

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

UMI<sup>®</sup>



# BIB<sub>T</sub><sub>E</sub>X Server

**Man Bao**

A Major Report  
in  
Department  
of  
Computer Science

Presented in Partial Fulfillment of the Requirements  
For the Degree of Master of Computer Science  
Concordia University  
Montreal, Quebec, Canada

April 2002

© Man Bao, 2002



**National Library  
of Canada**

**Acquisitions and  
Bibliographic Services**

**395 Wellington Street  
Ottawa ON K1A 0N4  
Canada**

**Bibliothèque nationale  
du Canada**

**Acquisitions et  
services bibliographiques**

**395, rue Wellington  
Ottawa ON K1A 0N4  
Canada**

**Your file / Votre référence**

**Our file / Notre référence**

**The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.**

**The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.**

**L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.**

**L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.**

0-612-72927-3

# **Abstract**

## **BIB<sub>T</sub>E<sub>X</sub> Server**

Man Bao

BIB<sub>T</sub>E<sub>X</sub> is a program that creates correctly formatted citations. It is widely used in the scientific community. A bib database is needed for BIB<sub>T</sub>E<sub>X</sub> to construct a citation. The bib database traditionally consists of files with a bib extension containing bibliographic entries. In recent years, an XML representation of BIB<sub>T</sub>E<sub>X</sub> data has been reported. When the bib database gets large, an efficient search mechanism for bibliographic information is needed. Two kinds of search systems have been developed, command line based and web based. The former doesn't tend to be easy-to-use, and the most web based search systems use CGI technology that doesn't meet the requirement of search efficiency. In this project, we developed a web based search system using Java servlet technology. An XML presentation of BIB<sub>T</sub>E<sub>X</sub> data is used, and the original data is kept in the format of BIB<sub>T</sub>E<sub>X</sub> which gives the database administration more flexibility. The search efficiency goal is achieved by using the JDOM API to manipulate the XML presentation of data. General search and advanced search are provided with a web interface. The search result can be saved to local disk or sent by email. Furthermore, we present methods for adding BIB<sub>T</sub>E<sub>X</sub> data entries through the Internet or manually to the bib database in a secure mechanism.

## **Acknowledgement**

I'd like to take this opportunity to express my appreciation to my supervisor, Professor Peter Grogono, for his patience and constant guidance.

I am grateful to the professors and staff in Computer Sciences department at Concordia for the wonderful courses and services. I would especially like to say thanks to Ms. Halina Monkiewicz, who was always friendly and prompt in providing assistance.

# Table of Contents

1. Introduction .....	1
1.1 The background of BiBT <sub>E</sub> X .....	1
1.1.1 The content of bib file .....	1
1.1.2 The format of bib file .....	3
1.2 BiBT <sub>E</sub> X database and query .....	4
1.3 The goal of this project .....	6
2 Design .....	8
2.1 Architecture design .....	8
2.1.1 Design rationale .....	8
2.1.2 System design .....	15
2.1.3 System tools and environment .....	23
2.2 Module interface design .....	24
3 Implementation .....	29
3.1 System environment .....	29
3.2 Function implementation .....	29
3.2.1 Search Process .....	30
3.2.2 Add record .....	53
4 Discussion and future work .....	56
4.1 Search efficiency .....	56
4.2 Scalability .....	57
4.3 Portability .....	58
4.4 Future Work .....	58
5 Conclusions .....	60
6 References .....	61
7 Appendix .....	63
7.1 Installation Guide .....	63
7.1.1 Install and Configure Apache Web Server 1.3.22 .....	63
7.1.2 Install and Configure Tomcat 3.3 .....	63
7.1.3 Deploy bib web application .....	64
7.1.4. Startup tomcat. Start apache .....	64

<b>7.2 User Manual .....</b>	<b>65</b>
<b>7.3 Source Code .....</b>	<b>73</b>
7.3.1 General search source code .....	73
7.3.2 Advanced search source code .....	73

# **1. Introduction**

## **1.1 The background of BibTeX**

BibTeX is a program and file format designed by Oren Patashnik and Leslie Lamport in 1985 (1) for the LaTeX document preparation system. The format is entirely characters based plain text, so BibTeX can be used by any program. It is tag based and the BibTeX program will ignore unknown fields, so it is expandable. The purpose of this BibTeX program includes: letting the style file worry about formatting the bibliography, avoiding retyping the same references for the next paper (even if it is for a journal with a completely different bibliography style), and being efficient and easy to use.

To use BibTeX, three steps are involved: 1. Create a database (.bib) file that describes the articles that you want to reference; 2. Specify the style and location of the bibliography in your LaTeX document; 3. Run LaTeX and BibTeX.

This project is related to the first step, updating the bib file and searching the interesting entries in the bib file. With the search results, the user can reference the entries in the bib file. For the purposes of this project, the explanation of the content of bib file and the format of the bib file are needed.

### **1.1.1 The content of bib file**

A bib file contains two kinds of entries: abbreviations and bibliographic entries.

### 1.1.1.1 Abbreviations

Abbreviations are used to reduce the amount of writing that a user has to do. For example, the abbreviation `@string { cacm = "Communications of the ACM" }` allows the user to write `journal=cacm` instead of `journal="Communications of the ACM"`.

It can maintain abbreviations in a database server, like Network Bibliography (2), using a set of semi-standard abbreviations. Usually, it is predefined according to the domain of project, such as names of commonly referenced journals. In our project, we provide the most commonly used set of abbreviations, these are the names of months: bibliographic entries typically contain fields like `month=dec` which expands to "December".

### 1.1.1.2 Bibliographic entries

A bibliographic entry has an *entry type* followed by a *key* and a list of *fields*. Each field has a *tag* and a *value*.

The entry types are: article, book, booklet, conference, inbook, incollection, inproceedings, manual, mastersthesis, misc, phdthesis, proceedings, techreport, and unpublished. Database entries of different types have different required fields and optional fields.

Normally, the format of the key is chosen by users. Valid keys include: hoare94, Hoare1994 etc. The key must be unique. BibTeX allows tags to have any value, provided

that they do not contain punctuation marks, quotes, or brackets. The following are recognized tags: address, chapter, howpublished, month, pages, title, annote, crossref, institution, note, publisher, type, author, edition, journal, number, school, volume, booktitle, editor, key, organization, series, and year. Other user-defined tags will be ignored by BibTEX.

### **1.1.2 The format of bib file**

There are various ways of formatting Bibtex entries. Here we describe only the most common format.

The start of an entry is indicated by a “@”. A .bib file may contain comments and other text, but Bibtex looks for “@” before doing anything. The “@” must be followed by an entry type. Bibtex is not case-sensitive and recognizes, for example, @book, @Book, and @BOOK, all of which introduce an entry for a book. The entry itself is enclosed between “{” and “}”.

The first item after “{” is the key which is followed by a number of fields, separated by commas. Each field consists of a tag, an “=”, and a value. The value must belong to one of the following categories:

- 1.A number. Example: 1997.
- 2.An abbreviation. Examples: cacm, dec.
- 3.A string enclosed in quotes. Example: “Donald Knuth”.
- 4.A string enclosed in braces. Example: {Bill Gates}.

## 1.2 BibT<sub>E</sub>X database and query

A bib database is needed to create a citation by the BibT<sub>E</sub>X program. The bib database traditionally consists of files with a bib extension. The bib file contains the bibliographic entries. As Extensible Markup Language (XML) has emerged in recent years, some researchers have developed the XML representation of Bibtex data (3, 4, and 5). Since there are powerful tools and methods for XML, such as XML parsers, it is easy to manipulate the database. The Bibtex data format can be converted back from XML when needed. However, these researchers didn't keep the original data in Bibtex format. This gives some challenge to bib database administrators who may not know XML and may be more comfortable with the traditional Bibtex format.

Updating and searching the bib database is necessary. Jonas Björnerstedt's BibEdit project (6) is an example of a program for updating a bib database. The main idea of BibEdit is to make navigation and editing simple. BibEdit has two views: list view and record view. In the list view, each record is a row in the list. In record view, one record is displayed for editing. Double-clicking on a record in list view also changes to record view. The current version of BibEdit has a rather primitive search feature. In record view, it starts searching for the next record containing the text. In list view it selects all records containing the text. BibEdit searches the contents of all fields to see if they contain the text. If the user searches the string "Test", the search engine will return all entries that include "Test" in any field of the entry. Therefore, BibEdit is more useful for editing the bib file than for searching.

For searching Bibtex bibliography files, BibSearch is developed by Nelson Beebe (7). This software uses the mgquery database search engine to provide super fast searching in a collection of Bibtex bibliography database files. The database is generally updated nightly, and on startup, bibsearch displays the date of the last update, plus some statistics on the size of the collection. It also gives some helpful hints about common-used commands. By default, bibsearch uses query-ranked searching: you type several words, and the search engine responds with a sorted list of bibliography entries that contain one or more of those words, in order of decreasing number of matches. Searches always ignore letter case. Partial word matches are not usually accepted: if you search for ``tex'', neither ``text" nor ``texture" will match. This software doesn't provide a user interface, so the user needs to use the command line to perform search.

Gerd Herzog and Clemens Huwig's Literature Information and Documentation System (LIDOS) (8) offers a large bibliographic database. This bibliography covers mainly research in Artificial Intelligence and related fields. LIDOS aims to provide easier access to bibliographic data and associated on-line information. It provides a web based search interface. The bib files are organized by category. This system uses CGI technology on the server side. They only provide for general search by author, title word(s), or biblook query format. Query processing may require up to one minute. This application is not designed for updating the bib database.

## 1.3 The goal of this project

By reviewing the previous work about the Bibtex search, we want to develop a web application to process the Bibtex search. This web application will have an easy-to-use interface and provide users with a high speed and advanced search engine and a facility to add entries to a bib file through the Internet. Also this server provides some convenient functions for the users to save the search results to local disk or email the search results to others. The capabilities of this application include:

- Searching: the user enters information about one or more fields and the server responds with a list of matching entries. The user can modify the initial search result .
- Searching options: match case, whole words, and whole field.
- Control buttons: "Search" button triggers the search process. "Stop" button stops the search process.
- Bibliographic data: the user selects one or more entries provided after a search request and can obtain full citation information for those entries in the Bibtex format.
- The search result can be displayed on the screen.
- The search result can be saved to local disk.
- The search result can be sent by email.
- Allow users to "batch" entries and have all of the information displayed or sent at once.

- Update: privileged users can add bibliographic information to the server's database manually or through the Internet. A privileged user can access the database by secure means, for example by entering the user name and password.

## **2 Design**

### **2.1 Architecture design**

#### **2.1.1 Design rationale**

According to the project requirements, the objective of this project is to enable a user to search bibliographic information through internet. The data sits on the server side, and the client uses the web browser to search information from the server side. The client-server communication uses the HTTP protocol. Due to these facts, we are building this project as a typical web application. For implementing this system, we have to make the decision to choose the appropriate technology. We particularly paid attention to some issues such as how we should process the data, how we can make the search process efficient, how we can make the application portable, and so on.

##### **2.1.1.1 Use Java servlet technology as server side implementation**

The common existing technologies for dynamic content generation on the server are Common Gateway Interface (CGI), Active Server Pages (ASP) from Microsoft, and Servlet from Sun.

Java servlets are more efficient, easier to use, more powerful than traditional CGI. With CGI, a new process is started for each HTTP request (9, 10). If the CGI program itself is relatively short, the overhead of starting the process can dominate the execute time. With

servlets, the Java Virtual Machine (JVM) stays running and handles each request using a lightweight Java thread, not a heavyweight operating system process. Similarly, in traditional CGI, if there are N simultaneous requests to the same CGI program, the code for the CGI program is loaded into memory N times. With servlets, however, there would be N threads but only a single copy of the servlet class. Servlets can also maintain information from request to request, simplifying techniques like session tracking and caching of previous computations.

ASP also supports the creation of dynamic web pages. Compared to Java servlets, ASP is platform dependent. It only runs on Microsoft IIS or PWS web server. So, servlets are more portable than ASP. They are written in the Java language and follow a standard API. Consequently, servlets can run on most web servers. When servlets are exported to another web server, there is no need to change the code.

#### **2.1.1.2 Use XML as data storage**

There are at least three ways in which the server might store bibliographic data: Bibtex format, database tables, or the Extensible Markup Language (XML). Even though a relational database can store a large amount of data, there are two limitations for using relational database system to store Bibtex information. First, there are about twenty-four tags that Bibtex can recognize. Each record in Bibtex format doesn't need to have all the tags and values. The space will be wasted in a table with one column for each tag, because most columns in a typical entry will be empty for a record with just a few tags.

Second, and most seriously, Bibtex allows arbitrary tags, but the structure of a database table is fixed when the relational database is designed. If we change the Bibtex tags, we have to change the database tables. For these reasons, using a relational database to store the Bibtex information is not appropriate.

Bibtex maintains the data as plain text in files. It has its own strict syntax. As we discussed in the introduction part, the Bibtex data can be manipulated by using some available tools, and an XML representation of the Bibtex data has been developed. The advantage of the XML representation over BibTeX's native syntax is that it can be easily managed using standard XML tools (XML parser, XSLT style sheets, etc.), while native Bibtex data can only be manipulated using specialized tools (11). XML is the Extensible Markup Language that specifies neither the tag set nor the grammar for that language (12, 13). User can define any tag in XML. This flexibility of XML makes it the most appropriate way to store the Bibtex data. For this project, our solution is to convert the Bibtex plain text file to an XML file, then parse the XML file into a document tree. In this way, the original Bibtex file is kept. It provides the data administrator more flexibility to maintain the data in its original format. The administrator doesn't need to know XML. Also it provides a very efficient way to search information, because the data stays in the memory after parsing the XML file into a document tree. When a user makes a request, the data will be fetched from the memory on the server side without the need of getting data from secondary storage.

Since the data is stored in the memory, this would require a very large space. In our project, the Bibtex information is not very large-scale data. More precisely, we pay much more attention on searching speed than the space the Bibtex data needs. For a large bib database, the solution could be to classify the bib files according to their category. This is going to be the future work. We discuss it in the “Discussion and future work”, on page 55. Balancing the space and efficiency needs for this particular problem domain in this project, we use XML to store data.

### **2.1.1.3 Use JDOM API to parse XML**

To parse an XML file, we need an XML parser. There are two kinds of XML parsers: one implements the Simple API for XML (SAX), and another one implements the Document Object Model (DOM).

SAX presents a view of the document as a sequence of events (14). It parses a document incrementally. It reads the XML document and raises events for the different elements it encounters. With defined callback functions, some tasks can be processed at the different stages of parsing. The advantage of SAX is that it doesn't need large memory. However, the sequential model that SAX provides does not allow for random access to the XML document. It is difficult to compare different elements by going back and forth. Basically, SAX is the better choice for quick, less-intense parsing and processing.

DOM represents a document tree held fully in random memory. It is a large API designed to perform almost every conceivable XML task. DOM also requires lots of processing power and memory, but it provides an easy-to-use, clean interface to data in desirable format, and provides fast access to entire data.

JDOM is a Java-based "document object model" for XML files (15). It is not an XML parser, rather a document object model that uses XML parsers to build documents. It is a sort of hybrid approach between DOM and SAX. The JDOM API lets you use either DOM or SAX parser under the hood. The JDOM API's `SAXBuilder` class is probably the best for parsing a very large XML document since it uses the SAX API for parsing and allows you to modify the document at the same time. Also, JDOM is a complete Java 2-based API, taking advantage of the Java collection classes. For our project, we choose the JDOM API to build the data document.

#### **2.1.1.4 Data processing**

When a user makes a search request or when the administrator adds a new record, the application will access the parsed XML data. Upon each request, we convert the Bibtex file to XML format, and then parse the generated XML file. Since all the users access to the same data, we can maintain only one data structure in the server side. This means we convert the Bibtext file and parse the XML file once for all requests. Further, we can process the converting and parsing on server startup so that the document tree is ready to use before any request coming. One class `StartUp` is designed to accomplish this task.

When the server starts up, an object of this class will be created. For each request, we use this object to access the data structure. Using the Singleton pattern (16), we can make sure only a single instance of this class can be created, enforced through the use of a private constructor. The instance will be retrieved through a static method that checks if there is already an object allocated, returning it if there is one and allocating and returning a new object if not.

In this project, the administrator can add a new record through the Internet or directly to the Bibtex file manually. To do this, it requires that the added record should be visible to all users when they are searching. We decided to convert the Bibtext file to an XML file and parse the generated XML file at regular time intervals. We can accomplish this by making `StartUp` class to be a thread. It will perform the converting and parsing at regular time intervals or as needed. When a record is added to the Bibtex file manually, the result can be obtained after the time interval. When a record is added through Internet, the added record can be queried immediately by notifying the thread to do its task.

Concurrent data access issue needs to be addressed. When a user is searching information, the application reads data from the parsed XML file. After the administrator add a record, the application will write to the Bibtex file, convert the Bibtex to an XML file, and parse the generated XML file. We have to guarantee the data consistency. This can be accomplished by using the Java synchronization mechanism.

### **2.1.1.5 Application portability**

Portability is a major advantage of Java language (17). Java code is compiled to byte code, and then interpreted into machine language on different platforms by the JVM. “Write once run anywhere” is the policy of the Java language. Our data is in the plain text format of Bibtext and XML. Compared to the relational database, the plain text data is completely platform independent. Java language plus XML really makes sense for portability.

Despite of the Java language and text data portability, sometimes the application cannot be completely portable. For example, we need a mail host to send mail. If we coded the mail host in the code, the code has to be changed if we want to use a different mail host at a later time. In this project, a configuration file will be used to store the needed information, such as which mailhost is going to be used, which Bibtext file is going to be converted, which XML file will be parsed after converting, and how long the time interval for processing data will be. This information will be used by the application at run time. We can change the parameters without changing the application code. Doing this makes the application more flexible and portable.

## 2.1.2 System design

### 2.1.2.1 Package view

According to their usage, all classes are separated into two packages: “bibsearch” and “bibupdate”. The relationship between two packages and environment packages is shown in Figure 3.

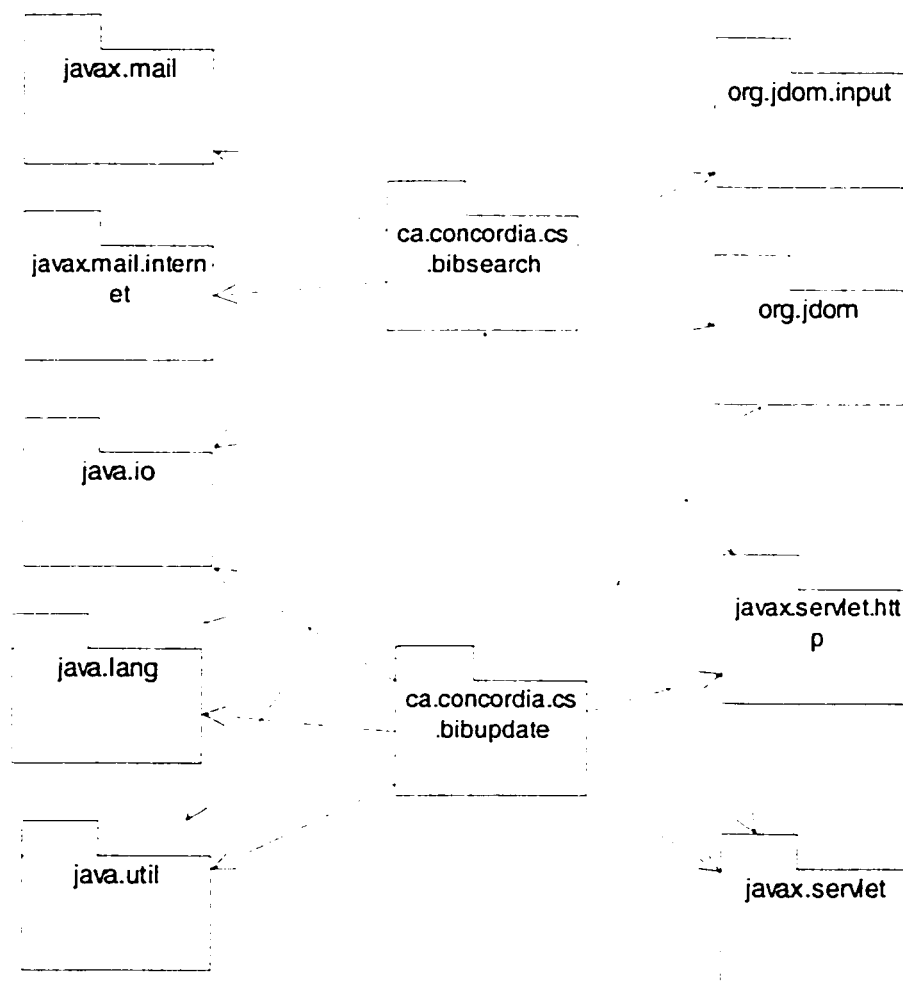


Figure 3: Package view

### **2.1.2.2 Class relationships**

The relationships between the classes in two packages are shown in Figure 4. Most of classes extend the `HttpServlet` class. The `Converter` class is used to convert a Bibtext file to an XML file. The `StartUp` class extends `Java Thread` class. It uses the `Converter` class to generate an XML file, and uses the JDOM API to parse the XML file into an JDOM Document.

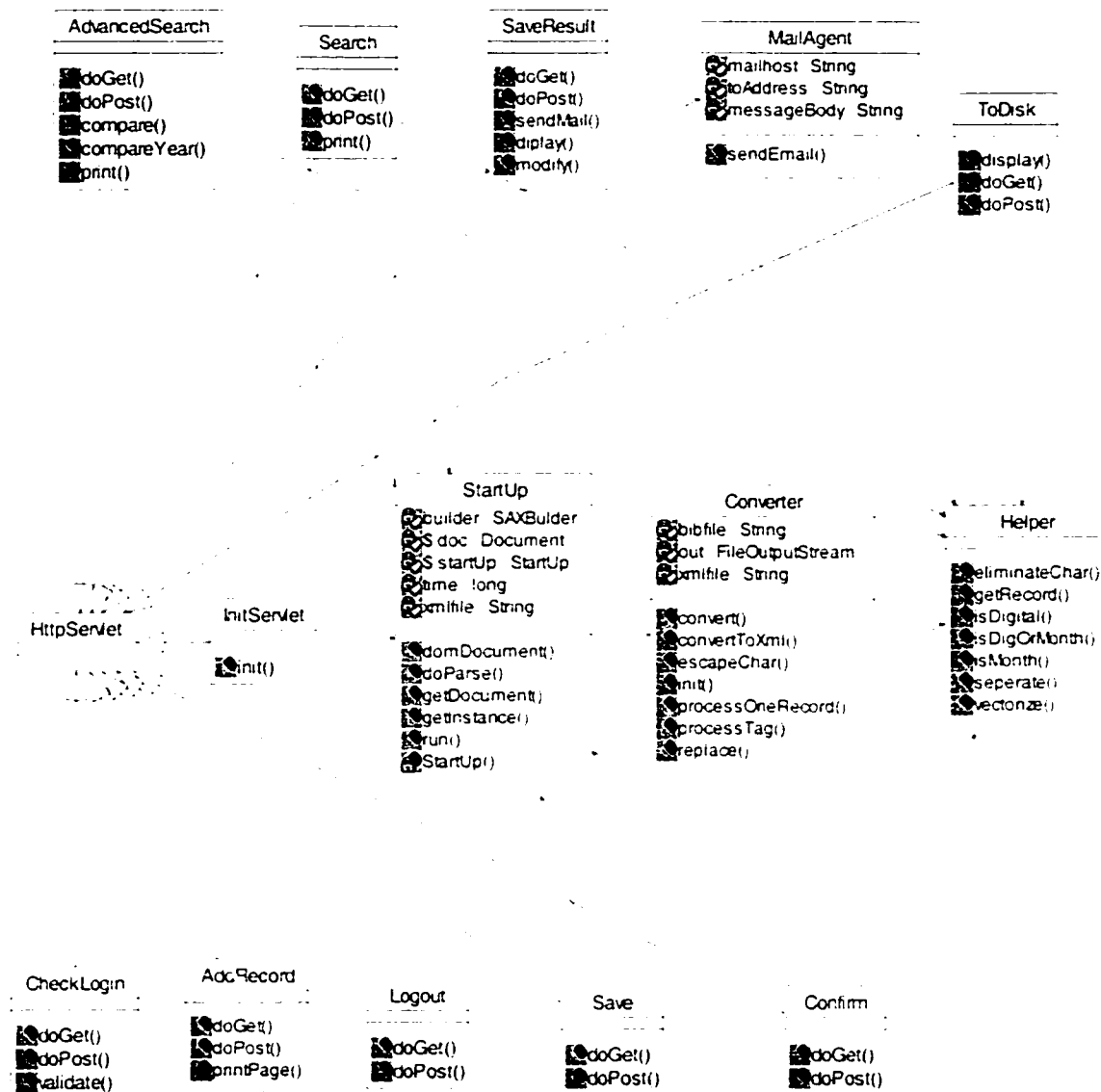


Figure 4: Class diagram

### 2.1.2.3 Sequence diagrams

There are two kinds of users for this application. One is the administrator who can add record to the data storage through the Internet or manually. The Figure 5 shows the time-ordering of messages when the administrator adds a record through the Internet.

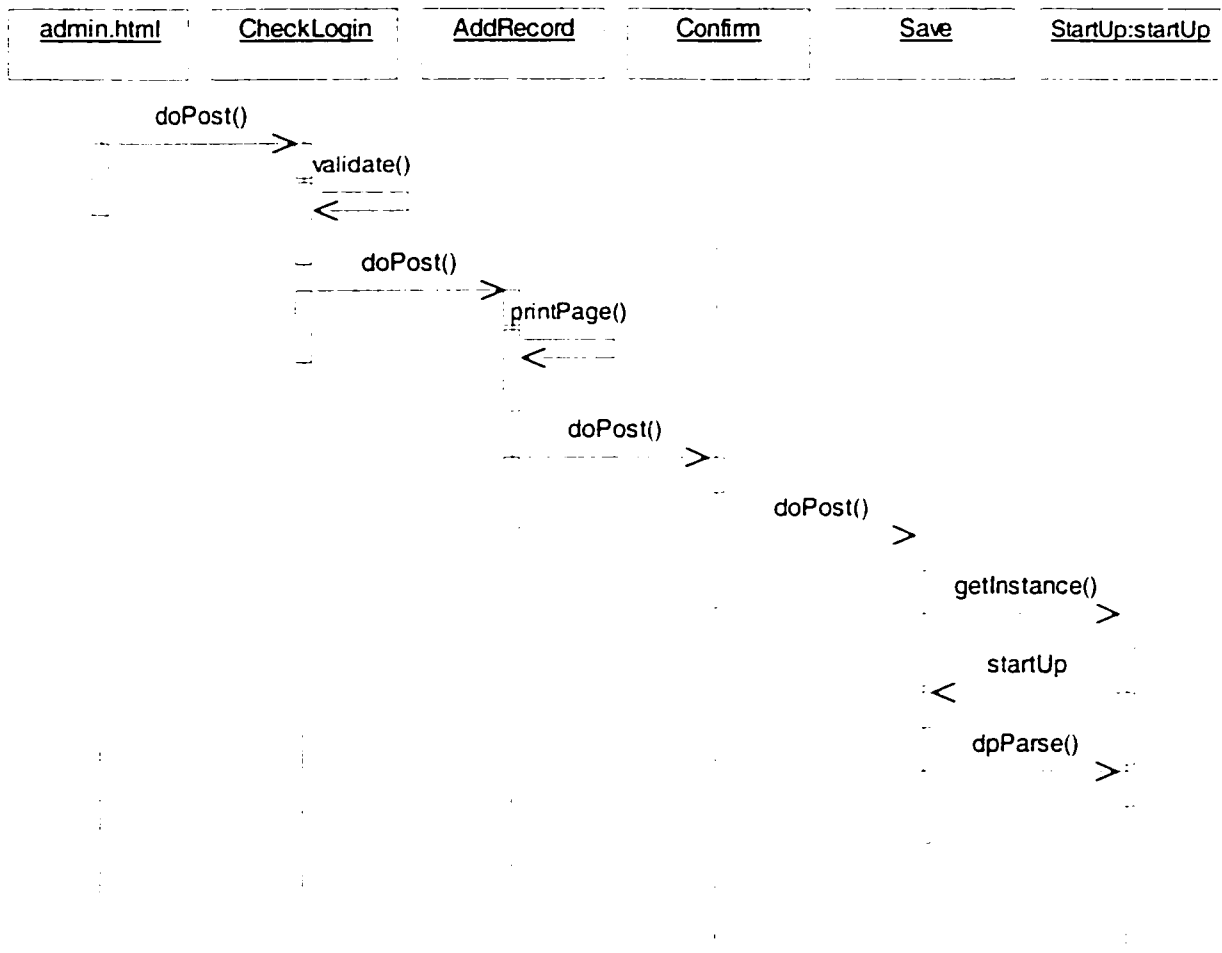


Figure 5: Sequence diagram for adding a record

Another kind of user can just search the Bibtex information through the general search and advanced search html pages. They do not have to be authenticated to access this system. When a user inputs the search query and the process begins searching, the application will create a session object in the server side. By using a session object, the information can be passed among different pages. The server will maintain a unique session object for each user. The search process is showed in Figure 6.

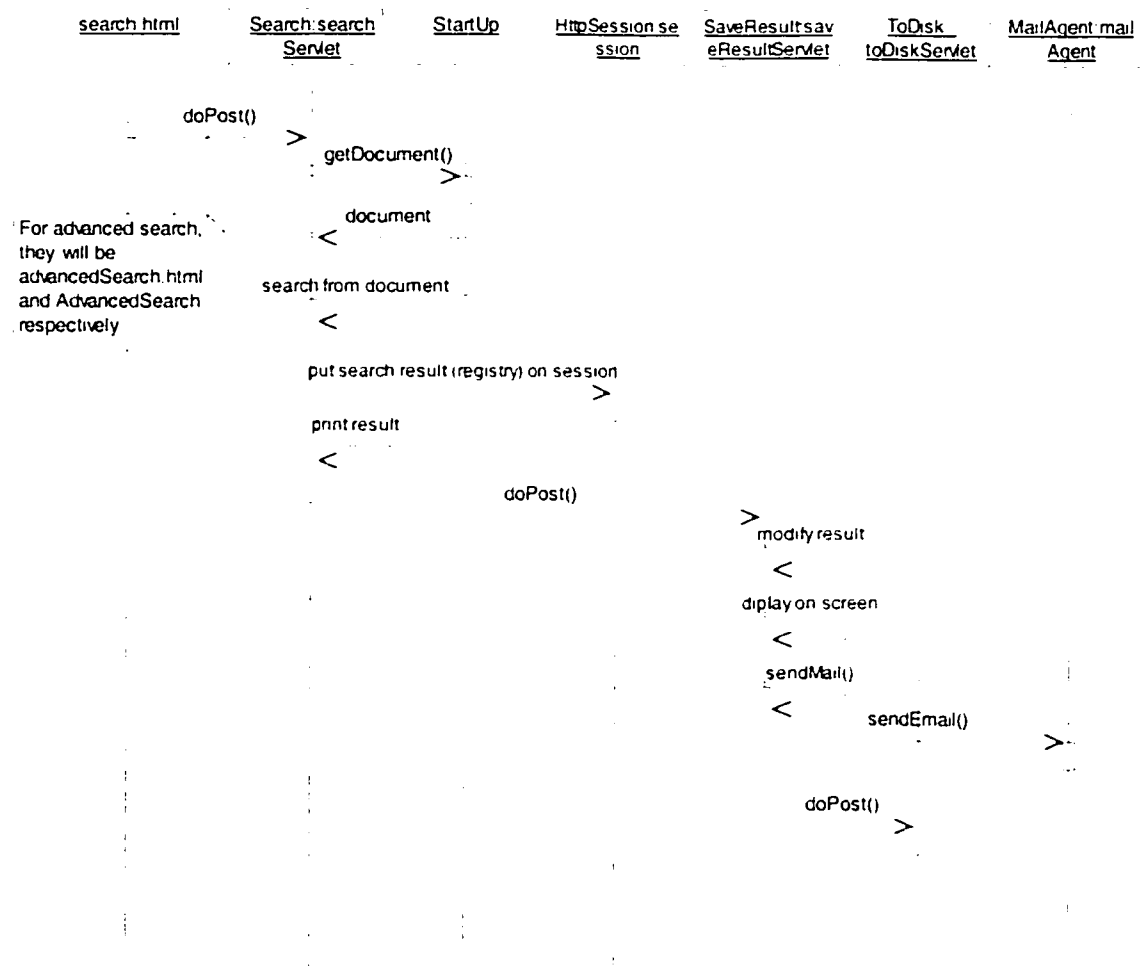


Figure 6: Search process sequence diagram

#### 2.1.2.4 Page navigation

This project is a web application. All interfaces will be static or dynamically generated html pages. We design to make it easy for users to navigate different pages when they are processing their tasks. For example, when searching information, the user can go back to the general search and advanced search pages at any time. For searching process, the page navigation is described as in Figure 7.

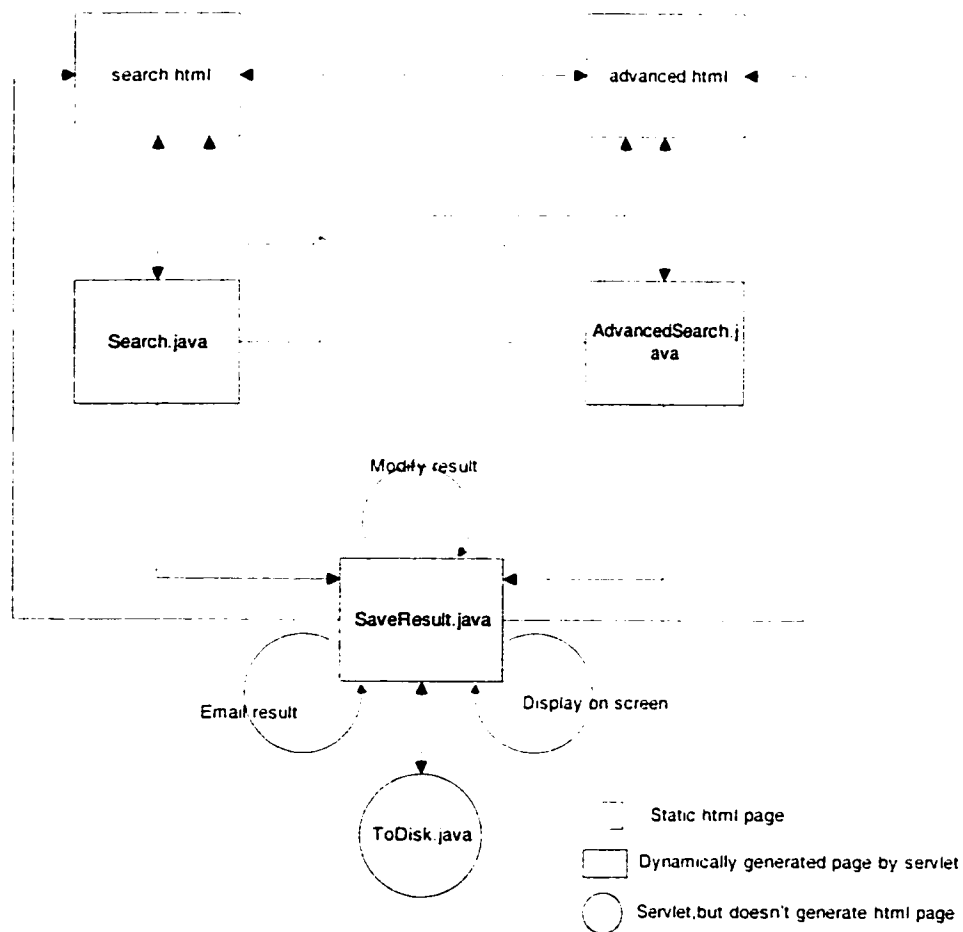


Figure 7: Page navigation for search process

For adding records, we have to take care of security issues. Only the authorized users can add records to data storage. In each page, we need to check if there is a user logged in. This can be done by taking advantage of the session object. When a user logs in, we can put the user object on the session. In every page, we check if the session has the user object. If it doesn't, we send the user to the login page. When the user logs out, we remove the user object from the session. In this way, unauthorized users cannot add records even though the user can type the address of the servlet into the browser directly. The page navigation for adding record is shown in Figure 8. At every stage, the user can log out.

The http is a stateless protocol. It can't remember the client state. For example, if the authorized user goes back to the confirmation page (`Confirm.java` as in Figure 8) by clicking browser's back button after saving the record, the user might click the "save" link again. In this case, we have to make sure the same record can't be added twice. From the confirmation page to the save page (`Save.java` as in Figure 8), the record is passed by session. After saving the record, the record is removed from session. In this way, the same record won't be saved more than once. The user can click on any link when navigating the pages.

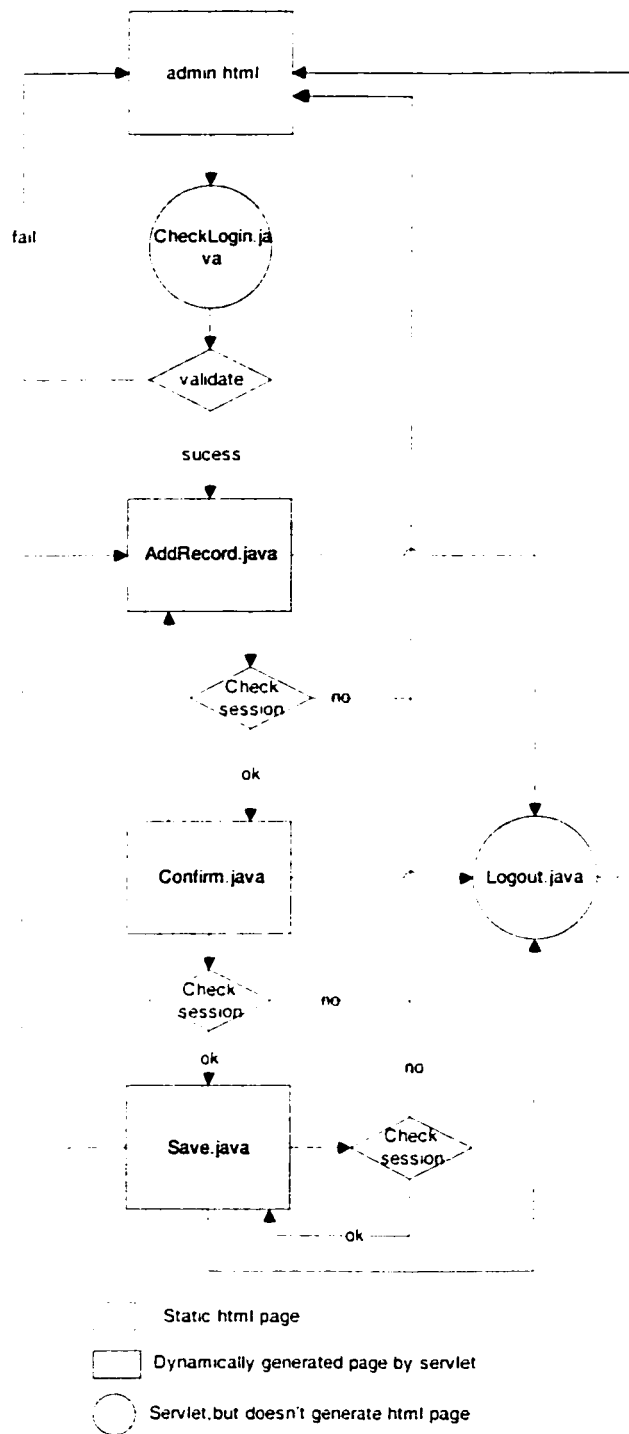


Figure 8: Page navigation for adding a record

### **2.1.3 System tools and environment**

We use Apache (version 1.3.23) (18) as the web server. It is open source. Apache has been the most popular web server on the Internet since April of 1996 (19). The January 2002 Netcraft Web Server Survey found that 56% of the web sites on the Internet are using Apache, thus making it more widely used than all other web servers combined. We use Tomcat (version 3.3) (20) as the servlet engine. Tomcat is an open source tool from Apache group. It implements the Servlet 2.2 and JSP 1.1 specifications. Even though Tomcat itself can be a web server, the web server Apache is much more stable, and it is faster when handling static html documents. As Figure 9 shows, if the client requests static html pages, Apache web server sends the page back to user directly. If the client requests a page which is a servlet, Apache web server then passes the request to Tomcat servlet engine which in turn gets the data from JDOM document and sends back a dynamically generated page.

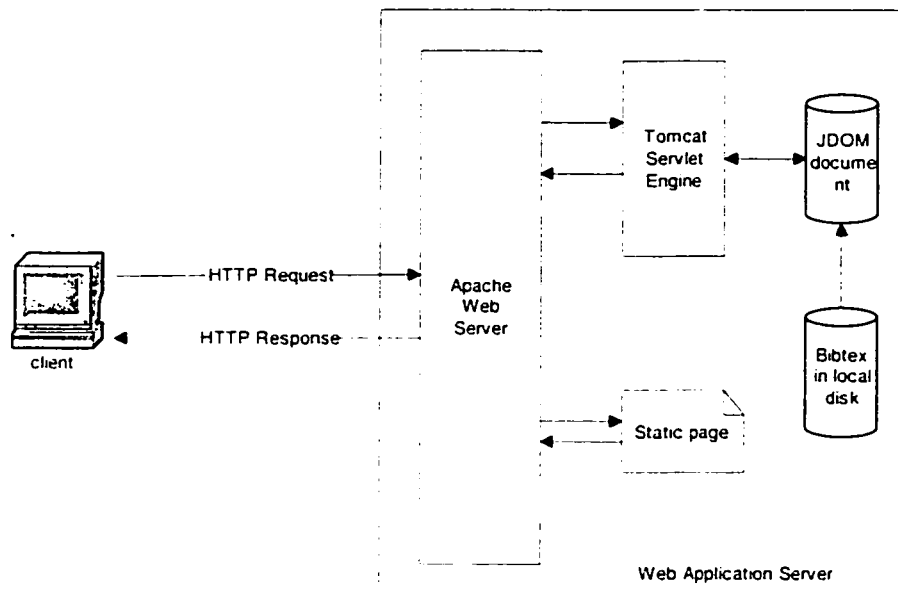


Figure 9: System tools and environment

## 2.2 Module interface design

As the Figure 2 shows, most of our classes extend the `HttpServlet` class. Their interfaces are quite simple and have `doGet()`, `doPost()` and a few helper functions. In this part, we describe the responsibility for each class, but will only show the interfaces of class `StartUp`, `Converter` and `Helper`.

The Class `StartUp` (Figure 10) is a subclass of the `Thread` class. When the server starts up, an object of `StartUp` will be created and keep running as a thread. By using Singleton pattern, only one object of `StartUp` will be maintained. At a time interval which can be read from configuration file (default value is twenty four hours as the value of data

member time), this thread converts the Bibtex file to an XML file and parses the XML file to a JDOM document tree which stays in the server's memory. This thread can be notified after a record is added to keep the data updated. For concurrent access to the document tree, the methods `domDocument()`, `doParse()`, and `getDocument()` are synchronized.

```
public class StartUp extends Thread{
    private org.jdom.input.SAXBuilder ;
    private static Document doc;
    private String path;
    private long time = 1000*60*60*24;
    private static StartUp startUp;
    private String xmlfile = "info.xml";

    private StartUp(String path);
    public static StartUp getInstance(String path);
    public static StartUp getInstance();
    public void run();
    public synchronized void doParse();
    public synchronized void domDocument(File file);
    public synchronized static Document getDocument();
}
```

Figure 10: Interface of class StartUp

The class Converter interface is shown in Figure 11. The responsibility of this class is to convert the Bibtex file to an XML file. It reads the Bibtex file line by line, and writes to an XML file when finished getting one record. In XML, some characters are reserved such as '&', '<', '>'. In converting process, they are replaced by "&"; "<"; and ">". Otherwise, there will be parsing errors. In the Bibtex file, sometimes the text in a field must be protected. This is done by enclosed the text in braces. When converting to

```

public class Convert {
    private FileOutputStream out;
    private PrintWriter writer;
    private String path;
    private String xmlfile = "info.xml";
    private String bibfile = "text.bib";

    public Convert(String path);
    public void init(String path);
    public synchronized void convertToXml();
    public String escapeChar(String str);
    public String replace(String str, char c, String replacement);
    public void processOneRecord(String record);
    public void processTag(String tagAndValue);
}

```

Figure 11: Interface of class Converter

an XML file, we will leave it as it is to present the original format to the users. In this class, concurrently access to the Bibtex and XML files will be protected by using the Java synchronization mechanism.

The `Helper` class (Figure 12) provides some utility functions for other classes. All the methods are static so that they can be called on the class directly.

```

public class Helper {
    public static String eliminateChar(String str);
    public static boolean isDigital(String str);
    public static boolean isMonth(String value);
    public static Vector seperate(String str, String sign);
    public static Vector vectorize(String str, String delim);
    public static String getRecord(Element e);
    public static boolean isDigOrMonth(String value);
}

```

Figure 12: Interface of class Helper

The class `InitServlet` creates an object of type `StartUp`, and starts this thread. This class will be loaded upon server startup. This will be specified in the `web.xml` file of the web application.

The class `Search` processes a general search. It gets input from the user, and searches the information from the `JDOM` document fetched from the object of the class `StartUp`. Then it puts the initial search results on the session object. The initial search results will be printed out in the browser and lets the user to choose the interested ones that will be passed to the class `SaveResult`.

The class `AdvancedSearch` processes an advanced search. It does the same thing as `Search` does, but searches the information with a different algorithm.

The `SaveResult` servlet will get the results in which the user is interested by the request object. Then modify the initial search results in the session object. This class can modify the results for the user, can print the result details in the `BibTeX` format in the browser to enable the user to copy and paste, can email the result details, and can send the request to the `ToDisk` class to save the results on local disk.

The `ToDisk` class will be responsible for saving the search results to local disk.

The `CheckLogin` class will validate the user. If the user is authenticated, it will send the user to the page generated by the `AddRecord` servlet and put the user on the session object, otherwise back to the login page.

The `AddRecord` servlet will generate the page with html form and pass the form fields to the `Confirm` servlet.

The class `Confirm` will get the form fields by the request object and validate them. If the key is empty, the key is not unique, or some fields are not in required format, it will send the user back to the previous page with a specific error message. If the validation passes, this class will print the record in Bibtex format in the browser for confirmation.

The `Save` servlet will get the added record from the session object and append it to the `BibTeX` file, and notify the `Startup` thread. After saving, it will remove the record from the session object.

The class `Logout` will remove the user object from session, invalidate the session object, and sends the user to the login page.

## **3 Implementation**

### **3.1 System environment**

Windows 98/NT/2000 is needed for installing this system. We use Apache 1.3.22 as the web server, and use Tomcat 3.3 as the servlet engine. Apache and Tomcat are open source tools and can be downloaded from Apache web site (18, 20). They are easy to install according to the installation guide (See Appendix).

### **3.2 Function implementation**

This is two-group members' project. One member is I, and another member is Mr. Aimin Han who is a master graduate student at Computer Science Department of Concordia University. The project is designed by both of us. The functionalities of this project include the search process and adding records to the database. The search process in turn contains the general search and the advanced search. We developed the search algorithms together. I mainly implemented the search process. Mr. Aimin Han put a lot of work in implementing the `StartUp` and `Converter` classes and the functions for adding records to the database. However, we always discussed the problems we encountered. There are a lot of overlaps between our works. I will mainly describe the search detail implementation. For other functions, Mr. Aimin Han will provide the detailed implementation in his major report.

### 3.2.1 Search Process

#### 3.2.1.1 Convert the Bibtex file to an XML file

This function is processed in the class Converter. The Bibtex file is read in line by line. In the Bibtex file, each record is labeled with “@” in the beginning of each record. Once we read a record, we process it in the function processOneRecord. To demonstrate the conversion process, we will give an example of a Bibtex record:

```
@book{abel194, author="Martha L. Abell and James P. Braselton",  
title="The Maple V Handbook", publisher=academic, year=1994,  
note="{\callnum{QA 76.95 A213 1994}}"}{}
```

To find the record type and key, we find the first character of “{”. then we know the word between “@” and the first “{” will be the type. The word between the first “{” and the first comma “,” will be the key. The phrase between the first comma and the last “}” will be the tags and values. Once we get the tag phrase, we then call the processTag function. For the above example, the tag phrase will be:

```
author="Martha L. Abell and James P. Braselton",  
title="The Maple V Handbook", publisher=academic, year=1994,  
note="{\callnum{QA 76.95 A213 1994}}"
```

We can get the first tag easily by finding the position of the first equation “=”. There are two cases for the first character after the first equation. It could be a double quotation or

not. If it is, we need to find another double quotation matching with it. We sequentially find the double quotation which is not prefixed with character “\”. In this way, the value for the first tag will be found. We will call the processTag function recursively for the rest part:

```
title="The Maple V Handbook", publisher=academic, year=1994,  
note="{\callnum{QA 76.95 A213 1994}}"
```

After finishing processing one record, the XML output will be:

```
<book>  
  <key>abel194</key>  
  <author>Martha L. Abell and James P. Braselton</author>  
  <title>The Maple V Handbook</title>  
  <publisher>academic</publisher>  
  <year>1994</year>  
  <note>{\callnum{QA 76.95 A213 1994}}</note>  
</book>
```

The sample code for the functions processOneRecord and processTag is given below:

```
public void processOneRecord(String record){  
    int firstBrace = record.indexOf('{');  
    String type = record.substring(1, firstBrace);  
    int firstComma = record.indexOf(',');
```

```

String key="";

if(firstComma < 0){

    key = record.substring(firstBrace+1, record.length()-1);

    writer.println("\t<" + type + ">");

    writer.println("\t\t<key>" + key.trim() + "</key>");

    writer.println("\t</" + type + ">");

}

else{

    key = record.substring(firstBrace+1, firstComma);

    String tagAndValue = (record.substring(firstComma+1,
                                           record.length()-1)).trim();

    writer.println("\t<" + type + ">");

    writer.println("\t\t<key>" + key.trim() + "</key>");

    processTag(tagAndValue);

    writer.println("\t<." + type + ">");

}

}

public void processTag(String tagAndValue){

    String rest="";

    if(tagAndValue.length() == 0 || tagAndValue.equals("") ||
        tagAndValue == null)

        return;

    else{

        int firstEq = tagAndValue.indexOf('=');

        if(firstEq >= 0){

            String tag = tagAndValue.substring(0, firstEq);

            String tempValue = (tagAndValue.substring(firstEq+1,
                                                       tagAndValue.length())).trim();

```

```

char c = tempValue.charAt(0);

if(c == '\\'){
    int secondQt = tempValue.indexOf('\\', 1);
    int temp = secondQt;
    while(tempValue.charAt(temp-1) == '\\'){
        secondQt = tempValue.indexOf('\\', temp+1);
        temp = secondQt;
    }

    String value = (tempValue.substring(1, secondQt)).trim();
    int comma = tempValue.indexOf(',', secondQt);
    if(comma >= 0)
        rest = (tempValue.substring(comma+1,
                                     tempValue.length())).trim();

    writer.println("\t\t<" + tag + ">" + value + "< " + tag + ">");
}

else{
    int secondEq = tagAndValue.indexOf('=', firstEq+1);
    String value="";
    if(secondEq >= 0){
        int coma = tagAndValue.lastIndexOf(',', secondEq);
        value = (tagAndValue.substring(firstEq+1, coma)).trim();
        rest = (tagAndValue.substring(coma+1,
                                     tagAndValue.length())).trim();
    }
    else{
        value = (tagAndValue.substring(firstEq+1,
                                     tagAndValue.length())).trim();
    }
}

```

```

        }
        writer.println("\t\t<" + tag + ">" + value + "</" + tag + ">");
    }
}

}

processTag(rest.trim());
}

```

There is one thing we have to pay attention to. In the Bibtex record, the tag value may contain characters "&", "<" or ">", those characters have to be replaced by "&"; "&lt;"; and "&gt;". Otherwise, there will be parsing errors when parsing the XML file.

### 3.2.1.2 Parse the XML file

By using the JDOM API, parsing an XML file is easily done. This process is done in the class `StartUp`. Sample code is given below:

```

SAXBuilder builder = new SAXBuilder()
File file = new File(xmlfile);
Document doc = builder.build(file);

```

The `doc` object represents the parsing tree. We can find information from the tree. Here we use the `SAXBuilder` instead of the `DOMBuilder`. The difference is that the `DOMBuilder` builds the whole tree at once, and the `SAXBuilder` doesn't load the

entire document. Rather, it will let you get an element, and then iterate over the element's children. It is better for parsing large XML files (14).

### **3.2.1.3 The StartUp thread**

For search efficiency, we want to convert the Bibtex file and parse the generated XML file on the server startup. Also, we want the Bibtex information to keep updated. This can be done by implementing a thread which only will be maintained as one copy.

We use the Singleton design pattern to create this thread. The class `StartUp` has a private constructor, and a static method `getInstance`. The client only can get an object of `StartUp` by calling the `getInstance` method. This `StartUp` object is a static data member, which means only one copy will be obtained. In the `getInstance` method, it checks if the static data member is null. If it is null, it will call the private constructor to create a new object and return it, otherwise will return the one existed. The snippet of code shows the approach:

```
public static StartUp getInstance(String path){  
    if(startUp == null)  
        startUp = new StartUp(path);  
    return startUp;  
}
```

In the run method of the class StartUp, convert the Bibtex file to an XML file, and parse this XML file by calling the domDocument method. Then wait for an time interval. The new information will be visible after this time interval or being notified. The sample code is given below:

```
public void run(){
    while(true){
        Convert convert = new Convert(path);
        convert.convertToXml();
        domDocument(new File(path + xmlfile));
        try{
            synchronized(this){
                wait(time);
            }
        }catch(InterruptedException e){
            System.out.println("Interrupted when waitinng. " + e);
        }
    }
}
```

An InitServlet class is implemented. This servlet will be loaded on server startup.

In the init method, it creates the StartUp thread, and starts it.

```
public void init(){
    String path = getServletContext().getRealPath("/") + "files\\";
    StartUp startUp = StartUp.getInstance(path);
    startUp.start();
}
```

This path is the location in which the configuration file stays.

### 3.2.1.4 General search

The general search is basic and important part of this project, users use general search frequently. We discuss the algorithm carefully, in order to achieve high search performance. In general search, the user can only search by one catalog, namely, one of author, title, booktitle, published after a year, published before a year, and journal. Three search options, namely, match case, and/or match words, and/or match whole field are provided for each search catalog. The algorithm and implement are as below:

Step1. Get the search type by the `request` object.

```
String searchType = (request.getParameter("search_type")).trim();
```

Step2. Get the search key word by the `request` object.

```
String searchName =(request.getParameter("name")).trim();
```

Step3. Get the search options and decide which search condition case it belongs to. There are five cases by combining the search options.

```
boolean matchCase = ((request.getParameter("case")) != null );
```

```
boolean matchWord = ((request.getParameter("word")) != null);
```

```
boolean matchField = ((request.getParameter("field")) != null);
```

```
int searchCondition;
```

```
if((matchCase == false) && (matchWord == false) && (matchField == false))
```

```
    searchCondition = 0;
```

```
else if((matchCase == true) && (matchWord == false) && (matchField == false))
```

```

        searchCondition = 1;

    else if((matchCase == false) && (matchWord == true) && (matchField == false))

        searchCondition = 2;

    else if(((matchCase == false) && (matchWord == false) && (matchField == true))

            ||((matchCase == false) && (matchWord == true) && (matchField ==

true)))

        searchCondition = 3;

    else if((matchCase == true) && (matchWord == true) && (matchField == false))

        searchCondition = 4;

    else

        searchCondition = 5;

    if(searchName.equals("") || searchName.length() == 0 || searchName == null){

        response.sendRedirect("index.html");

        return;

    }

```

Step 4: Get the JDOM document which refers to the Document tree.

```
Document doc = StartUp.getDocument();
```

Step 5: Initialize list of element nodes.

```
Element root = doc.getRootElement();
```

```
List nodes = null;
```

Step 6: If the search type is booktitle, fill the list with the conference nodes. If the search type is journal, fill the list with the article nodes. Otherwise fill the list with all nodes.

```
if(searchType.equals("booktitle")){  
    nodes = root.getChildren("book");  
    List nodesTwo = root.getChildren("conference");  
    nodes.addAll(nodesTwo);  
}  
  
else if(searchType.equals("journal"))  
    nodes = root.getChildren("article");  
  
else  
    nodes = root.getChildren();
```

Step 7: For each node in the list:

-- Get the key from the node.

```
ListIterator iter = nodes.listIterator();
```

```
while(iter.hasNext()){
```

```
    Element element = (Element)(iter.next());
```

```
    String key = element.getChildText("key");
```

```
    ...
```

```
}
```

-- Get the corresponding value from the node according to the search type.

There are five cases here. I only give an example of search by author:

```
if(searchType.equals("author")){
```

```
String author = Helper.eleminateChar(element.getChildText("author"));

if(author == null)

    author = Helper.eleminateChar(element.getChildText("editor"));

-- Compare the search key word with the value from the node with constraints of

the search condition case and the search type.

Example of search by author, match case:

if((author != null && author.indexOf(searchName) >=0)){

    registry.put(key, element);

-- If the comparison result is true, then put the key and the element node into a

Hashtable which represents the initial search result.

See above.
```

### **3.2.1.5 Advanced search**

For an advanced search, the user can input search values for author, title, from year, to year, and journal name. The user doesn't have to input all fields. The application will combine the input values as the search criteria for the filled fields.

The algorithm and implement are as below:

Step 1: Get the form input names and values. Add them to input vector. They are author,

title, from year, to year, and journal.

Here, check input error of year too.

```
Vector paras = new Vector();
```

```
Vector type = new Vector();
```

```

String author = (request.getParameter("author")).trim();

paras.addElement(author);

type.addElement("author");

String title =(request.getParameter("title")).trim();

paras.addElement(title);

type.addElement("title");

String from = (request.getParameter("from")).trim();

paras.addElement(from);

type.addElement("from");

String to =(request.getParameter("to")).trim();

paras.addElement(to);

type.addElement("to");

boolean isDigit = true;

String fromTo = from + to;

if(!Helper.isDigital(fromTo)){

    out.println("<html>");

    out.println("<head>");

    out.println("<title>" + "result" + "</title>");

    out.println("</head>");

    out.println("<body bgcolor=\"white\">");

    out.println("<body>");

    out.println("<p align=\"center\">Input format error, you must input number " +

        "in text field of from year or to year.</p>");

```

```

        out.println("<p align=\"center\"><a href=\"index.html\">Go to genreal
search</a></p>");

```

```

        out.println("<p align=\"center\"><a href=\"advanced.html\">Go to advanced
search</a></p>");

```

```

        out.println("</body>");

```

```

        out.println("</html>");

```

```

        isDigit = false;

```

```

        return;

```

```

    }

```

```

String journal =(request.getParameter("journal")).trim();

```

```

paras.addElement(journal);

```

```

type.addElement("journal");

```

Step 2: Get the JDOM document that refers to the Document tree.

See step 2 of general search.

Step 3: Create a list filled with all element nodes from the Document tree.

```

Element root = doc.getRootElement();

```

```

List nodes = root.getChildren();

```

Step 4: Outer loop: get next node from the node list.

```

-- Get the key from the node.

```

```

outer:while(iterator.hasNext()){

```

```

    Element element = (Element)(iterator.next());

```

```

    String key = element.getChildText("key");

```

```

...
}

-- Inner loop: get the next element of input vector
inner:for(int i=0; i<paras.size(); i++){

String para = (String)paras.elementAt(i);

...

    -- if the input value is empty, continue the inner loop.

    if(para == null || para.length() == 0 || para.equals(""))

        continue inner;

    -- get the corresponding value from the node.

    Example of search by author:

    if(typeValue.equals("author")){

        nodeValue = element.getChildText("author");

        if(nodeValue == null)

            nodeValue = element.getChildText("editor");

    }

    -- if the value from the node is null, continue the outer loop.

    if(nodeValue == null){

        continue outer;

    }

    -- compare the input value with the corresponding value from the node

        with the constrains of the search type and the search option case.

    This part is complicated, not only because there are a lot of cases, but also

    because advanced search allows the user to search by more than one

```

author, and also by as many key words as he/she wish in the title. If the user searches by more than one author, this part checks the tree node satisfied all authors and put this node in the Hashtable named registry. If the user searches by many key words in the title, this part checks the tree node satisfied all key words and put this node in the Hashtable registry and put other nodes satisfied partial key words in the Hashtable named titleOr.

```
boolean allTrueFlag = false;

boolean flagForOrTitle = false;

...

boolean flag = false;

if(typeValue.equals("author")){

    Vector andVector = Helper.seperate(para, " and ");

    boolean[] andFlag = new boolean[andVector.size()];

    for(int j = 0; j<andVector.size(); j++){

        String filledAuthor = (String)andVector.elementAt(j);

        andFlag[j]=compare(request, filledAuthor,

        Helper.eleminateChar(nodeValue), typeValue);

    }

    for(int t = 0; t<andVector.size(); t++){

        if( andFlag[t] == false)

            break;

        if( t == (andVector.size()-1)){
```

```

        flag = true;

        allTrueFlag = true;

    }

}

}

else if(typeValue.equals("title")){

allTrueFlag = false;

Vector titleVector = Helper.vectorize(para, " ");

boolean[] titleFlag = new boolean[titleVector.size()];

for(int j = 0; j<titleVector.size(); j++){

    String filledTitle = (String)titleVector.elementAt(j);

    titleFlag[j]=compare(request, filledTitle,

        Helper.eleminateChar(nodeValue), typeValue);

}

for(int t = 0; t<titleVector.size(); t++){

    if( titleFlag[t] == false)

        break;

    if( t == (titleVector.size()-1)){

        allTrueFlag = true;

        flag = true;

    }

}

for(int t = 0; t<titleVector.size(); t++){

```

```

        if( titleFlag[t] == true){

            flag = true;

            flagForOrTitle = true;

            break;

        }

    }

}

-- if the compare result is false, continue the outer loop.

if(flag == false){

    flagForOrTitle = false;

    continue outter;

}

-- after inner loop, indicates the node matches the input criteria, then add the key
and the node into a Hashtable representing the initial search result.

if( allTrueFlag == true)

    registry.put(key, element);

if(flagForOrTitle == true)

    titleOr.put(key, element);

-- finishing the outer loop, all satisfied nodes are added to the initial search result
Hashtable. Remove one copy in Hashtable titleOr if nodes duplicated in two
Hashtable.

session.setAttribute("titleOr", titleOr);

session.setAttribute("result", registry);

```

```

Enumeration v1 = titleOr.keys();

while(v1.hasMoreElements()){

String s1 =(String)v1.nextElement();

Enumeration v2 = registry.keys();

while(v2.hasMoreElements() ){

    String s2 =(String)v2.nextElement();

    if(s1.equals(s2)){

        titleOr.remove(s1);

        break;

    }
}

```

### **3.2.1.6 Modify the search result**

After searching, the initial search results are printed in a table with a check box for each record. Each record will be represented with the authors and the title. The user can check the check box and list the checked records. In this way, the user can choose the records they are interested in.

The initial search results are stored in a Hashtable with the key and the element node pairs. This Hashtable in turn is stored in the session object. The initial search results are printed in an html form with check boxes. Each check box has a name and value. The name is given in the form "check" + counter, and the value is the actual key. An example is shown here:

```

<form method = "post" action="/bib/SaveResult">
<input type="checkbox" name="check1" value="glas94" >
<input type="checkbox" name="check2" value="glas91" >
<input type="checkbox" name="check3" value="glas98" >
<input type="submit" name="submit" value="Check and Save Record">
</form>

```

This form will be sent to the `SaveResult` servlet. The `SaveResult` servlet will get the initial search results from session, and get checked records which are keys by the request object. The checked record elements will be found from the initial search results by keys, and will be printed. The process is mainly done in the `modify` function. The code snippet is:

```

registry = (Hashtable)session.getAttribute("result");
for(int i = 1; i <= registry.size(); i++){
    counter ++;
    String key = request.getParameter("check"+counter);
    if(key != null){
        Element element = (Element)registry.get(key);
        String author =
            Helper.eleminateChar(element.getChildText("author"));
        if(author == null)
            author =
                Helper.eleminateChar(element.getChildText("editor"));
        String title =
            Helper.eleminateChar(element.getChildText("title"));
    }
}

```

```

        print(out, author, title, key, counter);
    }
}

```

The user can modify the search results further in the `SaveResult` servlet. The mechanism is the same.

### 3.2.1.7 Display the search result on the screen

When the user clicks on “Display on Screen” button, the form will be sent to the `SaveResult` servlet. The `SaveResult` servlet will get the checked record keys by the request object, find the element nodes from the initial result `Hashtable` by keys. Convert the nodes into record string by calling the `getRecord` function of the `Helper` class. This will display the record string of BibTeX format in the browser. The following snippet of code shows how to convert a node to a record string:

```

public static String getRecord(Element e){
    String oneRecord = "@";
    String type = e.getName();
    oneRecord = oneRecord + type + "{";
    List tagAndValue = e.getChildren();
    int count = 0;
    ListIterator iter = tagAndValue.listIterator();
    while(iter.hasNext()){
        count++;
    }
}

```

```

        Element tag = (Element)(iter.next());
        String tagName = tag.getName();
        List valueList = tag.getContent();
        ListIterator rator = valueList.listIterator();
        String tagValue = (String)(rator.next());
        if(tagName.equals("key"))
            oneRecord = oneRecord + tagValue + ", ";
        else{
            if(isDigOrMonth(tagValue))
                oneRecord = oneRecord + tagName + "=" + tagValue;
            else
                oneRecord = oneRecord + tagName + "=\"" + tagValue + "\"";

            if(count == tagAndValue.size())
                oneRecord += "}";
            else
                oneRecord += ", ";
        }
    }
    return oneRecord;
}

```

### 3.2.1.8 Saving the search result on local disk

When the user clicks on “Save to Local Disk” button, the form will be sent to the SaveResult servlet. The SaveResult servlet will get the checked record keys by

the request object, and store the keys in a keys vector. It will then put the vector on the session object and send the request to the ToDisk servlet.

In the ToDisk servlet, get the initial result and keys vector from the session object. Get the elements from the initial result Hashtable by keys. Then it will convert the elements to a string and save it to disk.

To prompt a save file dialog box, the header must be set by the response object. The following code shows the process:

```
OutputStream out = response.getOutputStream();
response.setContentType("application/octet-stream");
response.setHeader("Content-Disposition", "attachment; filename=" +
                    "bibtext" + ".");
response.setHeader("Cache-Control", "no-cache");
..... //To get the record string named content
response.setContentLength(content.length());
out.write(content.getBytes());
out.flush();
```

When this response is sent to the browser, a save file dialog box will pop up to prompt the user to download and save the file.

### 3.2.1.9 Send the search result by email

When the user inputs an email address and clicks the “E-mail result to” button, the search result will be sent to the email address. The mailhost and sender e-mail are read from the configuration file. We use the Java Mail API (21) to perform this function. Sending an email message involves getting a session, creating and filling a message, and sending it. We specify the SMTP server by setting the “mail.smtp.host” property for the Properties object passed when getting the Session. The following snippet of code shows the process:

```
String host = ...;
String from = ...;
String to = ...;

// Get system properties
Properties props = System.getProperties();

// Setup mail server
props.put("mail.smtp.host", host);

// Get session
Session session = Session.getDefaultInstance(props, null);

// Define message
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
```

```
message.addRecipient(Message.RecipientType.TO,  
    new InternetAddress(to));  
message.setSubject("bibtex mail");  
message.setText(content);        //content is the result string  
  
// Send message  
Transport.send(message);
```

The code is placed in a try-catch block, as setting up the message and sending it can throw exceptions.

### **3.2.2 Add record**

The privileged user can add a record through the Internet or manually. I only briefly introduce functions for each part, detail explanations is reported in Aimin Han's major report.

#### **3.2.2.1 Login**

The user has to be authenticated for security reasons. The privileged user and password are stored in the users.txt file. In the CheckLogin servlet, get the input user name and password by the request object, and then check them in the validate function

### **3.2.2.2 Check key for input record**

For the Bibtex records, the key can't be empty and must be unique. When a user fills the input form and clicks the submit button, the form will be sent to the `Confirm` servlet to check key's property.

### **3.2.2.3 Check field format**

Some fields such as chapter, year, number, and volume must be input as digital numbers. When the input values for these fields are obtained, call the `isDigital` function of the `Helper` class to check input format.

### **3.2.2.4 Display error message**

If the user input key is empty or is not unique, or some field values are not in the correct format, an error message will be displayed, and the user will be sent to the previous page – the add record page.

### **3.2.2.5 Save record**

In the `Save` servlet, get the added record from the `session` object:

```
String record = (String)session.getAttribute("record");
```

Read the configuration file to get the output Bibtex file name and append the record to this file. Also write the additional information such the time of addition and the user who added the record.

#### **3.2.2.6 Logout**

In the Logout servlet, invalidate the session object.

#### **3.2.2.7 Add records manually**

The privileged user can add records to the Bibtex file manually. The added records will be searchable after a time interval which is the `StartUp` thread wait time.

## **4 Discussion and future work**

### **4.1 Search efficiency**

For searching the bib database, the search efficiency is important. There are three aspects involved the searching efficiency: the time of converting the bib file to an XML file, the time of parsing the generated XML file, the actual searching time in server and the request and response time through the network.

When the server starts up, the bib file will be converted to an XML file, and the generated XML file will be parsed to a JDOM document tree. This also happens in a time interval of thread waiting and when a record is added through the Internet. This process won't affect the search time significantly. Using a computer with Pentium 266 processor and 256M SDRAM for a bib file of size 1060KB, the average conversion time is 5658 ms, and the average parsing time is 7821ms.

When a user makes a search request, the server searches the information from the JDOM document tree. Since the tree stays in the server memory, the search process is very fast compared to fetching data from a secondary storage. For the general search, if the search type is booktitle or journal, only the conference node or article nodes will be searched. For searching by other type or the advanced search, all the nodes will be scanned. The worst case is when all fields are filled in the advanced search. Using a computer with

Pentium 266 processor and 256M SDRAM for the server, for a bib file of size 1060KB, the average search time for advanced search is 1521ms when all the fields are filled.

The data transport time from the server to the client through the Internet depends on the bandwidth of the network. It is outside the scope of this project, so we don't discuss it here.

## **4.2 Scalability**

Scalability is an important requirement of software development. One aspect is about the number of users. When the number of users is getting large, the concurrent access to the data is protected. This application can be accessed by large number of users.

Another aspect about the scalability is of the database. The bib database is the bib file in the server side. This is going to be converted to an XML file and be parsed to a JDOM document tree that stays in the server random memory. This raises the question: can the memory accommodate the whole data if the database becomes huge?

The reason we use the XML data presentation in this project instead of using a relational database management system is discussed in the design part. The tag extensibility of the bib file entry excludes the use of a relational database. For a large bib database, the solution could be to classify the bib files according to their category. In this way, the huge bib file is split to parts. When searching information from the database, the application will process them one by one. This is going to be future work.

## **4.3 Portability**

As we discussed in the design part, the combination of the Java language and the plain text file format of bib file and the XML file makes the application portable on different platform. The variables that the application uses are stored in a configuration file. The application can read them in at run time. Changing the variable values in configuration file doesn't need to change the application code. This increases the portability of the application.

Using the Java servlet technology, we can specify the variables in the web.xml file of the web application. This is an alternative way to configure the application. However if we change any variable in the web.xml file, we have to restart the server. In this application, we use a configuration file to store the variable information instead. For example, we can change the email host in the configuration file when the application is running, and the change will be in effect immediately.

## **4.4 Future Work**

When we finished this project, we found there are some aspects we could improve.

1. In advanced search, implement the support of AND, OR, NOT for title key words.
2. When the bib database is too large, split the bib file into parts, and manage bib files by category. In searching process, process the bib files one by one according to the searching category. Improve the application to enable it to perform this task.

3. Enable privileged user to modify entries in bib database.
4. The tag extensibility is a feature of Bibtex record. Enable the privileged user to add more tags and values for a record when adding a record online.

## **5 Conclusions**

The implementation of BibT<sub>E</sub>X server shows that it is a good choice to use the Java servlet technology, to use the XML presentation of data and to use the JDOM API. High search efficiency is achieved. This application is totally portable, independent of platforms.

The functionality of this application was accomplished according to the requirements.

## 6 References

1. Leslie Lamport, "LaTeX: A Document Preparation System", Addison-Wesley, 1986.
2. Henning Schulzrinne, Columbia University, "Network Bibliography",  
<http://www.cs.columbia.edu/~hgs/netbib/>
3. Erik Wilde, "BibTeXML: An XML Representation of BibTeX Data", Internationales Congress Centrum (ICC), Berlin, German, 21-25 May 2001.
4. Robin Cover, "BiblioML - XML for UNIMARC Bibliographic Records",  
<http://www.oasis-open.org/cover/biblioML.html>.
5. "BiblioML Project", <http://www.culture.fr/BiblioML/en/index.html>.
6. Jonas Björnerstedt, "BibEdit Version 1.1 beta",  
<http://www.iui.se/staff/jonasb/bibedit/readme.pdf>.
7. Nelson H. F. Beebe, "BIBSEARCH 1 Version 1.01",  
<http://www.math.utah.edu/~beebe/software/bibsearch/bibsearch.html>
8. Gerd Herzog and Clemens Huwig, "LIDOS: Literature Information and Documentation System", <http://www.dfki.uni-sb.de/imedia/lidos>.
9. Marty Hall, "Core Servlets and Java Server Pages", Prentice Hall PT, 2000.
10. Karl Avedal, Danny Ayers, et al., "Professional JSP", Wrox Press, 2000.
11. Robin Cover, "BiblioML - XML for UNIMARC Bibliographic Records",  
<http://www.oasis-open.org/cover/biblioML.html>
12. Brett McLaughlin, "Java and XML", O'Reilly, 2000.
13. "Extensible Markup Language (XML) 1.0 (Second Edition)", <http://www.xml.org>.
14. Jason Hunter and Brett McLaughlin, "Easy Java/XML integration with JDOM, Part 1: Learn about a new open source API for working with XML",  
<http://www.javaworld.com/javaworld/jw-05-2000/jw-0518-jdom.html>.
15. JDOM specifications, <http://www.jdom.org>.
16. James W. Cooper, "Java Design Patterns: A Tutorial", Addison-Wesley, 2000.
17. Cay S. Horstmann and Gary Cornell, "Core Java, Volume I – Fundamentals", Sun, 1999.

18. "Apache HTTP Server Version 1.3". <http://httpd.apache.org/docs>
19. "Netcraft Web Server Survey". <http://www.netcraft.com/survey>
20. "Tomcat Documentation". <http://jakarta.apache.org/tomcat/tomcat-3.3doc/index.html>
21. JAVAMAIL™ API. <http://java.sun.com/products/javamail>

## **7 Appendix**

### **7.1 Installation Guide**

Windows NT/2000 is needed for installing this system.

#### **7.1.1 Install and Configure Apache Web Server 1.3.22**

- Double click `apache_1.3.22-win32-x86.exe` file. It will install apache automatically.
- Copy `mod_jk.dll` to {apache installation directory}\apache\modules.
- Edit `http.conf` file in {apache installation directory}\apache\conf: Add a the following line to the end of this file:  
  
`Include {tomcat3.3 installation directory}\conf\auto\mod_jk.conf`

#### **7.1.2 Install and Configure Tomcat 3.3**

- Unzip `jakarta-tomcat-3.3.zip` to a directory.
- Edit {tomcat3.3 installation directory}\conf\server.xml file: Change the “noRoot” option to “false” in the `<ApacheConfig noRoot= “true”>`
- Edit {tomcat3.3 installation directory}\conf\jk\workers.properties file: Change the value of `tomcat-home` to { tomcat3.3 installation directory}, and change the value of `java-home` to { jdk1.3.1 installation directory}.
- Add system environment variables:
  - Right click “my computer” icon in desktop. Choose “properties”. Click the “advanced” tab. Click “Environment variables...” button.

- Under “system variables”, click “new” button.
- Add TOMCAT\_HOME variable. Set value to {tomcat3.3 installation directory}.
- Add JAVA\_HOME variable. Set value to {jdk1.3.1 installation directory}
- In DOS window, go to {tomcat3.3 installation directory}\bin. Start tomcat by typing command: startup -jkconf. This doesn't really start tomcat. It is configuration only. After this, close this window.

### **7.1.3 Deploy bib web application**

Drop the bib.war file to {tomcat3.3 installation directory}\webapps.

### **7.1.4. Startup tomcat. Start apache**

Go to {tomcat3.3 installation directory}\bin and double click the “startup.bat” file to start tomcat. In the computer start menu, find the apache HTTP server in the programs and start the Apache server. In the browser, point the address to: <http://localhost/bib>

## 7.2 User Manual

### 1. Search

#### 1.1 General search

For general search, user can search by author, title, booktitle, after year, before year, and journal (Figure 13). User can cancel the search by clicking the "stop" button.

The screenshot shows a web browser window titled "General Search - Microsoft Internet Explorer". The address bar displays "http://localhost:8080/bib/search.html". The main content area has the heading "Bib Search Engine". Below the heading, there is a search form. The "Search by" dropdown menu is open, showing options: "author" (selected), "title", "booktitle", "after year", "before year", and "journal". To the right of the dropdown is a text input field. Below the input field are two checkboxes: "Match words" and "Match whole field". There are two buttons: "Search" and "stop". Below these buttons is a link labeled "Advanced Search". At the bottom of the page, there is a list of instructions for the search engine.

Search by  ☐ Match words ☐ Match whole field

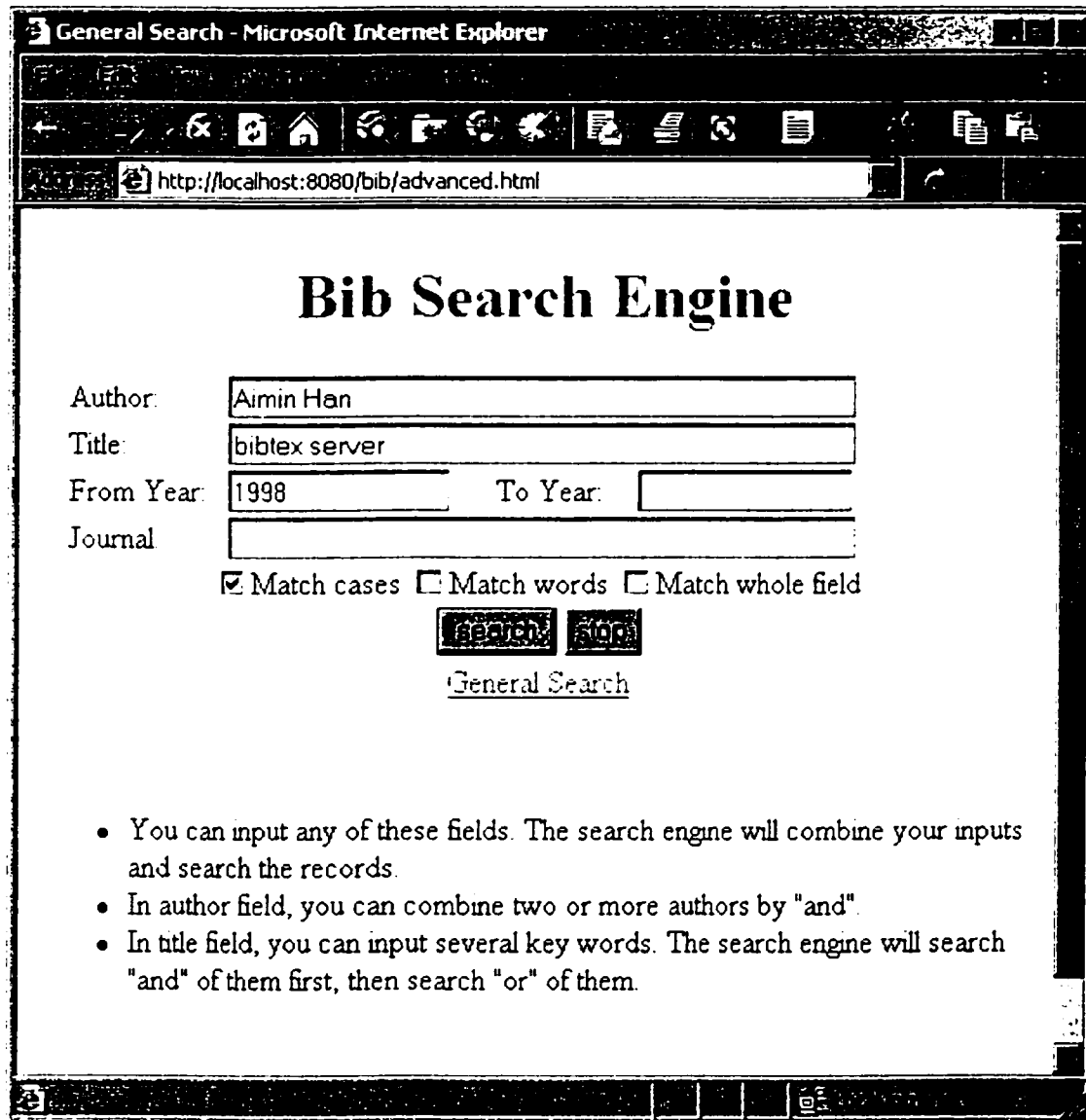
[Advanced Search](#)

- Search by author: in this general search, you only can search by one author. The input key word could be part or all of the author's or editor's name. If you want to search by combining authors, please use the advanced search.
- Search by title: you can type the keyword. the search engine will search the record that its title contains the key word you input. The title could be the title of a book, a article, and so on. If you want to search by more key words, please use the advanced search.
- Search by booktitle: you can input a keyword. The key word could be part or all of the title of the conference book.
- Search by journal: input one key word. The search engine will search the journal name. The key word you input could be part or all of a journal name.

Figure 13: General search

## 1.2 Advanced search

For advanced search, user can input any fields in the form (Figure 14). The search engine will search information by combining the criteria input by the user. The user does not have to input information for all fields.



The screenshot shows a Microsoft Internet Explorer window titled "General Search - Microsoft Internet Explorer". The address bar displays "http://localhost:8080/bib/advanced.html". The main content area features the heading "Bib Search Engine" in a large, bold font. Below the heading is a search form with the following fields and options:

- Author:
- Title:
- From Year:  To Year:
- Journal:
- ☒ Match cases ☐ Match words ☐ Match whole field
- 
- [General Search](#)

Below the form, there is a list of instructions:

- You can input any of these fields. The search engine will combine your inputs and search the records.
- In author field, you can combine two or more authors by "and".
- In title field, you can input several key words. The search engine will search "and" of them first, then search "or" of them.

Figure 14: Advanced search

## 1.3 Modify research results

The search result will be printed with author or editor and item title. If the user is interested in some of them, the search result can be modified by checking the candidate records and click the “Check and Save Record” button (Figure 15).

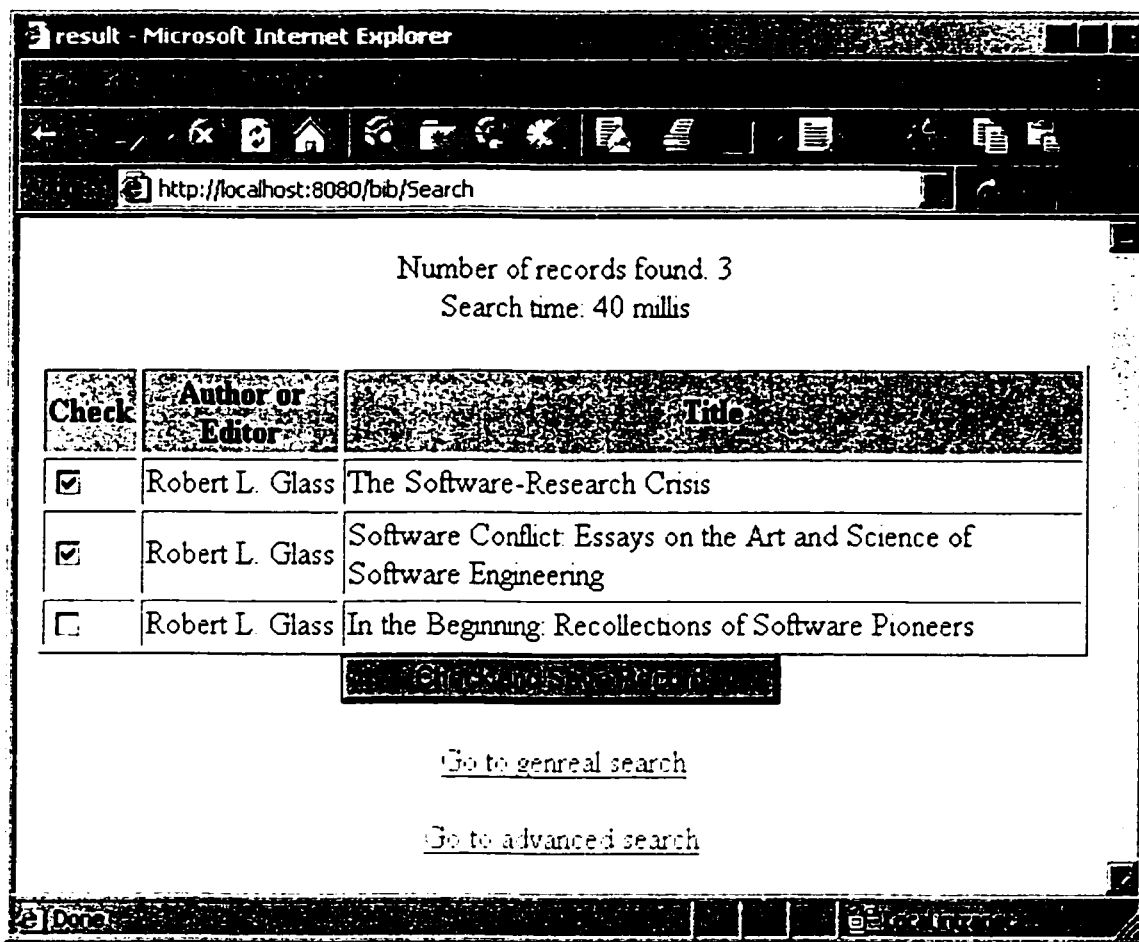


Figure 15: Search result

#### 1.4 Final result and display options

At this point, the user still can modify the result. If the user doesn't need some records, uncheck the record that is going to be filtered, and click “Save Checked Record” button (Figure 16).

There are three options for displaying the final result. Clicking “Display on Screen”, the record details will be displayed in browser to enable user to copy and paste (Figure 17). Clicking “Save to Disk” will prompt user to save the search result to local disk (Figure 18).

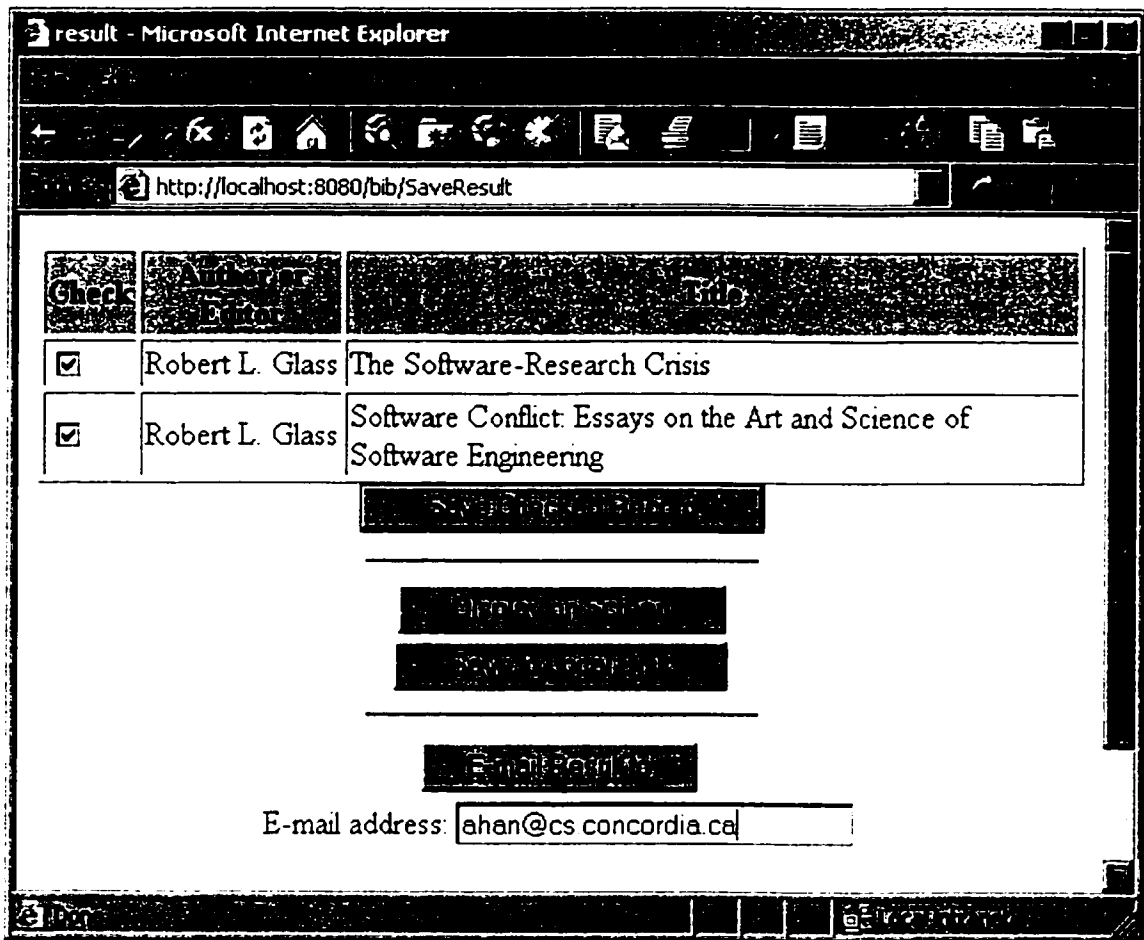


Figure 16: Final result and display options

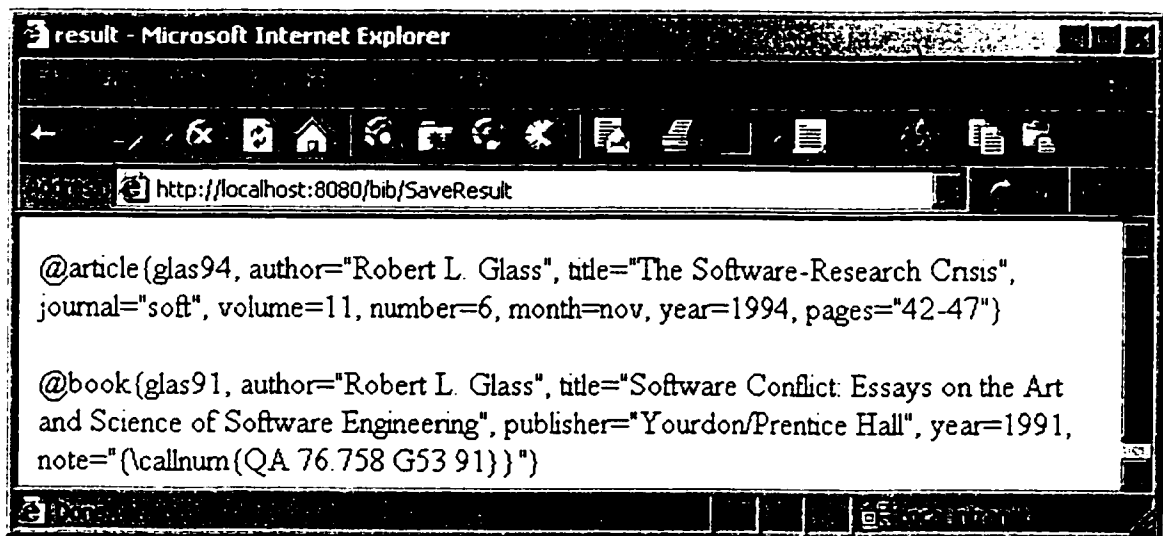


Figure 17: Display on screen

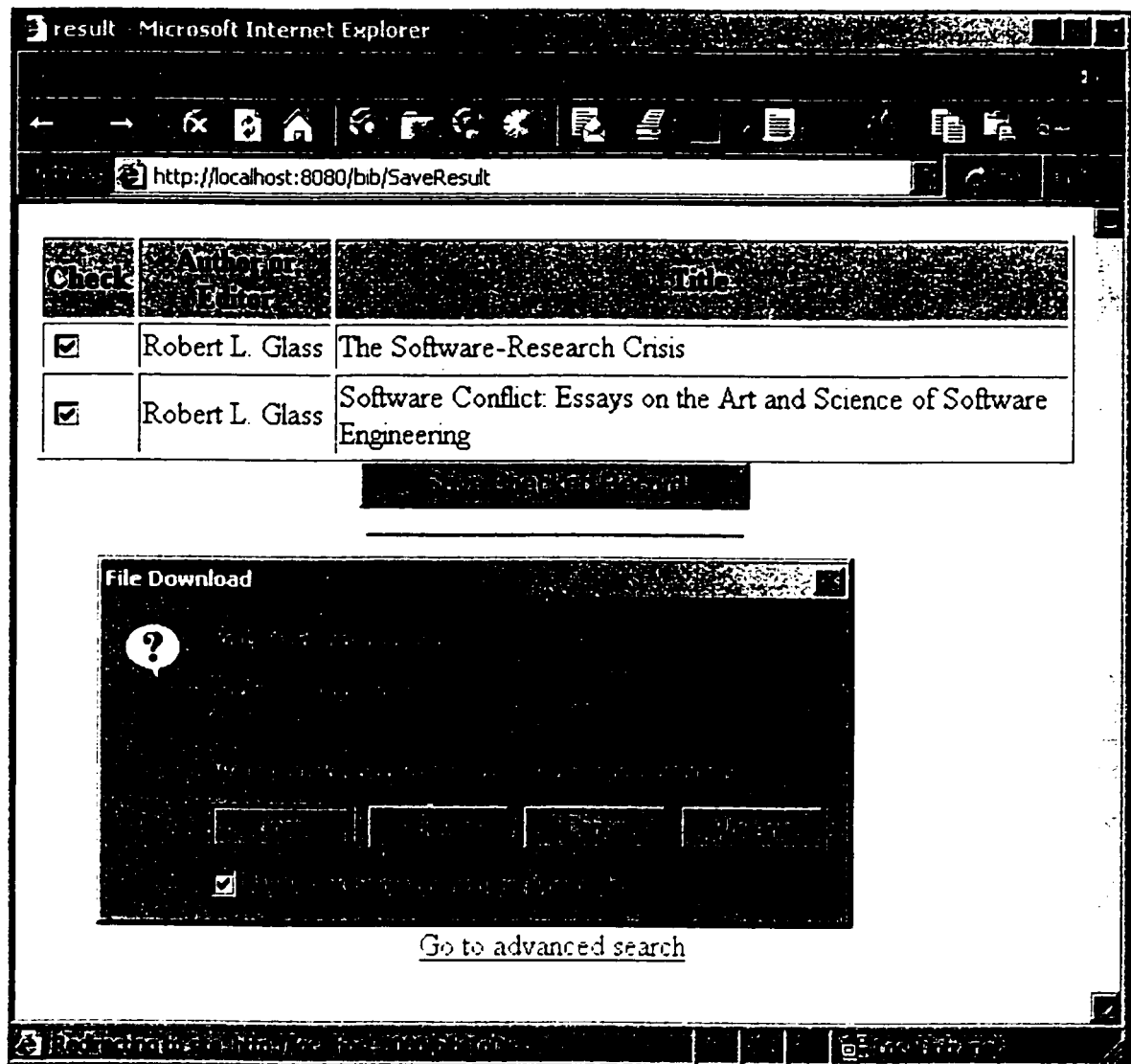


Figure 18: Save the result to local disk

## 2. Add record (administrator only)

### 2.1 login

For security, only authorized user can add record. The login page prompts the user to input user name and password (Figure 19).

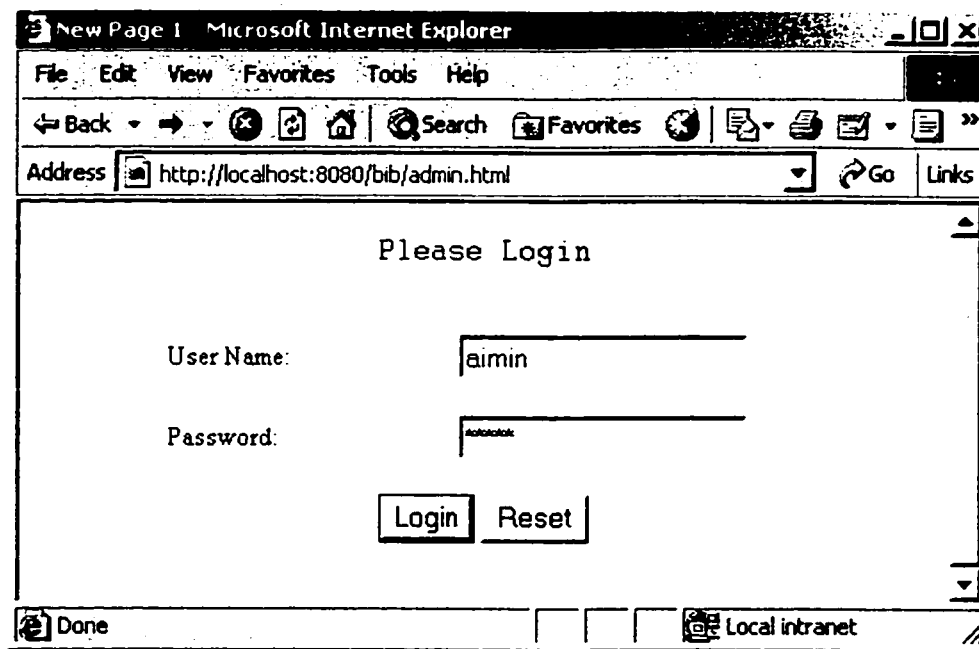


Figure 19: Log in

## 2.2 Add record

If login step is successful, the user can add record (Figure 20). Any error will be showed in the up-right corner of the table. To add a record, the key field has to be input. Also the key has to be unique. If a duplicate key is typed, the error message will show up, and all the similar keys will be printed to give the user information.

The fields labeled with \* can't be input character type. They require input as number format. The month field has to be input in appropriate format. The instructions are in the right side of the table. The error message will appear if the format is not correct.

**Add Record - Microsoft Internet Explorer**

C:\Documents and Settings\Aimin Han\Desktop\Add Record.htm

Entry Type:	<input type="text" value="article"/>	Entry Key:	<input type="text" value="ahan12"/>	This key existed. Please choose another one. HINT: The following keys already existed ahan ahan12	
Address:	<input type="text"/>				
Title:	<input type="text"/>				
Author:	<input type="text" value="Aimin Han"/>				
Booktitle:	<input type="text"/>				
Chapter:	<input type="text" value="12"/>	*			
Crossref:	<input type="text"/>				
Edition:	<input type="text"/>				
Editor:	<input type="text"/>				
Howpublished:	<input type="text"/>				
Institution:	<input type="text"/>				
Journal:	<input type="text"/>				
Kw:	<input type="text"/>				
Year:	<input type="text"/>	*	Month:	<input type="text"/>	**
Note:	<input type="text"/>				
Volume:	<input type="text"/>	*	Number:	<input type="text"/>	*
Organization:	<input type="text"/>				
Pages:	<input type="text"/>				
Publisher:	<input type="text"/>				

\* These fields must be numbers  
 \*\* Month is the form of abbreviation: jan, feb, ...

Figure 20: Add record

### 2.3 Confirm and Save

Before saving the record, there is a confirmation page, which enables user to make sure the record is correct (Figure 21). If the record is not correct, the user can go back to change it. If there is no error in the record, the user can click "save" link to save the record. After saving, a confirmation will be showed (Figure 22).

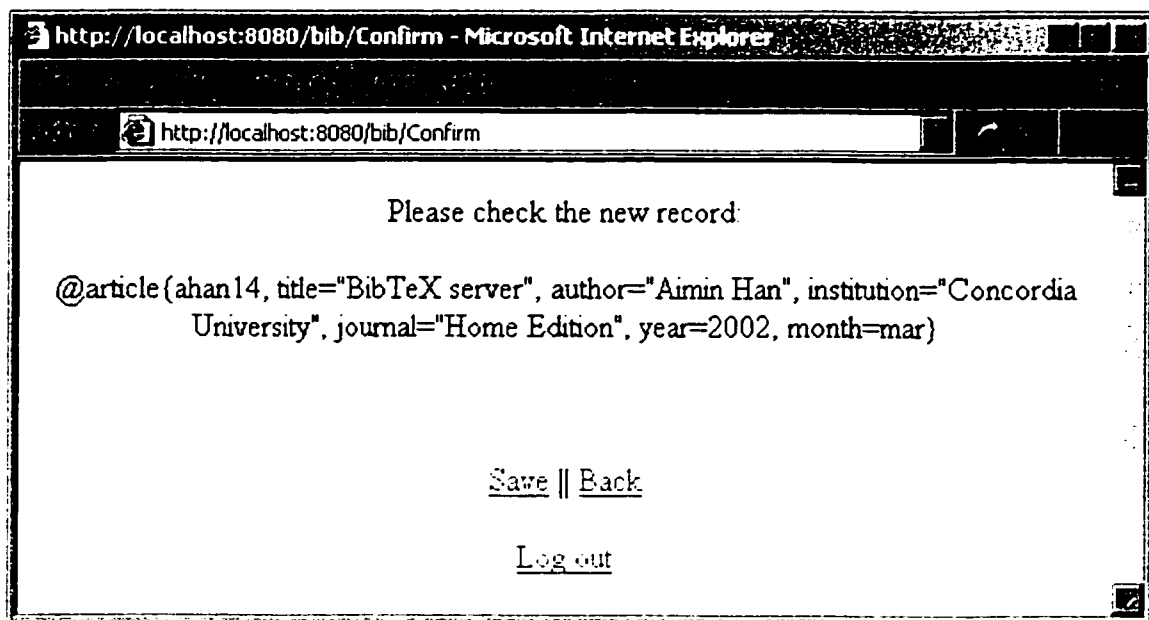


Figure 21: Confirm the record

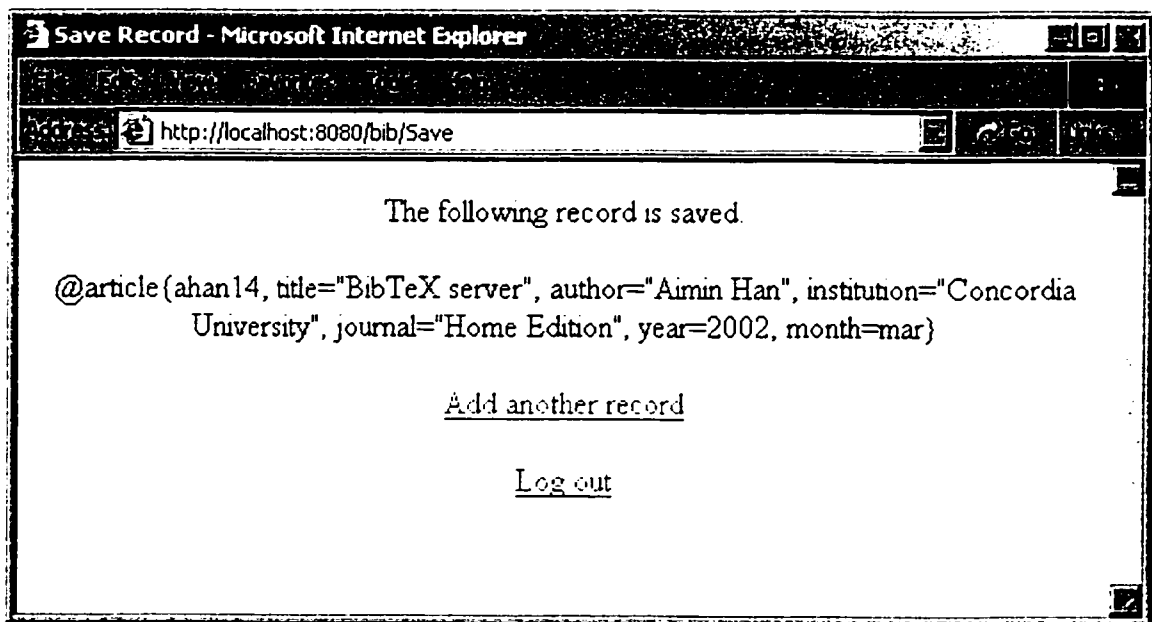


Figure 22: Confirmation of saving a record

## **7.3 Source Code**

### **7.3.1 General search source code**

### **7.3.2 Advanced search source code**

```

package ca.concordia.cs.bibsearch;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

import java.lang.Character;

/**
 * This class processes general search. It gets input from the user, and search
 * the information from the JDOM document fetched from class StartUp. Then put
 * the initial search result on session object. The initial search result will
 * be printed out in the browser and let the user to choose the interested ones
 * which will be passed to {@link ca.concordia.cs.bibsearch.SaveResult
 * SaveResult} class.
 *
 * Title: Bibtex search /p>p>
 *
 * Description: Search bibtex information from bibtex file /p>p>
 *
 * Copyright: Copyright (c) 2002 /p>p>
 *
 * Company: Department of Computer Science, Concordia University /p>
 *
 * @author      Man Bao (mbao@cs.concordia.ca), Almin Han (ahandes@concordia.ca)
 * @created      September 11, 2002
 * @version      1.0
 */
public class Search extends HttpServlet {
    /**
     * Get the user input and search information from JDOM Document object
     * fetched from {@link ca.concordia.cs.bibsearch.StartUp StartUp} class. In
     * the JDOM Document object, each record is represented as an element node.
     * If the user is searching by bootitle, it will search from the sentence
     * elements. If the user is searching by journal, it will search from article
     * element. Otherwise, it will search all elements. According to the options
     * "match case", "match word", and "match whole field", there will be five
     * cases. This method will search the information corresponding to the case.
     * The initial search result will be stored in a hashtable which holds the
     * key and the element paris. This hashtable will be put on session so that
     * next page can get it. The initial result will print out in a table with
     * author or editor and title.
     *
     * @param request
     * @param response
     * @throws IOException
     * @throws ServletException
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        Hashtable registry = new Hashtable();
        //hold key and element

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(true);

        String searchType = (request.getParameter("search_type")).trim();
        String searchName = (request.getParameter("name")).trim();
        boolean isDigit = true;
        if (searchType.equals("after year") || searchType.equals("before year")) {
            if (!Helper.isDigital(searchName)) {

```

```

        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + "result" + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"white\">");
        out.println("<body>");
        out.println("<p align=\"center\">Input format error, you must input number in text"
            + " field of after year or before year.</p>");
        out.println("<p align=\"center\"><a href=\"index.html\">Go to general search page</p>");
        out.println("<p align=\"center\"><a href=\"advanced.html\">Go to advanced search page</p>");
        out.println("</body>");
        out.println("</html>");
        isDigit = false;
        return;
    }

    boolean matchCase = (request.getParameter("case") != null);
    boolean matchWord = (request.getParameter("word") != null);
    boolean matchField = (request.getParameter("field") != null);

    int searchCondition;
    if ((matchCase == false) && (matchWord == false) && (matchField == false))
        searchCondition = 0;
    else if ((matchCase == true) && (matchWord == false) && (matchField == false))
        searchCondition = 1;
    else if ((matchCase == false) && (matchWord == true) && (matchField == false))
        searchCondition = 2;
    else if ((matchCase == false) && (matchWord == false) && (matchField == true)
        || (matchCase == false) && (matchWord == true) && (matchField == true))
        searchCondition = 3;
    else if ((matchCase == true) && (matchWord == true) && (matchField == false))
        searchCondition = 4;
    else
        searchCondition = 5;

    if (searchName.equals("")) {
        if (searchName.length() != 0) {
            searchName = null;
            response.sendRedirect("index.html");
            return;
        }
    }

    String search = request.getParameter("search");
    String stop = request.getParameter("stop");

    if (search == null && stop.equals("stop")) {
        response.sendRedirect("index.html");
        return;
    }

    //get document
    Document doc = StartUp.getDocument();

    //search
    Element root = doc.getRootElement();
    List nodes = null;
    if (searchType.equals("booktitle")) {
        nodes = root.getChildren("book");
        List nodesTwo = root.getChildren("conference");
        nodes.addAll(nodesTwo);
    }
    else if (searchType.equals("journal")) {
        nodes = root.getChildren("article");
    }
    else {

```

```

nodes = root.getChildren();
}

ListIterator iter = nodes.listIterator();
while (iter.hasNext()) {
    Element element = (Element) (iter.next());
    String key = element.getChildText("key");

    if (searchType.equals("after year")) {
        String typeString = element.getChildText("year");

        //Go to error page if number format error
        if (isDigit == true && typeString != null &&
            ((Integer.parseInt(typeString)) >= Integer.parseInt(searchName))) {
            registry.put(key, element);
        }
    }
    else if (searchType.equals("before year")) {
        String typeString = element.getChildText("year");

        //Go to error page if number format error
        if (isDigit == true && typeString != null &&
            ((Integer.parseInt(typeString)) <= Integer.parseInt(searchName))) {
            registry.put(key, element);
        }
    }
    else {
        switch (searchCondition) {
            case 0:
                if (searchType.equals("author")) {
                    String author = Helper.eleminateChar(element.getChildText("author"));
                    if (author == null) {
                        author = Helper.eleminateChar(element.getChildText("editor"));
                    }

                    if (author != null &&
                        (author.toLowerCase().indexOf(searchName.toLowerCase()) >= 0)) {
                        registry.put(key, element);
                    }
                }
                else {
                    String typeString = Helper.eleminateChar(element.getChildText(searchType));

                    if (typeString != null &&
                        (typeString.toLowerCase().indexOf(searchName.toLowerCase()) >= 0)) {
                        registry.put(key, element);
                    }
                }
                break;
            case 1:
                if (searchType.equals("author")) {
                    String author = Helper.eleminateChar(element.getChildText("author"));
                    if (author == null) {
                        author = Helper.eleminateChar(element.getChildText("editor"));
                    }

                    if ((author != null && author.indexOf(searchName) >= 0)) {
                        registry.put(key, element);
                    }
                }
                else {
                    String typeString = Helper.eleminateChar(element.getChildText(searchType));

                    if (typeString != null && typeString.indexOf(searchName) >= 0) {
                        registry.put(key, element);
                    }
                }
                break;
            case 2:

```

```

if (searchType.equals("author")) {
    //eliminate \ ' " { }
    String author = Helper.eleminateChar (element.getChildText ("author"));
    if (author == null) {
        author = Helper.eleminateChar (element.getChildText ("editor"));
    }
    if (author != null) {
        author = author.toLowerCase();
    }

    //seperate names
    Vector authorVec = Helper.seperate (author, " and ");
    if (authorVec.contains (searchName.toLowerCase())) {
        registry.put (key, element);
    }
}
else {
    String typeString = Helper.eleminateChar (element.getChildText (searchType));
    if (typeString != null) {
        typeString = typeString.toLowerCase();
    }
    Vector vec = Helper.vectorize (typeString, " ");
    if (vec.contains (searchName.toLowerCase())) {
        registry.put (key, element);
    }
}
break;
case 3:
if (searchType.equals("author")) {
    String author = Helper.eleminateChar (element.getChildText ("author"));
    if (author == null) {
        author = Helper.eleminateChar (element.getChildText ("editor"));
    }

    if (author != null && author.toLowerCase().equals (searchName.toLowerCase())) {
        registry.put (key, element);
    }
}
else {
    String typeString = Helper.eleminateChar (element.getChildText (searchType));
    if (typeString != null && typeString.toLowerCase().equals (searchName.toLowerCase())) {
        registry.put (key, element);
    }
}
break;
case 4:
//same as case 2 without toLowerCase()
if (searchType.equals("author")) {
    //eliminate \ ' " { }
    String author = Helper.eleminateChar (element.getChildText ("author"));
    if (author == null) {
        author = Helper.eleminateChar (element.getChildText ("editor"));
    }

    //seperate names
    Vector authorVec = Helper.seperate (author, " and ");
    if (authorVec.contains (searchName)) {
        registry.put (key, element);
    }
}
else {
    String typeString = Helper.eleminateChar (element.getChildText (searchType));
    Vector vec = Helper.vectorize (typeString, " ");
    if (vec.contains (searchName)) {
        registry.put (key, element);
    }
}
break;

```

```

        case 5:
            if (searchType.equals("author")) {
                String author = Helper.eleminateChar (element.getChildText ("author" ));
                if (author == null) {
                    author = Helper.eleminateChar (element.getChildText ("editor" ));
                }
                if (author != null && author.equals (searchName)) {
                    registry.put (key, element);
                }
            }
            else {
                String typeString = Helper.eleminateChar (element.getChildText (searchType));
                if (typeString != null && typeString.equals (searchName)) {
                    registry.put (key, element);
                }
            }
            break;
        default:
            ;
    }

    session.setAttribute ("result", registry);
    session.setAttribute ("titleOr", new Hashtable ());

    out.println ("<html>");
    out.println ("<head>");
    out.println ("<title>" + "result" + "</title>");
    out.println ("</head>");
    out.println ("<body bgcolor=\"white\">");
    out.println ("<body>");

    if (registry.size () != 0) {
        out.println ("<p align=\"center\">Number of records found: " +
            registry.size () + "<br>");
        out.println ("<form method = \"post\" action=\"/bib/SaveResult \">");
        out.println ("<table align=\"center\" border=\"1\">");
        out.println ("<tr bgcolor=\"#FFD000\">");
        out.println ("<th>Check</th><th>Author or Editor</th><th>Title</th></tr>");

        print (registry, out);

        out.println ("</table>");
        out.println ("<center>");
        out.println ("<input type=\"submit\" name=\"submit\" "
            + " value=\"Check and Save Record\">");
        out.println ("</center>");
        out.println ("</form>");
    }
    else {
        out.println ("<p align=\"center\">No record matches the search criterion.</p>");
    }

    out.println ("<p align=\"center\"><a href=\"/index.html\"> "
        + "Go to genreal search</a></p>");
    out.println ("<p align=\"center\"><a href=\"/advanced.html\"> "
        + "Go to advanced search</a></p>");
    out.println ("</body>");
    out.println ("</html>");
}

/**
 * This method will print out the initial result in a table with author or
 * editor and title.
 *
 * @param registry hashtable holding the key and the element node pairs.
 * @param out
 */
public void print (Hashtable registry, PrintWriter out) {

```

```

List keys = new ArrayList (registry.keySet ());
ListIterator iter = keys.listIterator ();
int counter = 0;
while (iter.hasNext ()) {
    counter++;
    String key = (String) iter.next ();
    Element element = (Element) registry.get (key);
    String author = Helper.eleminateChar (element.getChildText ("author"));
    if (author == null) {
        author = Helper.eleminateChar (element.getChildText ("editor"));
    }
    String title = Helper.eleminateChar (element.getChildText ("title"));

    if (author == null) {
        author = " ";
    }
    if (title == null) {
        title = " ";
    }
    out.println ("tr" );
    out.println ("td<input type=\"checkbox\" name=\"check\" " + counter
        + "\" value=\"" + key + "\" >/td>" );
    out.println ("td" + author + "</td>");
    out.println ("td" + title + "</td>");
    out.println ("</tr>");
}
}

/**
 * Pass the <code>request</code> and <code>response</code> to <code>@link
 * #doPost (HttpServletRequest, HttpServletResponse) doPost</code> method.
 *
 * @param request
 * @param response
 * @throws IOException
 * @throws ServletException
 */
public void doGet (HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    doPost (request, response);
}

```

```

package ca.concordia.cs.bibsearch;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;

/**
 * This class processes advanced search. It gets input from the user, and search
 * the information from the JDOM document fetched from class StartUp. Then put
 * the initial search result on session object. The initial search result will
 * be printed out in the browser and let the user to choose the interested ones
 * which will be passed to class <code>SaveResult</code>.
 * <p>Title: Bibtex search</p>
 * <p>Description: Search bibtex information from bibtex file</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Department of Computer Science, Concordia University</p>
 * <p>Author: Man Bao (mbao@cs.concordia.ca)</p>
 * <p>Author: Amin Han (ahan@cs.concordia.ca)</p>
 * <p>Version 1.0</p>
 */

public class AdvancedSearch extends HttpServlet {
    /**
     * This method processes the advanced search. It gets the user input from
     * request object, combines the search
     * criteria and search from JDOM document fetched from <code>StartUp</code>
     * thread.
     * @param request
     * @param response
     * @throws IOException
     * @throws ServletException
     */
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        Hashtable registry = new Hashtable(); //hold key and element
        Hashtable titleOr = new Hashtable(); //hold key and element
        Vector paras = new Vector();
        Vector type = new Vector();

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession(true);

        String author = (request.getParameter("author")).trim();
        paras.addElement(author);
        type.addElement("author");
        String title = (request.getParameter("title")).trim();
        paras.addElement(title);
        type.addElement("title");
        String from = (request.getParameter("from")).trim();
        paras.addElement(from);
        type.addElement("from");
        String to = (request.getParameter("to")).trim();
        paras.addElement(to);
        type.addElement("to");
        boolean isDigit = true;
        String fromTo = from + to;
        if (!Helper.isDigital(fromTo)) {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>" + "result" + "</title>");
            out.println("</head>");
            out.println("<body bgcolor=\"white\">");

```

```

        out.println("<body>");
        out.println("<p align=\"center\">Input format error, you must input number "
            + "in text field of from year or to year.</p>");
        out.println("<p align=\"center\"><a href=\"index.html\">Go to general search</a><p>");
        out.println("<p align=\"center\"><a href=\"advanced.html\">Go to advanced search</a><p>");
        out.println("</body>");
        out.println("</html>");
        isDigit = false;
        return;
    }

    String journal = request.getParameter("journal").trim();
    paras.addElement(journal);
    type.addElement("journal");

    String path = getServletContext().getRealPath("/") + "files\\";

    if (author.equals("") || author.length() == 0 || author == null) {
        (title.equals("") || title.length() == 0 || title == null) {
            from.equals("") || from.length() == 0 || from == null) {
                to.equals("") || to.length() == 0 || to == null) {
                    journal.equals("") || journal.length() == 0 || journal == null) {
                        response.sendRedirect("advanced.html");
                        return;
                    }
                }
            }
        }

    String search = request.getParameter("search");
    String stop = request.getParameter("stop");

    if (search == null || stop.equals("stop")) {
        response.sendRedirect("advanced.html");
        return;
    }

    Document doc = StartUp.getDocument();
    //if generated xml file has problem, the doc will be null
    Element root = doc.getRootElement();
    List nodes = root.getChildren();
    ListIterator iterator = nodes.listIterator();
    while (iterator.hasNext()) {
        Element element = (Element) iterator.next();
        String key = element.getChildText("key");

        String elementAuthor = element.getChildText("author");
        if (elementAuthor == null)
            elementAuthor = element.getChildText("editor");
        String elementTitle = element.getChildText("title");
        String elementYear = element.getChildText("year");
        String elementJournal = element.getChildText("journal");

        boolean allTrueFlag = false;
        boolean flagForOrTitle = false;
        inner: for (int i=0; i<paras.size(); i++) {
            String para = (String) paras.elementAt(i);
            if (para == null || para.length() == 0 || para.equals(""))
                continue inner;
            else {
                String nodeValue = "";
                String typeValue = (String) type.elementAt(i);
                if (typeValue.equals("from") || typeValue.equals("to")) {
                    nodeValue = element.getChildText("year");
                }
                else {
                    if (typeValue.equals("author")) {
                        nodeValue = element.getChildText("author");
                        if (nodeValue == null)
                            nodeValue = element.getChildText("editor");
                    }
                    else

```

```

        nodeValue = element.getChildText (typeValue);

        if (nodeValue == null) {
            continue outter;
        }
        else {
            boolean flag = false;
            if (typeValue.equals("author")) {
                Vector andVector = Helper.seperate(para, " " and "<");
                boolean[] andFlag = new boolean[andVector.size()];
                for (int j = 0; j < andVector.size(); j++) {
                    String filledAuthor = (String) andVector.elementAt(j);
                    andFlag[j] = compare(request, filledAuthor, Helper.eleminateChar (nodeValue,
typeValue);

                }

                for (int t = 0; t < andVector.size(); t++) {
                    if (andFlag[t] == false)
                        break;
                    if (t == (andVector.size() - 1)) {
                        flag = true;
                        allTrueFlag = true;
                    }
                }

            }
            else if (typeValue.equals("title")) {
                allTrueFlag = false;
                Vector titleVector = Helper.vectorize(para, " " );
                boolean[] titleFlag = new boolean[titleVector.size()];
                for (int j = 0; j < titleVector.size(); j++) {
                    String filledTitle = (String) titleVector.elementAt(j);
                    titleFlag[j] = compare(request, filledTitle, Helper.eleminateChar (nodeValue,
typeValue);

                }

                for (int t = 0; t < titleVector.size(); t++) {
                    if (titleFlag[t] == false)
                        break;
                    if (t == (titleVector.size() - 1)) {
                        allTrueFlag = true;
                        flag = true;
                    }
                }

                for (int t = 0; t < titleVector.size(); t++) {
                    if (titleFlag[t] == true) {
                        flag = true;
                        flagForOrTitle = true;
                        break;
                    }
                }

            }
            else if (isDigit == true && (typeValue.equals("from") || typeValue.equals("to"))) {
                flag = compareYear(para, Helper.eleminateChar (nodeValue, typeValue);
                String checkTitle = (String) paras.elementAt(1);
                if (flag == true) {
                    if (checkTitle == null || checkTitle.length() == 0 || checkTitle.equals(""))
                        allTrueFlag = true;
                }
            }
            else {
                flag = compare(request, para, Helper.eleminateChar (nodeValue, typeValue);
                String checkTitle = (String) paras.elementAt(1);
                if (flag) {
                    if (checkTitle == null || checkTitle.length() == 0 || checkTitle.equals(""))
                        allTrueFlag = true;
                }
            }
        }
        if (flag == false) {
            flagForOrTitle = false;
            continue outter;
        }
    }

```

```

    }
    //end of inner

    if( allTrueFlag == true)
        registry.put(key, element);
    if(flagForOrTitle == true)
        titleOr.put(key, element);
    //end of butter

    session.setAttribute("titleOr", titleOr);
    session.setAttribute("result", registry);

    Enumeration v1 = titleOr.keys();
    while(v1.hasMoreElements()){
        String s1 =(String)v1.nextElement();
        Enumeration v2 = registry.keys();
        while(v2.hasMoreElements()){
            String s2 =(String)v2.nextElement();
            if(s1.equals(s2)){
                titleOr.remove(s1);
                break;
            }
        }
    }

    out.println("<html>");
    out.println("<head>");
    out.println("<title>" + "result" + "</title>");
    out.println("</head>");
    out.println("<body bgcolor=\"white\">");
    out.println("<body>");
    if (registry.size() != 0 && titleOr.size() != 0){
        out.println("<p align=\"center\">Number of records found: " +
            registry.size() + titleOr.size() + "</p>");
        out.println("<form method = \"post\" action=\"bib/SaveResult\">");
        out.println("<table align=\"center\" border=\"1\">");
        out.println("<tr bgcolor=\"#FFD000\">");
        out.println("<th>Check</th><th>Author or Editor</th><th>Title</th><tr>");
        int counter = 0;
        print(registry, out, counter);
        print(titleOr, out, registry.size());
        out.println("</table>");
        out.println("<center>");
        out.println("<input type=\"submit\" name=\"submit\" value=\"Check and Save Record\">");
        out.println("</center>");
        out.println("</form>");
    }
    else{
        out.println("<p align=\"center\">No record matches the search criterion.</p>");
    }

    out.println("<p align=\"center\"><a href=\"index.html\">Go to genreal search/</a></p>");
    out.println("<p align=\"center\"><a href=\"advanced.html\">Go to advanced search/</a></p>");
    out.println("</body>");
    out.println("</html>");
}

/**
 *
 * @param para String representation of year which is input by user.
 * @param nodeValue String representation of year which is from record
 * @param type From or Before
 * @return The <code>boolean</code> value of comparing the <code>para</code> with <code>
 * nodevalue</code> according to the type.
 */
public boolean compareYear (String para, String nodeValue, String type){

```

```

//TODO: parse error
if (type.equals("from")) {
    return Integer.parseInt(para) <= Integer.parseInt(nodeValue);
} else {
    return Integer.parseInt(para) >= Integer.parseInt(nodeValue);
}

/**
 * Compare the user input value with the corresponding value in an element node.
 * According to match case, match word, and match whole field, there are five
 * cases. Each record is represented as a node in the JDOM document tree.
 * The user input value will be compared to the corresponding element value
 * of the node. The comparison is processed according to the different cases.
 * @param request HttpServletRequest
 * @param para Parameter which is the user input value.
 * @param nodeValue Element value of the node.
 * @param typeValue The search type such as author, title, year, and so on.
 * @return The <code>boolean</code> value of comparing of para and nodeValue.
 */
public boolean compare(HttpServletRequest request, String para, String nodeValue, String
typeValue) {
    boolean result = false;
    boolean matchCase = (request.getParameter("case") != null);
    boolean matchWord = (request.getParameter("word") != null);
    boolean matchField = (request.getParameter("field") != null);

    int searchCondition;
    if (matchCase == false) {
        if (matchWord == false) {
            if (matchField == false)
                searchCondition = 0;
            else if (matchField == true)
                searchCondition = 1;
        } else if (matchWord == true) {
            if (matchField == false)
                searchCondition = 2;
            else if (matchField == true)
                searchCondition = 3;
        } else if (matchCase == false) {
            if (matchWord == false) {
                if (matchField == true)
                    searchCondition = 4;
            } else if (matchWord == true) {
                if (matchField == false)
                    searchCondition = 5;
            } else if (matchWord == true) {
                if (matchField == true)
                    searchCondition = 6;
            }
        } else if (matchCase == true) {
            if (matchWord == false) {
                if (matchField == false)
                    searchCondition = 7;
            } else if (matchWord == true) {
                if (matchField == false)
                    searchCondition = 8;
            } else if (matchWord == true) {
                if (matchField == true)
                    searchCondition = 9;
            }
        }
    }

    switch (searchCondition) {
        case 0:
            if (nodeValue.toLowerCase().indexOf(para.toLowerCase()) >= 0)
                result = true;
            break;
        case 1:
            if (nodeValue.indexOf(para) >= 0)
                result = true;
            break;
        case 2:
            if (typeValue.equals("author")) {
                //separate names
                Vector authorVec = Helper.separate(nodeValue, " and ");
                if (authorVec.contains(para.toLowerCase()))
                    result = true;
            } else {
                Vector vec = Helper.vectorize(nodeValue, " ");
                if (vec.contains(para.toLowerCase()))
                    result = true;
            }
            break;
        case 3:
            if (nodeValue.toLowerCase().equals(para.toLowerCase()))
                result = true;
            break;
        case 4:
            //same as case 2 without toLowerCase()
    
```

```

        if (typeValue.equals("author")) {
            //seperate names
            Vector authorVec = Helper.seperate(nodeValue, " and " );
            if (authorVec.contains(para))
                result = true;
        }
        else {
            Vector vec = Helper.vectorize(nodeValue, " ");
            if (vec.contains(para))
                result = true;
        }
        break;
    case 5:
        if (nodeValue.equals(para))
            result = true;
        break;
    default:
        break;
    }
    return result;
}

/**
 * Print the initial search results. It will print the author and title. In short,
 * sometimes text in the field must be "protected". This is done by enclosing the text
 * in braces. Here we eliminate the escape characters \, /, ', ", etc.
 * @param registry Hashtable. Key is the record key, value is the a node.
 * @param out
 * @param counter Counts the number of the result.
 */
public void printHashtable (registry, PrintWriter out, int counter) {
    List keys = new ArrayList (registry.keySet());
    ListIterator iter = keys.listIterator();
    while (iter.hasNext()) {
        counter++;
        String key = (String) iter.next();
        Element element = (Element) registry.get(key);
        String author = Helper.eleminateChar (element.getChildText("author"));
        if (author == null)
            author = Helper.eleminateChar (element.getChildText("editor"));
        String title = Helper.eleminateChar (element.getChildText("title"));

        if (author == null)
            author = " ";
        if (title == null)
            title = " ";
        out.println("<tr>");
        out.println("<td><input type=\"checkbox\" name=\"check\" " + counter +
            "\" value=\"\" " + key + "\" ></td>");
        out.println("<td>" + author + "</td>");
        out.println("<td>" + title + "</td>");
        out.println("</tr>");
    }
}

public void printElement (element, PrintWriter out, String key, int counter) {
    String author = Helper.eleminateChar (element.getChildText("author"));
    if (author == null)
        author = Helper.eleminateChar (element.getChildText("editor"));
    String title = Helper.eleminateChar (element.getChildText("title"));

    if (author == null)
        author = " ";
    if (title == null)
        title = " ";
    out.println("<tr>");
    out.println("<td><input type=\"checkbox\" name=\"check\" " + counter +
        "\" value=\"\" " + key + "\" ></td>");
    out.println("<td>" + author + "</td>");
    out.println("<td>" + title + "</td>");
}

```

```
        out.println("</tr>");
    }

    /**
     * Pass the <code>request</code> and <code>response</code> to
     * {@link #doPost(HttpServletRequest, HttpServletResponse) doPost} method.
     * @param request
     * @param response
     * @throws IOException
     * @throws ServletException
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        doPost(request, response);
    }
}
```