.

In parts (2) and (3) above, subjects were asked to give their responses to all of the three tools for each question being asked (i.e., three responses to each question) so that we could easily compare

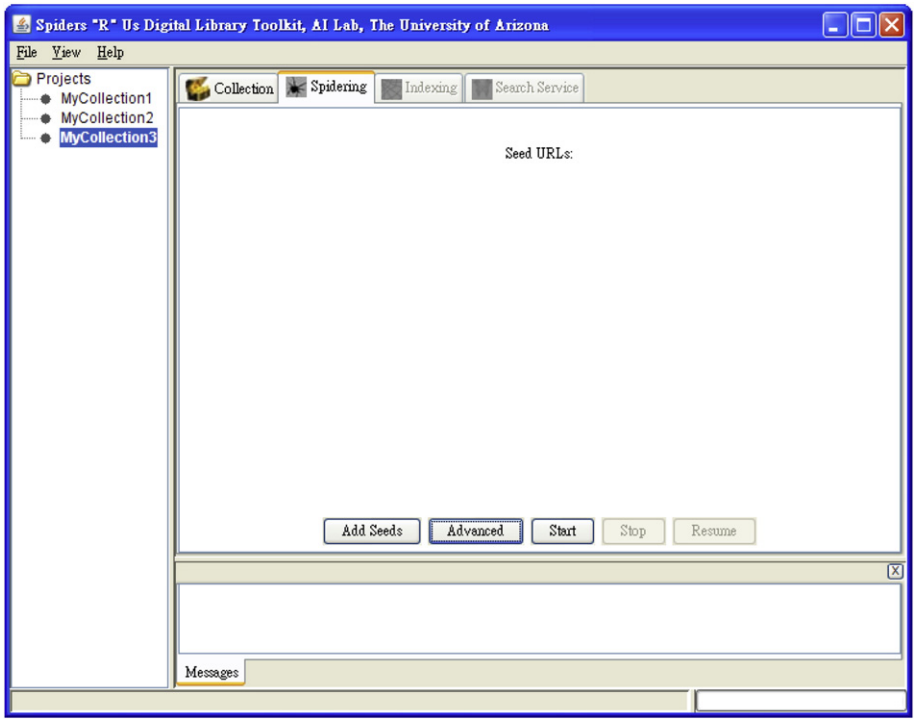

Fig. 12. Adding seed URLs.

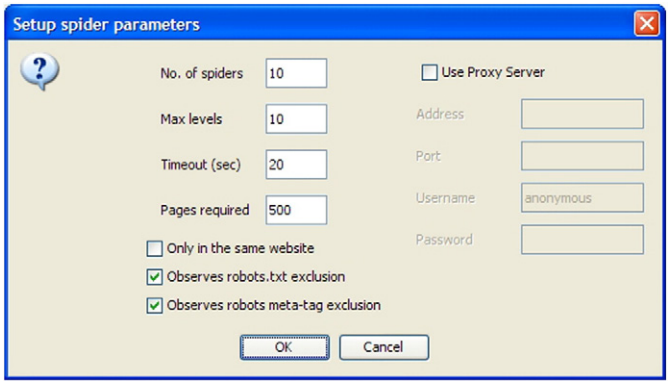

Fig. 13. Advanced spider parameters.

among the different tools. For these three sections, the response to each item was rated on a 10-point scale (with a positive adjective on one side of the scale and a negative one on the other side). In a question about "use of terms throughout system", for example, answering 0 meant the use of terms in the tool was "mostly inconsistent" and 9 "mostly consistent".

5.3. Results of the study

A total of 28 sets of responses were received from the 33 subjects, yielding a response rate of 84.85%. In the following sections, we analyzed our results by presenting the mean scores and *t*-test pair-wise comparisons between the means. As mentioned earlier, each response was given on a 0–9 scale (9 being the most favorable). Also note that the sample size for all of the results was 28.

5.3.1. Overall reactions to the tools

The participants' responses regarding their overall reactions to the tool are summarized in Table 1. In terms of overall reactions to each of the three tools, SpidersRUs scored the highest in all of the six areas specified in the questionnaire, which suggests that it was preferred over the other two. Besides, Alkaline was also given favorable scores, particularly in the area of flexibility, where its mean score was very close to that of SpidersRUs. Surprisingly Greenstone had the lowest scores among the three tools, even though it was designed for computer users with different levels of computer skills.

We conducted a series of *t*-test and the results showed that the mean scores for SpidersRUs are significantly better than that for Greenstone, especially in terms of overall impression, user friendliness and being interesting, for which the *p*-values are smaller than 0.0001. On the other hand, the differences are smaller between Alkaline and SpidersRUs. The difference is only significant in two items (being interesting and ease of use).

5.3.2. Four QUIS standard measures

The four QUIS standard measures we included in the questionnaire were screen layout and sequence, terminology and system information, learning to use the system, and system capabilities. The results of these measures are shown in Table 2.

In terms of screen layout and sequence, the mean score of SpidersRUs is higher than that of Alkaline, which in turn is higher than that of Greenstone. A pair-wise *t*-test further confirmed that SpidersRUs has the highest scores for all four items as well as the average score in this area. The high scores of SpidersRUs in screen layout can be partly explained by the clear interface layout, which starters would find easier to control without having to consult the documentation. For example, as shown in Fig. 11, SpidersRUs has a collection menu (the panel on the left) which lists all projects that the user is currently working on, whereas Greenstone can have only one project opened at a time, which means

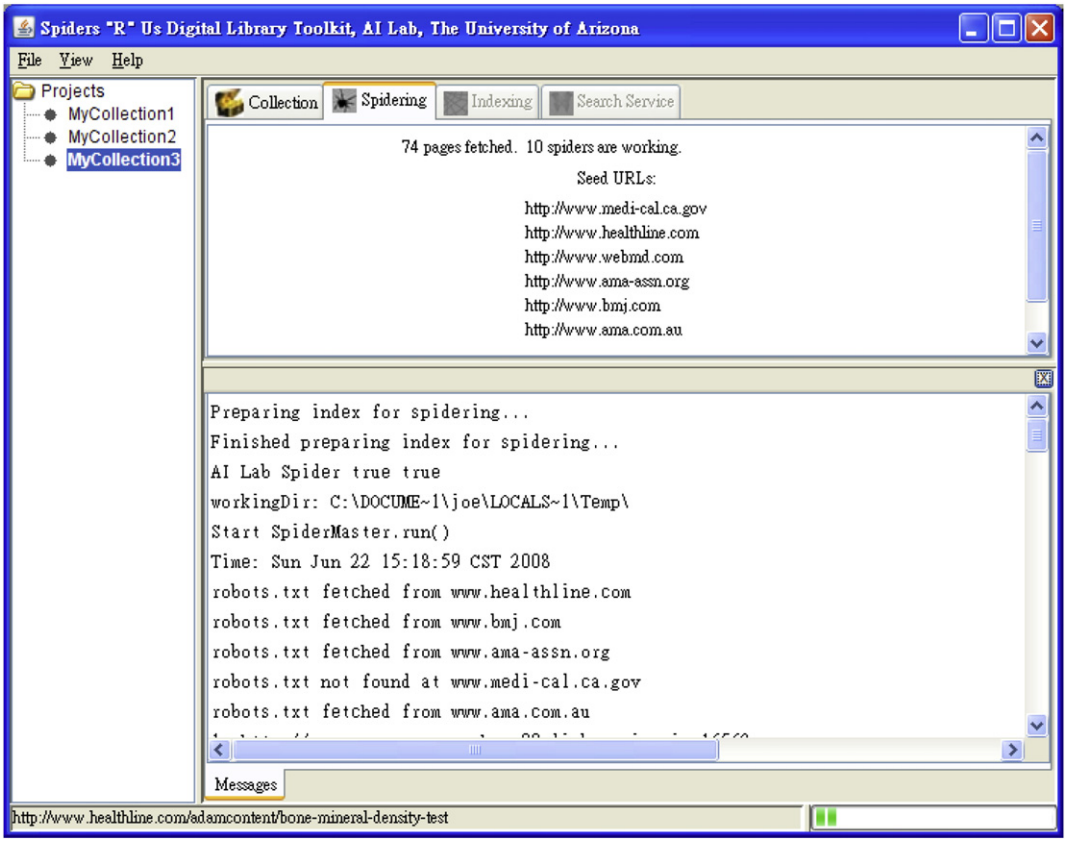

Fig. 14. Spidering in progress.

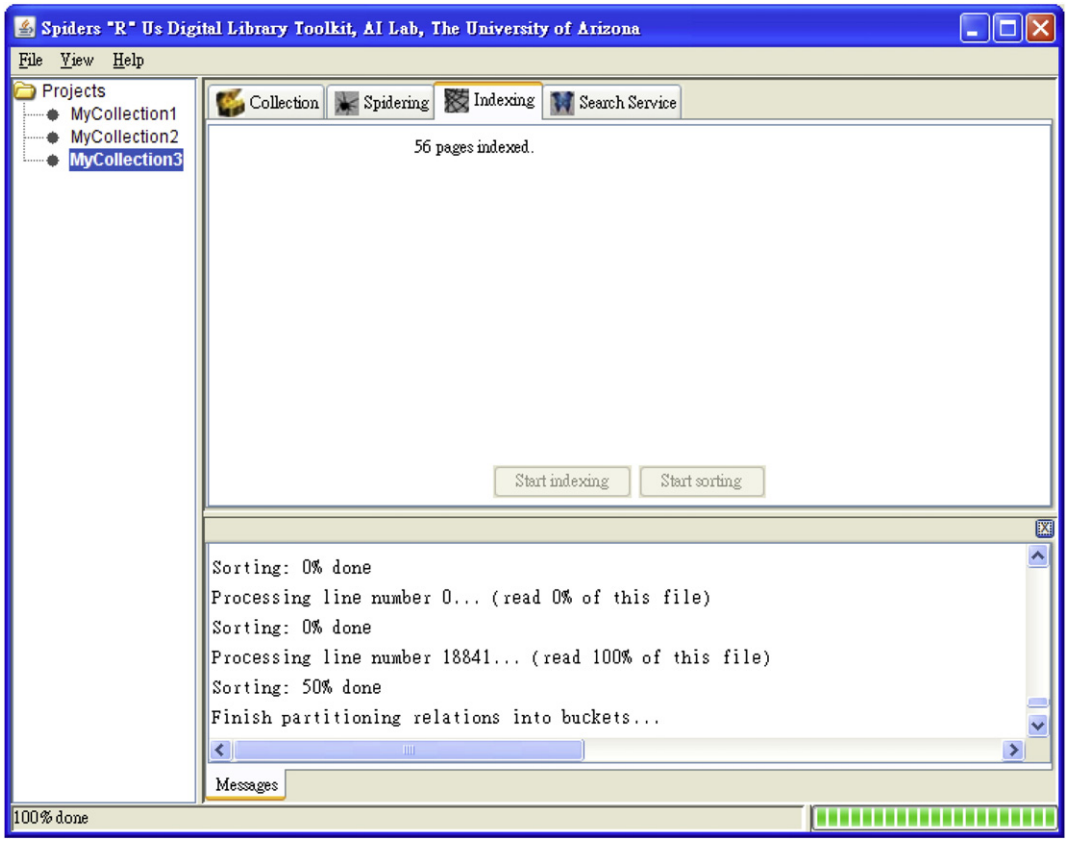

Fig. 15. Indexing.

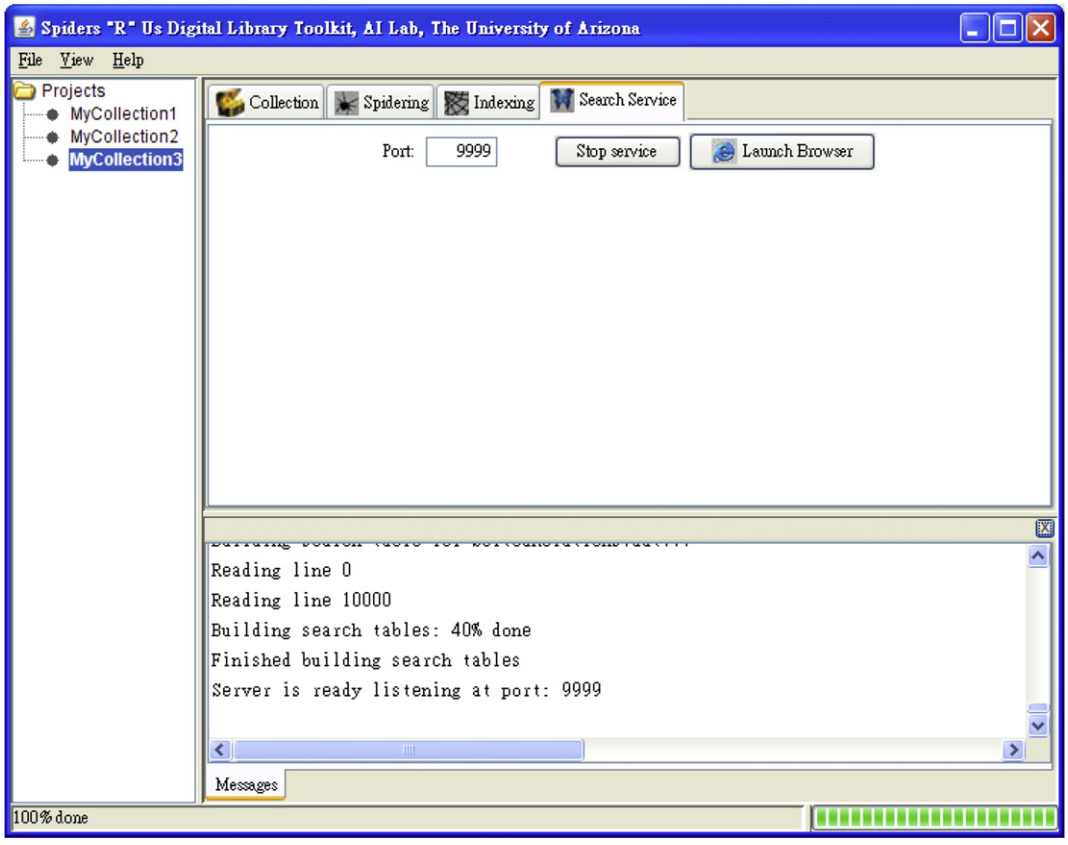

Fig. 16. Search service.

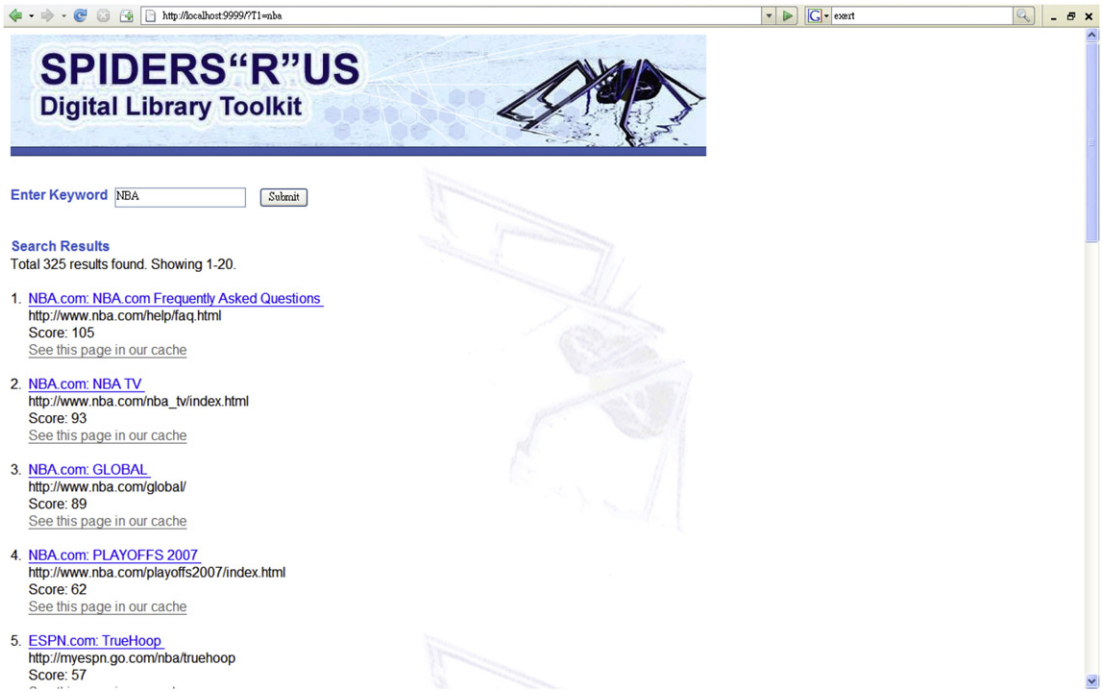

Fig. 17. Sample search engine results page in a Web browser.

in order to switch from a project to another, users have to save the current project before opening another one.

In the area of terminology and system information, SpidersRUs' scores are again the highest in every question that we included, and the differences between the two pairs (Alkaline/SpidersRUs and Greenstone/SpidersRUs) are also significant, as demonstrated by the small *p*-values. In addition, while Alkaline scored generally higher than Greenstone, the former obtained a lower score in messages on screen which prompt the user for input.

We suggest that the main reason for SpidersRUs' high scores in this particular area is again its user interface design. For example, as shown in Fig. 14, any information or error messages are always shown in the bottom panel in the same frame, so users can easily keep track of the spidering process. Also, the tool always makes the screen simpler by hiding unnecessary information. When users create a new collection, for example, they can choose either "New" or "Advanced New", where the former hides all the technical options which beginners may not understand. This allows them to work on the tool without consulting a lot of documentation.

In the area of learning to use the system, the same result was observed. SpidersRUs has the highest mean score in every question. On the contrary, Greenstone has the lowest scores, except in remembering names and use of commands, for which Greenstone scored higher than Alkaline. In addition, the *p*-values are generally very small (most of them are less than 0.01), which shows the statistical significance of the high mean values of SpidersRUs.

Table 1
Overall reactions to the tools (ALK: Akaline, GRN: Greenstone, SPD: SpidersRUs).

Item	ALK	GRN	SPD	ALK vs. SPD (*t*-test *p*-value)	GRN vs. SPD (*t*-test *p*-value)
Overall impression	5.75	2.32	6.07	0.3753	<0.0001
User friendliness	5.50	2.43	6.00	0.2700	<0.0001
Being interesting	5.07	2.68	5.96	0.0339	<0.0001
Ease of use	5.14	4.25	6.52	0.0115	0.0014
Powerfulness	5.74	4.39	6.22	0.1993	0.0002
Flexibility	5.78	3.82	5.89	0.8342	0.0006
Average	5.50	3.32	6.11	0.0942	<0.0001

We believe that a major explanation of the high scores of SpidersRUs is the design to allow beginners to use the simpler version of the user interface, which contains less technical information. In addition, each step of the search engine development process is shown on a different tab in the main interface: Collection, Spidering, Indexing, and Search Service. This step-by-step approach is easier to follow, even for beginners.

The last part of the QUIS measures was system capabilities, in which SpidersRUs again scored the highest in every question, except in system reliability. While the differences between SpidersRUs and Greenstone are significant for all items, it is not the case for the comparison between Alkaline and SpidersRUs. In particular, the *p*-values are larger than 0.05 for four items, showing that SpidersRUs' scores are not significantly higher than those of Alkaline.

One barrier that hindered SpidersRUs from getting a significantly higher score was that the tool was designed to print out error messages when it encountered program exceptions. For example, in the spidering process, the tool has to fetch a number of Web pages from the Internet. In case any of these pages fail to load due to network problems (e.g., server not responding, traffic congestion), some exception errors (e.g., UnknownHostException) are directly displayed in the message window without further explanations. While users who know about these messages would understand the tool is still working fine, others could consider these messages a sign of malfunction of the tool.

5.3.3. Qualitative comments

We included several open-ended questions asking for areas of potential improvements in each of the three tools. For Alkaline, which only had a command-based interface, most participants suggested that a graphical user interface should be used to make the tool more user-friendly. There were also a few comments on the speed of the tool. For example, a participant complained that the tool indexing was too slow and another suggested that better multi-threading implementation could help improve its performance.

For Greenstone, the majority of participants complained about the slow spidering process, as too much data was fetched from a single seed. Some participants suggested the tool include an option that allows users to specify the amount and types of data to be fetched from a single site.

Table 2
Four QUIS standard measures.

Item	ALK	GRN	SPD	ALK vs. SPD (*t*-test *p*-value)	GRN vs. SPD (*t*-test *p*-value)
Screen layout and sequence					
Characters on the computer screen	5.50	4.50	6.79	0.0193	<0.0001
Highlighting on the screen simplifies task	4.86	4.75	6.36	0.0202	0.0036
Organization of information on screen	5.36	4.93	6.52	0.0216	0.0020
Sequence of screens	5.50	5.11	6.75	0.0177	0.0012
Average	5.30	4.82	6.60	0.0117	0.0002
Terminology and system information					
Use of terms throughout system	5.56	5.11	6.48	0.0159	0.0064
Terminology on screen is related to the task you are doing	5.32	4.57	6.54	0.0032	0.0003
Position of messages on screen	5.57	4.93	6.54	0.0295	0.0007
Messages on screen which prompt user for input	5.04	5.21	6.64	0.0013	0.0015
Computer keeps you informed about what it is doing	5.50	5.25	6.54	0.0080	0.0065
Error messages	4.89	4.61	6.04	0.0082	0.0039
Average	5.31	4.95	6.46	0.0010	0.0003
Learning to use the system					
Learning to operate the system	5.14	4.79	6.75	0.0018	0.0004
Exploring new features by trial and error	5.21	4.39	6.43	0.0090	0.0001
Remembering names and use of commands	5.07	5.54	6.68	0.0010	0.0118
Tasks can be performed in a straight-forward manner	5.18	5.14	6.79	0.0016	0.0018
Help messages on the screen	5.36	5.29	6.57	0.0021	0.0072
Supplemental reference materials	5.25	4.96	6.32	0.0311	0.0192
Average	5.20	5.02	6.59	0.0006	0.0005
System capabilities					
System speed	6.33	2.25	6.41	0.8584	<0.0001
System reliability	6.30	3.11	6.22	0.8700	<0.0001
System tends to be noisy/quiet	5.63	4.18	6.27	0.0894	0.0002
Correcting your mistakes	5.04	3.46	5.46	0.3275	0.0007
Experienced and inexperienced users' needs are taken into consideration	4.96	4.59	6.42	0.0017	0.0010
Average	5.65	3.52	6.16	0.1333	<0.0001

Lastly for SpidersRUs, comments were generally about adding more features to the tool. In the spidering process, for example, a participant would like to see the option to delete a seed from the list. A few participants also suggested adding more configurable options so they had more control during the process.

6. Guidelines for graphical user interface design

The general idea to make a software tool usable is to provide an intuitive and self-explanatory user interface. In other words, users should not find it necessary to consult manuals before they can complete simple tasks provided by the tool. By considering the designs of Greenstone and SpidersRUs and the evaluation results, this section lists some basic design guidelines to improve the usability of such tools. It also explains why SpidersRUs is a preferred tool in terms of usability. We focus our discussion here on SpidersRUs and Greenstone since Alkaline does not provide a graphical user interface during the search engine development process.

6.1. Focus each screen on related tasks only

By comparing the three tools in the user study, we can give a few suggestions on how these tools can be improved. Firstly, offering too many functions in a tool may negatively affect its usability. Greenstone is a tool with many advanced options. Though Greenstone offers more options than SpidersRUs in some aspects, such as indexing downloaded files, it obtained lower scores on average on all the QUIS measures. The problem is that putting too many options on the same screen can confuse users who do not find them necessary. We suggest that in order to avoid confusion, all the features and options on each screen of the user interface should be related to the same task. For example, in Greenstone, after creating a new collection, users would have to provide a list of seed URLs (see Fig. 3). The page contains text fields for entering the URLs, and also some descriptions on how files are downloaded from FTP and HTTP servers.

SpidersRUs makes use of tabs to divide the various available features into groups, where each group contains features related to the same task. Under the “Spidering” tab (Fig. 12), for example, there are the necessary functions such as “Add Seeds” and “Start”. There is also the “Advanced” button which lets users specify more detailed spider parameters.

6.2. Make it easy to handle multiple collections

Users often need to create multiple collections. It is desirable for users to manage all these collections easily. In SpidersRUs, the list of collections is shown on the left (Fig. 11). Users can switch to any collection by clicking on it, which means they do not have to leave the current collection. In Greenstone, however, users need to exit from any opened collection in order to go back to the first step (see Fig. 2) and then open an existing collection.

6.3. Hide advanced options

While it is desirable to provide more advanced options in a tool, showing too many options on the same screen can easily confuse users. Consider the two screenshots for SpidersRUs shown in Figs. 12 and 13. The advanced options are not shown in the main page. They are accessible using the “Advanced” button. This prevents general users from getting frustrated by the options they may not find necessary.

6.4. Make the sequence of required steps clear

If there is a pre-condition for a particular task, the tool should make it clear to users what the pre-condition is and how it can be achieved. Tools involving that kind of requirements often employ the

use of wizards to make the order of each step clear. For example, when creating a chart, spreadsheet tools often use wizards to guide users through the different steps such as selection of chart design and data source.

Both Greenstone and SpidersRUs try to guide users through the different steps in developing a search engine. However, there is a minor difference which makes SpidersRUs more user-friendly. In Greenstone, most of the buttons are available right at the beginning (Fig. 3), which could be confusing because users would not know what the next step should be. In addition, users can even select "Build Collection" when there are no files in a collection, which would merely create an empty collection. On the other hand, in SpidersRUs the "Indexing" and "Search Service" tabs are both disabled when the "Spidering" step has not been completed (Fig. 11). This will show users more clearly where they are in the process.

7. Conclusions and future directions

In this paper, we reviewed three different tools for building topic-specific search engines and discussed their unique features which made them useful for different users. We then tested the usability of each tool using a survey based on the QUIS. As we have observed from the results of the survey, SpidersRUs outperformed the others with statistical significance in most areas that we covered in the questionnaire. In particular, SpidersRUs was highly favored for its user interface design, which made it significantly better than the other two tools in such areas as screen layout and sequence, terminology and system information, and learning to use the system. On the other hand, there were some areas in which SpidersRUs did not stand out as much, particularly in system capability.

While our study was conducted on university undergraduate students, it would be worthwhile to find out how other types of users compare the three tools. A group of programmers, for example, may find command-driven tools more convenient and thus may not favor a graphical interface. A class of high school students, on the other hand, may find that understanding how Web spiders work is challenging and none of the tools would stand out from their point of view. In that case, a similar system that provides, for example, a step-by-step guide of building search engines may easily stand out even if it does not provide as many other features as the three tools we have considered.

Future work will be conducted in several directions. First, we are planning to further improve the user interface of our tool based on the issues identified. Moreover, it would be interesting to compare the tools for other user groups. Lastly, we explore the possibility of extending the tool to work with Web 2.0 contents such as blogs.

Acknowledgments

This project has been supported in part by a Seed Funding for Basic Research grant from the University of Hong Kong. We thank Hsinchun Chen, Yilu Zhou, Jialun Qin, Chunju Tseng, Chiayung Hsu, William Lau, and all the participants for their contribution to this project.

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] D. Bainbridge, I.H. Witten, Greenstone digital library software: current research, Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries, Tuscon, AZ, USA, 2004, p. 416-416.
[3] C.M. Bowman, P.B. Danzig, U. Manber, F. Schwartz, Scalable Internet resource discovery: research problems and approaches, Communications of the ACM, August 37 (8) (1994) 98–107.
[4] S. Chakrabarti, M. van den Berg, B. Dom, Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery, Proceedings of the 8th International World Wide Web Conference, Toronto, Canada, 1999, May 1999.
[5] M. Chau, H. Chen, in: N. Zhong, J. Liu, Y. Yao (Eds.), Web Intelligence, Springer-Verlag, February 2003, pp. 197–217.
[6] M. Chau, H. Chen, Comparison of three vertical search spiders, IEEE Computer 36 (5) (2003) 56–62.
[7] M. Chau, H. Chen, J. Qin, Y. Zhou, Y. Qin, W.K. Sung, D. McDonald, Comparison of two approaches to building a vertical search tool: a case study in the nanotechnology domain, Proceedings of The Second ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'02), Portland, Oregon, USA, July 14–18, 2002, pp. 135–144.
[8] M. Chau, J. Qin, Y. Zhou, C. Tseng, H. Chen, SpidersRUs: automated development of vertical search engines in different domains and languages, Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05), Denver, Colorado, USA, June 7–11, 2005, pp. 110–111.
[9] M. Chau, J. Qin, Y. Zhou, C. Tseng, H. Chen, SpidersRUs: creating specialized search engines in multiple languages, Decision Support Systems 45 (3) (2008) 621–640.
[10] H. Chen, M. Chau, D. Zeng, CI Spider: a tool for competitive intelligence on the Web, Decision Support Systems (DSS) 34 (1) (2002) 1–17.
[11] H. Chen, H. Fan, M. Chau, D. Zeng, Testing a cancer meta spider, International Journal of Human-Computer Studies 59 (5) (2003) 755–776.
[12] H. Chen, A.M. 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.
[13] F.C. Cheong, Internet agents: spiders, wanderers, brokers, and bots, New Riders Publishing, Indianapolis, Indiana, USA, 1996.
[14] J. Chin, V. Diehl, K. Norman, Development of an instrument measuring user satisfaction of the human–computer interface, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Washington, D.C., USA, 1988, pp. 213–218.
[15] D. Czarnecki, A. Deitsch, Java Internationalization, O'Reilly & Associates, Sebastopol, California, USA, 2001.
[16] B.D. Harper, K.L. Norman, Improving user satisfaction: the questionnaire for user interaction satisfaction version 5.5, Proceedings of the 1st Annual Mid-Atlantic Human Factors Conference, 1993, pp. 224–228.
[17] B. Ives, M.H. Olson, J.J. Baroudi, The measurement of user information satisfaction, Communications of the ACM 26 (10) (1983) 785–793.
[18] S. Lawrence, C.L. Giles, Inquirus, the NECI meta search engine, Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, 1998, pp. 95–105.
[19] F. Menczer, Complementing search engines with online Web mining agents, Decision Support Systems 35 (2003) 195–212.
[20] G. Pant, P. Srinivasan, Learning to crawl: comparing classification schemes, ACM Transactions on Information Systems, Oct 2005.
[21] G. Salton, Developments in Automatic Text Retrieval. 1991, Science, vol. 253. no. 5023, 1991, pp. 974–980.
[22] L. Sheble, 2006. Greenstone User Survey: Technical Report. School of Library and Information Science, University of North Carolina, June 2004, available at: http://www.ils.unc.edu/~sheble/greenstone/survey-report.html.
[23] 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.
[24] I.H. Witten, D. Bainbridge, S.J. Boddie, Greenstone: open-source DL software, Communications of the ACM 44 (5) (2001) 47.
[25] I.H. Witten, D. Bainbridge, A retrospective look at Greenstone: lessons from the first decade, Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries, Vancouver, BC, Canada, 2007, pp. 147–156.
[26] A.B. Zhang, I.H. Witten, T.A. Olson, L. Sheble, Greenstone in practice: implementations of an open source digital library system, Proceedings of American Society for Information Science and Technology Annual Meeting, Charlotte, North Carolina, USA, October, 2005, pp. 769–794.

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, and security informatics. He has published more than 70 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/.

Cho Hung Wong is a Research Assistant in the School of Business at the University of Hong Kong. He received his Bachelor of Business Administration degree in information systems from the University of Hong Kong and his Master of Science degree in computing science from Imperial College London. His research interests include search engine, data mining, and human computer interaction.