WSA - Web Site Analyzer

Yuan Xu

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements for

the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

September 2002

ABSTRACT

WSA - Web Site Analyzer

Yuan Xu

The purpose of this Major report is to develop a tool to analyze a web site. Maintaining a large web site is a significant task. There are tools available to help, but they are slow (because they probe the web site from a remote location), expensive, or cumbersome to use. The objective of this project is to design and implement an analyzer, which is simple to use, and which provides useful results quickly.

WSA is JAVA based program, which can be used as either an application or an Applet embedded in a web page. The main goal of this tool is to generate a report, which states the relationship of links in the web site. The cores distinctive of features of WSA are:

- For each HTML file, report links from it and links to it.
- Report the leaf URL, which does not have any outgoing URLs.
- Report the URLs, which are outside analyzed web site.
- Report the non-existing URLs inside analyzed web site.
- Report the permission denied URLs inside analyzed web site.

ACKNOWLEGEMENTS

Sincerely, I would like to express my deepest respect and gratitude to my supervisor Dr. Peter Grogono for his guidance, invaluable suggestions, encouragement and various prompt help throughout the course of this research work. I am very thankful for the opportunity that I had to work with Dr. Peter Grogono, and for everything he taught me during my master 's studies. The author is extremely thankful to his examiner Dr. Bipin C. DESAI for his professional supervision, intellectual guidance, and patience, through this study. It was a pleasure working with him.

Table of Figures

1 Introduction

The WSA [1]- Web Site Analyzer was designed to generate a report for a medium-sized web site. Although there are some similar commercial software products, the goal of this project is to produce a fast, optimized analyzer, which could help a Webmaster to easily maintain his/her web site(s).

1.1 Overview of Web Site

Most people today are familiar with the WWW(World Wide Web) [2]. You can find most information you need with your computer using a web browser. A web browser is an application used to visit web pages. The two most well known web browsers are Netscape Navigator and Microsoft Internet Explorer, which are used by the vast majority of people. Other browsers are available as well. Tim Berners-Lee [2] wrote the very first web browser at CERN, a European center for physics research. The first web browser to capture the public's interest was Mosaic [2], written by Marc Andreesen [4] and other undergraduate students at the National Center for Supercomputing Applications (NCSA) in the United States. Most of that group went on to form the core of Netscape Communications Corporation.

The Web was designed to be a universal space of information, so when you make a bookmark or a hypertext link, you should be able to make that link to absolutely any piece of information that can be accessed using networks. The universality is essential to the Web: it looses its power if there are certain types of things to which you can't link.

There are a lot of sides to that universality. You should be able to make links to a hastily jotted crazy idea and to link to a beautifully produced work of art. You should be able to link to a very personal page and to something available to the whole planet. There will be information on the Web, which has a clearly defined meaning and can be analyzed and traced by computer programs; there will be information, such as poetry and art, which requires the full human intellect for an understanding, which will always be subjective.

And what was the purpose of all this? The first goal was to work together better. While the use of the Web across all scales is essential to the concept, the original driving force was collaboration at home and at work. The idea was, that by building together a hypertext Web, a group of whatever size would force itself to use a common vocabulary, to overcome its misunderstandings, and at any time to have a running model - in the Web - of their plans and reasons.

A web browser, sometimes called a "User Agent", works by using a protocol called Hypertext Transport Protocol (HTTP [5]) to request a specially encoded text document from web server, such as Apache or IIS [6]. This text document contains special markup written in Hypertext Markup Language (HTML [4]). The User Agent interprets this markup. The job of the User Agent is to render the content of the document appropriate for the user.

The HTML may include such things as references to other web documents by using hyperlinks, suggestions for text, color and position and other content such as images and

audio/visual ("multimedia") content. Web pages may employ a technique called Cascading Style Sheets (CSS)[3], which is becoming more widely implemented by User agent developers, to make layout suggestions for various media.

1.2 Introduction to HTML

HTML, or Hypertext Markup Language, is designed to specify the logical organization of a document by using hypertext extensions. It is not designed to be the language of a WYSIWYG [4] word processor such as Word or WordPerfect. This choice was made because many different browsers may view the same HTML document. Thus, for example, HTML allows you to mark selections of text as titles or paragraphs and leaves the interpretation of these marked elements up to the browser. For example, one browser may indent the beginning of a paragraph, while another may only leave a blank line [4].

HTML instructions divide the text of a document into blocks called elements. These can be divided into two broad categories, those that define how the body of the document is to be displayed by the browser, and those that define information 'about' the document, such as the title or relationships to other documents.

The detailed rules for HTML (the names of the tags/elements, how they can be used) are defined using another language known as the standard generalized markup language, or SGML [6]. SGML is an international standard for the description of marked-up electronic text. More specifically, SGML is a meta-language, which is a means of formally

describing a language, in this case, a markup language. Before going any further we should define these terms.

However, SGML has useful features that HTML lacks. For this reason, markup language experts and software experts have developed a new language called XML [6] (Extensible Markup Language), which has most of the most useful features of HTML and SGML.

1.3 Objective and Scope of the Major Report

The objective of this Major Report is to develop a tool to analyze a web site. Maintaining a web site is a significant task. For a personal web site, there are thousands of files in it usually. This tool focuses on the web sites that consist less than 2000 files. Webmaster may want to know how many links do not exist; he/she may want to know the relationships of links inside his/her web site. There are tools available to help, but they are slow due to remote access, expensive, or cumbersome. The objective of this project is to design and implement an analyzer which is simple to use, and which provides useful results quickly.

The main goal of the tool is to give a clear relationship of links under a given URL. The core distinctive features of WSA are:

- This tool can be either an application or Applet. Applet could make WSA used via a web browser.

- An HTML formatted report can be generated by WSA. HTML formatted report make users surf on line easily. When user wants to generate the report, the browser will open report HTML file. URLs in the report are hyperlinks so that user can browser the link he/she wants.
- The table of URLs and outgoing URLs could be reported inside specific web site.
- A table of URLs and URLs using it could be reported inside specific web site.
- Leaf URLs that do not have outgoing URLs could be reported inside specific web site.
- Foreign URLs that are outside of specific web site could be reported.
- Report the non-existed URLs inside specific web site.
- Report the permission denied URLs inside specific web site.

2 Requirement for WSA

In this section, we will describe the requirement for WSA. The requirement will be divided into three parts: capabilities, development environment and additional requirement. The capabilities are core part in those requirements. The additional requirement will make application user-friendlier.

2.1 Capabilities

The user starts WSA by giving the URL as input via the user interface. The program scans HTML files recursively and builds a graph. Vertices of the graph are files and edges are links between files.

The Analyzer should find all links in the starting URL and all links that are:

- Reachable from those files by traversing links; and
- Inside user 's input URLs. For example, if user inputs "http://www.cs.concordia.ca/~grad/xuy", Analyzer would only search URLs starting with it.

In the web server, the private files usually are placed into different directories according to their category. For example, image files are usually located in the "/image" directory and audio files are located in the "/audio" directory. For the personal web site, there is home directory for the user. For example, Inside

"http://www.cs.concordia.ca/~grad/xuy", the private files are located under "../xuy". They consist of the graph. The analyzer should find all the files under this directory recursively. Because of those limitations, analyzer could not find dangling files in the web server. The dangling file means that no files reference it.

The reason that we set the limitation to search inside user's input URL is to prevent the Analyzer from returning the entire World Wide Web as its result. If there were no such limitation, the program would never end.

The Analyzer reports:

- The files that it found.
- For each file, the links from it and the links to it.
- "Orphans"- files with no incoming links.
- "Leaves"- files with no outgoing links.
- "Foreigners" - names of remote links.
- "Dead link" – URLs which are not found.
- "Permission denied link" – URLs which User does not have permission to access.

From the Webmaster's view, he/she would like to know how many files exist in his/her web site and the relationship between links within the web site. In order to work properly, WSA must parse the HTML, at least partially. As a side effect, it could produce a list of

warnings of HTML errors. Those report items should be transformed into HTML file format so that users could view them by using web browser.

2.2 Development Environment

The software tools that were used to develop WSA include:

- Platform – WINDOWS 98,WINDOWS 2000/NT, UNIX and LINUX.
- Developing language—JAVA, version JDK1.3.1
- IDE- JBuilder Foundation 4.0.

WINDOWS is the default platform because of its popularity. If user wants it to run on another platform, no modifications need to be made. The choice of JAVA as a programming language is due to its platform independence. Applet is another asset of using JAVA, because it makes it easy to be used on the Internet. JBuilder is most popular developing tool for JAVA application and it is very convenient for development and debugging.

2.3 Additional Requirements

In addition to the capabilities given in Section 2.1, the following requirements were added to WSA for the convenience of the user:

- Allow user to input a URL from a graphical user interface.

- Validate input URLs.
- Could be either an Application program or an Applet.
- Generate HTML in a well-formatted report.
- In the report HTML file, Users could traverse the links.
- Optimize the algorithm and make the application faster.
- Allow user to set up limitation of searched files

User-friendliness is a common requirement for all of today's applications. Our goal for interface design is that it is easy to use. Generated reports should be well formatted and straightforward, so that users can understand quickly.

3 Architecture and Design

In this chapter, we discuss the architecture and design of WSA. The architecture and design for WSA are not complicated, because efficiency is most important requirement for WSA.

3.1 Architecture

WSA has three subcomponents. These are: the main process, the parser subcomponent, and the report subcomponent. The use-case diagram in Figure 1 shows the relationships between these components.

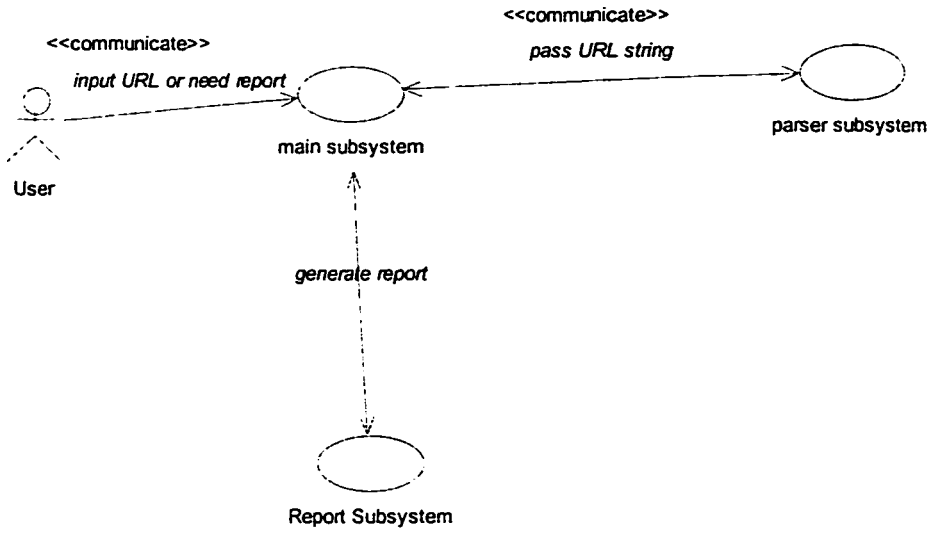

Figure 1 Use Case Diagram

After the user inputs the URL via the user interface, the main process component gets this URL and validates it. After validation of the input URL, the URL connection is established. The contents of the URL are obtained by opening a URL stream. Then the Parser subcomponent parses the string. The parser will get all URL links in the string and then put appropriate URLs into the appropriate data structure. After parsing the string, the control will go back to the main process subcomponent and repeat until the search list is empty. When the searching and parsing is finished, if the user wants a HTML format report, the report subcomponent will take over the control and transform the structure in memory into a report file.

3.2 Design of WSA

In this section, the detailed design of the three subsystems will be described respectively. The detailed algorithm will be cited in the implementation chapter and Appendix.

3.2.1 Main Process Subsystem

This component is the main control point of the program.

- Obtaining input from user. User Interface should allow user input a string format URL. The URL string format should be consistent with the input format of web browser.
- Validating the input. User may input wrong URL format, so WSA should validate user 's input and give corresponding error messages.

- Establishing a URL connection using URL given by users. After validation of URL format, a URL connection should be established so that WSA can open URL stream.
- Passing the URL stream to the parser. After opening URL stream, WSA will obtain the string format of HTML file. Then the string will be passed to the parser. The parser will gather the useful information in the string, such as URL links. Those links will be analyzed and sorted into different data structure.
- Creating all reported information and passing them to the report subcomponent. After searching is finished, the report information will be created.

In this component, all the GUI components are constructed for either the main application or the Applet. It communicates with the other components, so it is a core part of WSA.

3.2.2 Parser Subcomponent

The parser subcomponent takes care of parsing the HTML file. It obtains all the HTTP URLs from a given string format of the HTML file. It can find all links in the <a href="...">, "<FRAME src="...">, <BODY background ="..."> and " <IMG src="...">. The WSA will ignore the links such as "mail to", "gopher", "ftp" and so on. The parser will find the desired links by comparing the HTML tag.

- Receiving strings, parsing them, and constructing them into tag objects. An HTML file is consisted by a group of tags, so we shall define a tag object to make WSA more object-oriented.

- Constructing the string into tag objects. The parser shall transform the string format HTML into a group of tags.
- Argument objects construct the tag objects. A **Tag** will include a group of **Arguments.** An **Argument** is the text delimited by space inside a **Tag**. An **Argument** includes two parts which separated by "=". If right hand side of "=" is "SRC" or "HREF", right hand side of "=" shall be the URL link.

3.2.3 Report Subcomponent

The report subcomponent takes the results of the program as input. The results are some data structure, which is either vector or hash table. Then the report server creates a well-formatted HTML file and outputs it to the file.

- The report object is initialized after searching is finished. It should receive all reported information as parameters.
- The expected results are listed in the HTML file. HTML format report is used, because it is user-friendlier. User can traverse the links via web browser.
- Report the table of URLs and referenced URLs. User may want to know every HTML files and the links in those HTML files inside his/her web site.
- Report the table of URLs and URLs using it. User may want to know which HTML files reference a specific URL inside his/her web site.
- Report the leaf URLs, which do not have outgoing URLs. Some image files, document files are definite leaf URLs. If some HTML files do not have outgoing links, they belong to leaf URLs either.

- Report the foreign URLs, which are outside of the input URL. If the URLs are outside of input URL, they belong to foreign URLs.
- Report the dead URLs, which are out of date URLs. Inside specific web site, if some URL links do not exist, they should be reported as dead URLs.
- Report the permission denied URLs, which user does not have permission to access.
- All reported URLs are links, so users can traverse among them. In the report, all report items should be hyperlink format.

4 Detailed Design and Implementation

In this chapter, we discuss the detailed design and implementation of WSA. The algorithm of main flow control also is described in this chapter.

4.1 General Structure of WSA

In this section, we describe the general structure of WSA by means of UML diagrams. The reason that I chose UML diagram is that it is good graphic representation for object-oriented design.

4.1.1 Collaboration

In Figure 2, interactions of classes are described. It states the message and token passing between the classes.

The "WSA" object receives the user input. Then it processes user's input and gets string format HTML files. "WSA" will pass this string to "HTMLStringParser" in step 1. Then "HTMLStringParser" object will pass the string to the "Tag" object in step 2. The "Tag" object will pass string to "Argument" object in step 3 and it will get links in step 4. Those links will be passed back to "WSA" object in step 5. "WSA" object will process those links in step 6. In the last step, all information will be passed to "ReportServer" object.

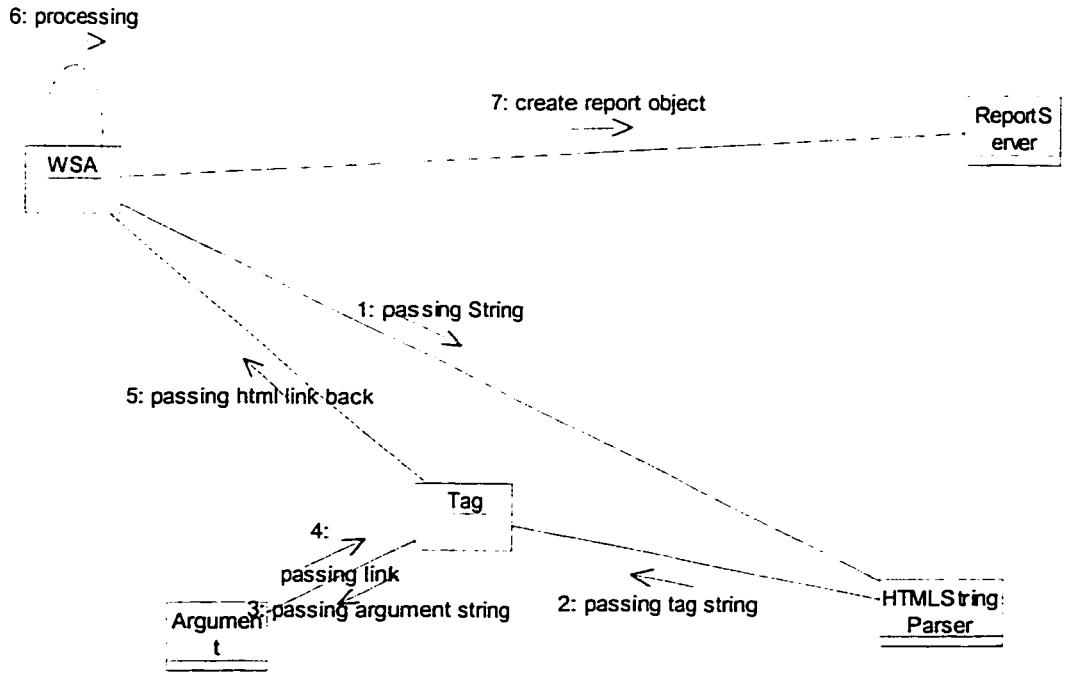

Figure 2 Collaboration Diagram

4.1.2 Message Sequencing

In Figure 3, we describe the message sequencing in the program. The message will be passed among five objects, "WSA", "ReportServer", "Tag", "Argument" and "HTMLStringParser".

Firstly, "WSA" get user's input and open the specific URL. Then "WSA" passes the string format HTML file to the "HTMLStringParser". Secondly, the message will be passed to the "Tag" object, because HTML file is consisted by a group of "Tag". Then the message will be passed to the "Argument" object. Finally the message that contains

the URL links will be passed back to "WSA" object. If User wants to generate the report, the "ReportServer" object will receive the message and generate the report.

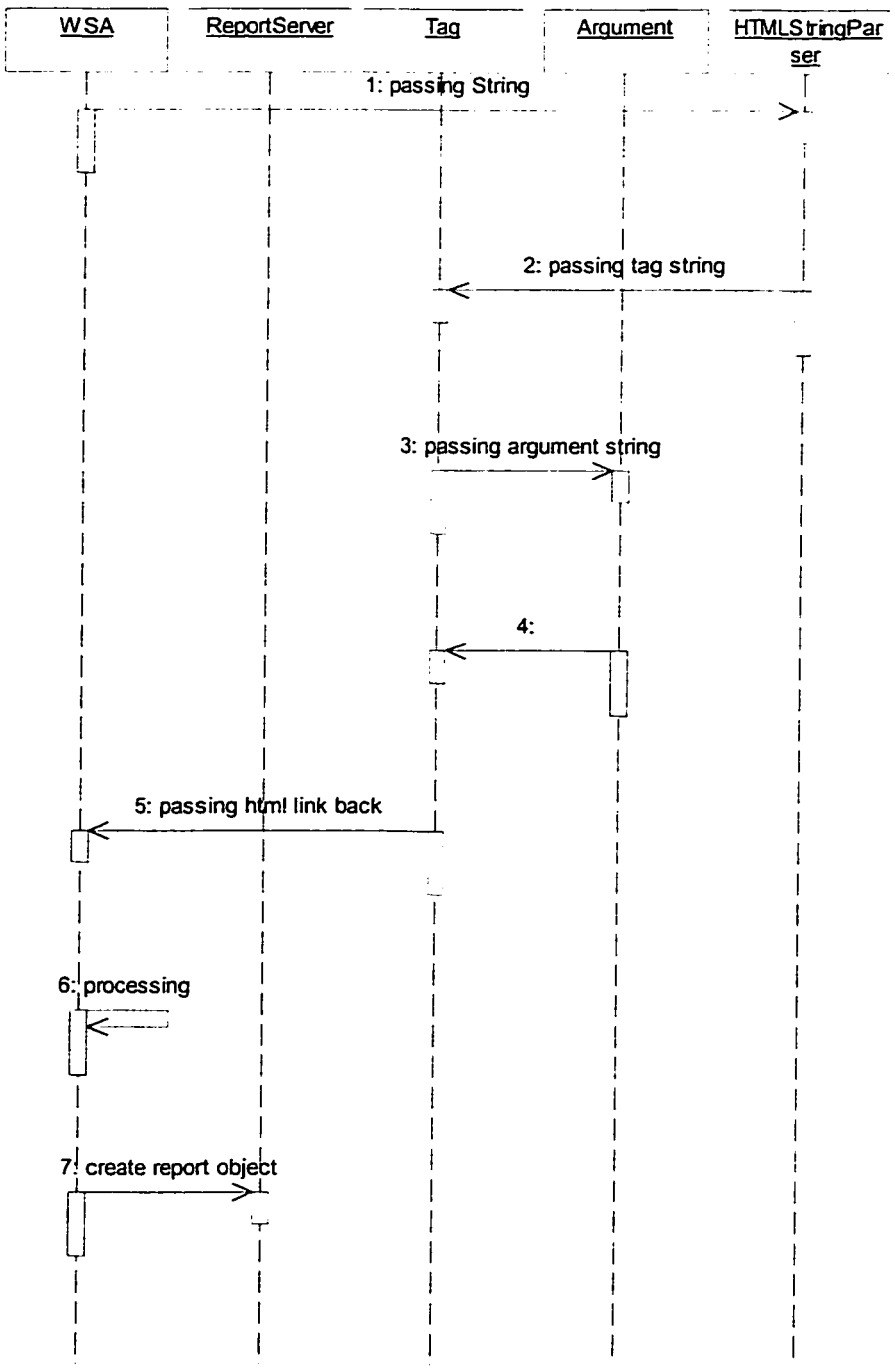

Figure 3 Sequence Diagram

4.1.3 Class Relationships

In figure 4, we describe the class relationships in the system. "Wsa" class plays most important role in the system. "Wsa" class has "use" relationship with "HtmlStringParser", "ReportServer", and "Tag" directly. "ReportServer" object is a static attribute in "Wsa" class. After the program stop running, the "ReportServer" instance will be instantiated. The "generate_report_html_file()" method will be called if "report" button is pressed. In "run()" method of "Wsa", the instance of HtmlStringParser" will be instantiated, which could parse the whole HTML file. In the "Tag" class, the method "URLString" could pass the string format URL to "Wsa" class. "Tag" class has a "use" relationship with "Argument" class. Vectors of Arguments consist of a "Tag" object. "HtmlUtils" class is helper class for "ReportServer" class.

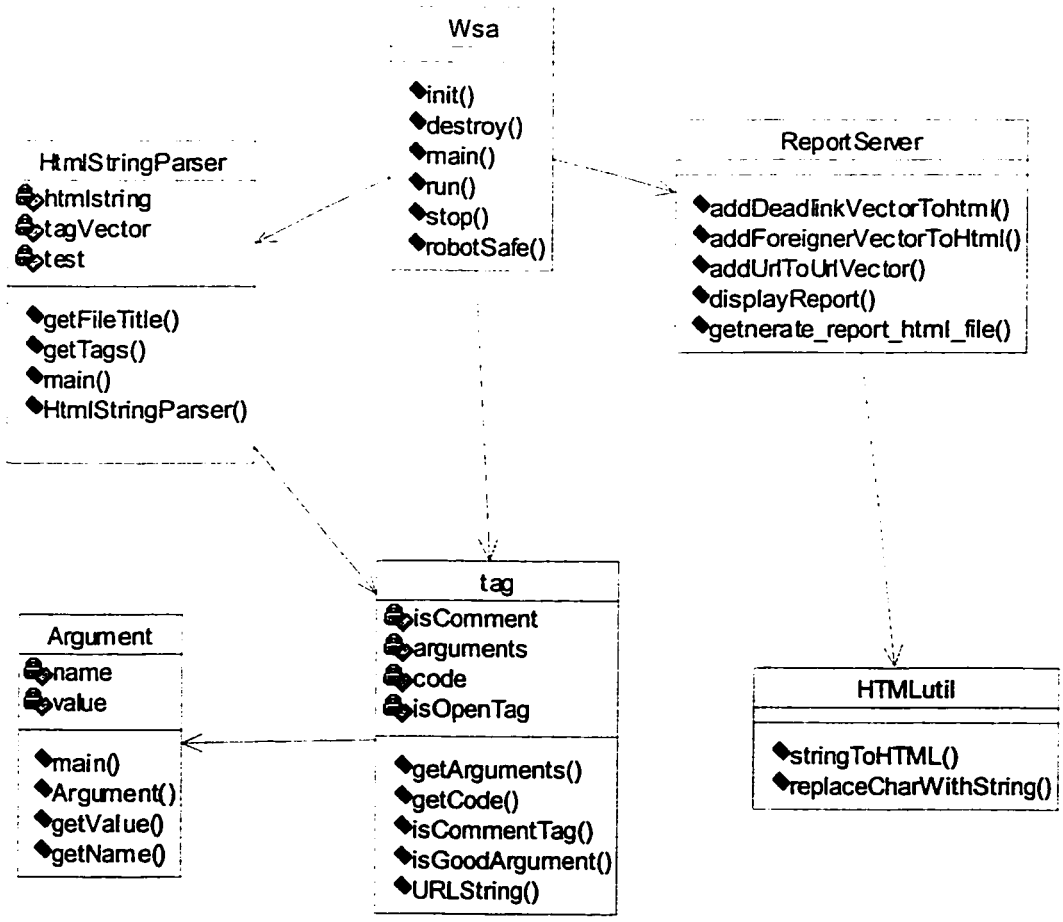

Figure 4 Class Diagram

4.2 Main Subsystem

The main subsystem has only one class, "WSA.java". This class is the core part of WSA. It constructs the GUI item for WSA and handles the main flow of control.

4.2.1 GUI Items in the Main Application Window

In figure 5, the main application window is showed. We will discuss functionality of GUI items respectively.

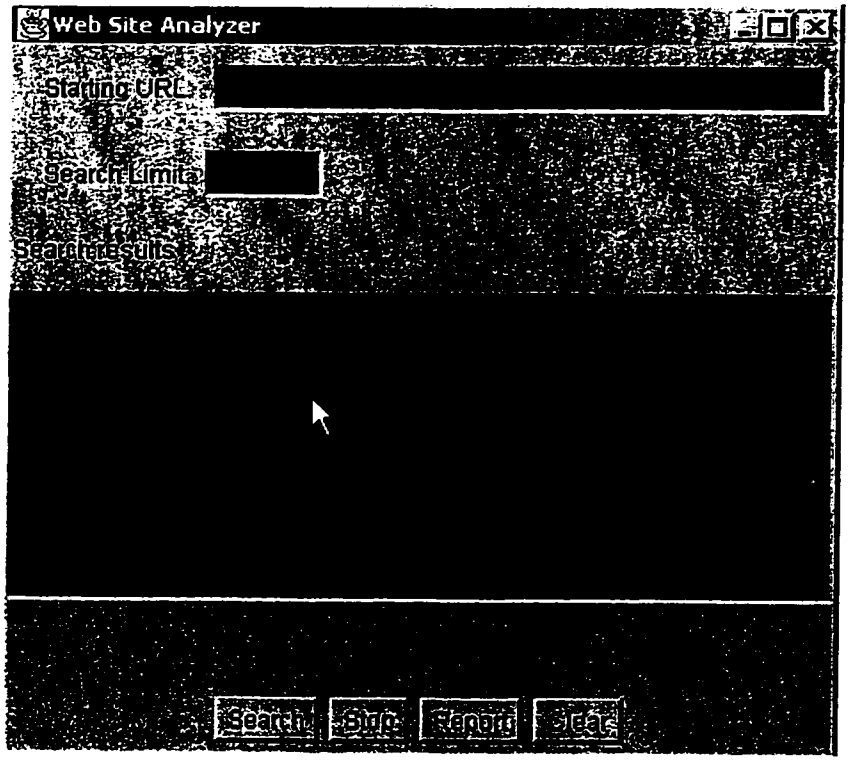

Figure 5 Application Window

- It creates the input text field, which takes the user input (URL). The program will validate the input URL format. The two formats of URL are acceptable. One is "http://xxxxxxxxx/xxx...", another format is "xxxxxxxxx/...". If any other format string is the input or empty string is inputted, an error message will show up in the status bar. The validation will inhibit user to input invalid URL string.
- It creates a text field for limiting searched URLs. The user may want to limit the number of searched URL in the case that some web sites have too many files. The

program will stop searching when this number is reached or all the files have been searched. User can set up the limitation. The default value is 1000. If user put a null value in this field, that will cause default value being used. When the limit is reached or searching is finished, either "done" or "limit reached" message will be displayed in the status bar.

- It creates an AWT list, which displays the searched URLs dynamically, so that the user can know how many links have been searched. User also could open the link from list by double clicking the link.
- It creates three buttons. The "Search" button – when this button is pressed the application program will start searching. Pressing this button will call "start()" method in WSA directly. The "Stop" button – when this button is pressed the application will stop searching. Pressing this button will call "stop()" method in WSA directly. The "Clear" button – when this button is pressed the application will clear AWT list, URL text field and set the limit text field to default value. At the same time, ReportServer object will be set to "null". The "Report" button – when this button is pressed the report server will be launched, a web browser will be opened and an HTML format report will be displayed. Pressing this button will call the "generate_html_report()" in "ReportServer" class directly. If user accidentally press report before searching or after pressing clear button, the error message shall tell user that report file is empty.
- It creates the status bar, which could display the current status of the application. When the application is searching the URL, it will display the searching URL. When the input is incorrect, an error message will be displayed.

4.2.2 Main Flow Control

When the search button is pressed, the searching thread is started. The "run()" method is called. The run method is the key part of this class.

- The first step is to validate the input URL string. The program allows the user to input two kinds of valid string formats. One is" http://xxxxxxxxxx... ", Another is "xxxxxxxxxx...". If latter is inputted, the "http://" is concatenated automatically. If users input the empty URL, the error message should be shown in status bar either. After validation, the base URL must be obtained because all relative URLs need a base URL in order to get a full URL. The program analyzes the input URL in order to get the appropriate base URL.
- The second step is to initialize some important data structures. "VectorToSearch" –This vector contains all the HTTP links to be searched. 'VectorSearched" - this vector contains all the HTTP links which the application has already searched. "LeafVector" - This vector contains all the HTTP links that don't have outgoing links. "UrlUrlVectortable" - this hash table contains a table of URLs and URLs referenced. "UrlUrlReferencedtable" – this hash table contains a table URL and URL using it. "ForeignerLink" - this vector contains all the HTTP links that are outside the base URL. "deadLink"—this vector contains all the HTTP links that are unavailable. "permissionDenyVector" – this vector contains the all the HTTP links that users are not authorized to view.
- The third step is to parse the given URL recursively and put the result into the above data structure. The following pseudo-code shows the algorithm.

Clear all the data structures (vectorToSearch, vectorSearched and etc).

Comments: We don't want any data from previous runs.

Put the given URL into vectorToSearch

Comments: It will initiate while loop.

While vectorToSearch.size() > 0 do

Remove first element from vectorToSearch.

If vectorSearched does not contain this URL,

put it into vectorSearched.

Open a URL stream and convert it into string format.

If opening the URL fails,

put it into the deadLink vector and break..

If a "URL NOT FOUND" page is opened,

put it into deadLink vector and break

If "permission deny " page is opened

put it into permissionDenyVector and break.

Pass string formatted HTML file to the Parser and obtain all HTTP links in it.

If the links are image or another format instead of HTML links and leafvector does not include them,

put them into leafvector.

If the HTML file does not have outgoing links,

put this URL into leafvector.

If vectorToSearch does not contain those links,

put them into vectorToSearch.

Put this URL and all the links inside this HTML file into urlUrlVectortable.

Comments: In this hash table, key is a URL link, value is a vector of URL going out of the Key URL.

If the link is outside of the base URL,

put it into foreigner vector.

End.

After searching all the URLs in the searchToSearch, obtain the UrlUrlReferencedtable by comparing vectorSearched and urlUrlVectortable

Comments: In this hash table, key is a URL, value is a vector of URL referencing the Key URL

- The last step is to create the "ReportServer" object and pass all above Data Structures to it.

4.2.3 User Interaction

Users can perform some actions on the GUI items. When the "search" button is pressed, the searching thread starts; When the "stop" button is pressed, the searching thread stops; When the "Report" button is pressed, the "displayreport" method of "ReportServer" is called; When the "Clear" button is pressed, all GUI items will be set to original state; If user accidentally press the "report" button when the report file is not generated yet, the error message will show up via dialog box; If user presses "enter" via keyboard after he/she finishes input, the searching will start; If user double click the item on the AWT list, the browser will open this link.

4.3 Parser Subcomponent

The parser subcomponent includes three classes, HtmlStringParser.java, Argument.java and Tag.java.

- "HtmlStringParser.java" – this class has a private attribute called "tagvector", which contains all the tags in this string. A **Tag** consists of text delimited by '<' and '>'. The constructor of this class will convert the given string into a **Tag** Vector.
- "Tag.java" – This class is responsible for converting a **Tag** object into an **Argument** object. An **Argument** is the text delimited by space in a **tag**. In the constructor, all tags that contain links will be put into the **Argument** vector, which is private attribute of the tag class. **Tag** includes an important method—"URLString()", which returns a string formatted link in the **tag**. If a tag have a **code** "IMG" or "A", " URLSTring()" method will return a string format of HTML links.

- "Argument.java"—In the **Tag**, there are a group of **Arguments** which have the following format: "x=y". "x" is called name and "y" is called value. The constructor of this class will convert **Tag** into a vector of argument.

4.4 Report Subcomponent

The report subcomponent includes two classes, "ReportServer.java" and "HTMLutils.java". They are responsible for creating the report HTML file.

"ReportServer.java" is core class in this component; "HTMLutils.java" is a helper class of "ReportServer.java".

- "ReportServer.java"—the attributes in this class map the corresponding attributes in "WSA.java". The constructor will pass a given value to those attributes. The "displayReport" method will call the system command to launch IE and open the given file. The "generate_report_Html_file" method will print the HTML formatted string into the "fileoutputstream". All report items is printed by their category.
- "HTMLutils.java" – this is the helper class for the generation of the HTML file. For some characters, such as "<" and ">", the browser cannot translate them directly, so "HTMLutils" class changes the format of the string into an understandable format.

5 Results, Conclusion and Future extension

In this section, we discuss the results, conclusion and future extension. WSA is small tool to help Webmaster to manage his/her web site. It may have some limitations. In future, it should be developed to suit most of web site.

5.1 Results

The HTML report is generated after the user presses the "Report" button. The browser will bring up an IE explorer window, which will display the report the file. This file is divided into several sections that can be browsed by the user.

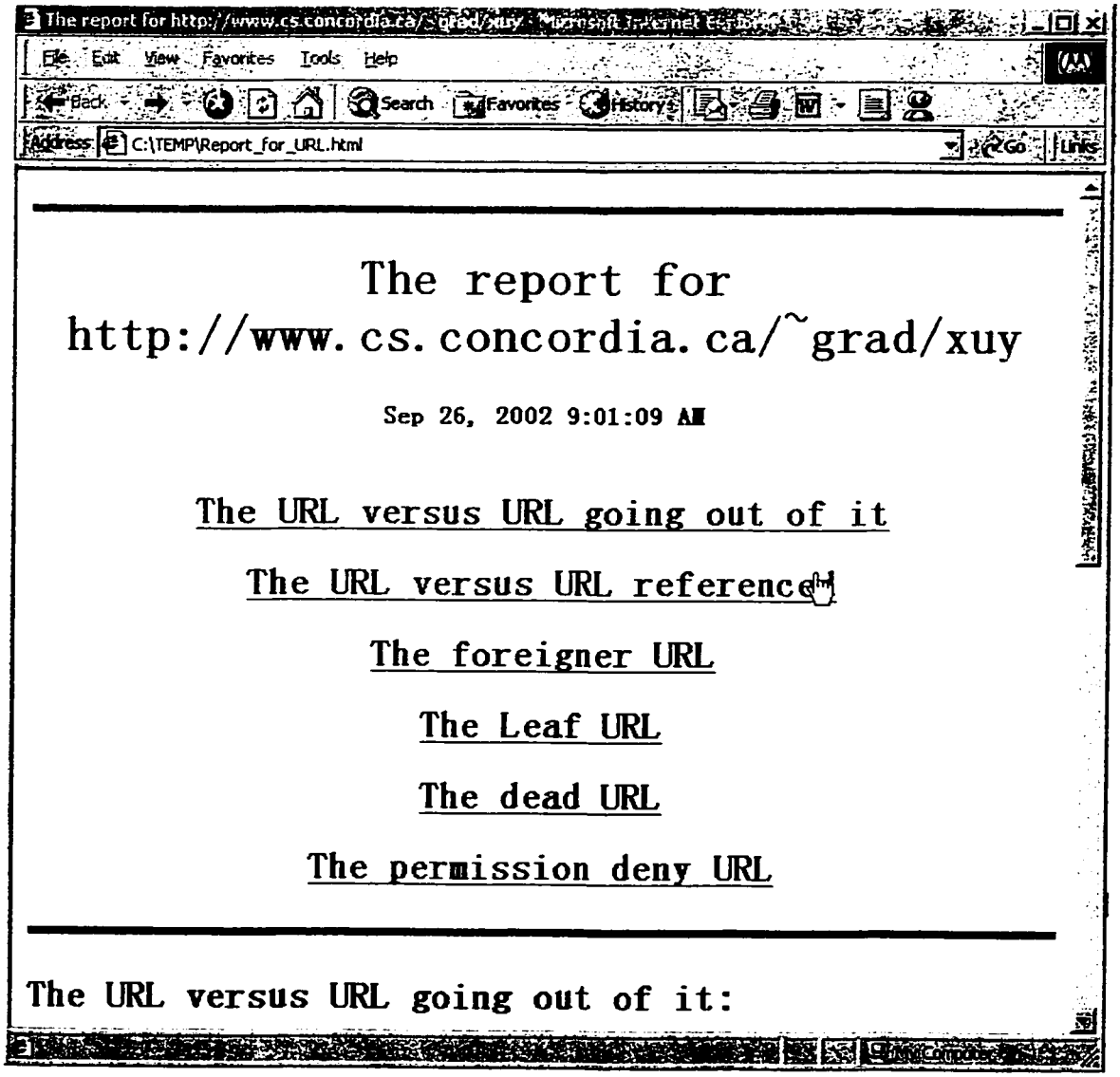

Figure 6 Reports HTML

5.2 Conclusion

WSA is a powerful tool for analyzing web sites. It is suitable for personal website management, especially for frequently updated web sites. For larger web sites, Webmasters usually cannot remember which links are dead. In addition, a Webmaster may want to know the relationships between links in his/her web site because traversing the links to find the relationship is tiresome. By using this tool, the user can find all needed information by inputting a one-line string. The HTML formatted report is straightforward; the user can browse any link in the report by just clicking it.

5.3 Future Extension

WSA—Web Site Analyzer and the description here may appear quite incomplete. The specification was kept minimal in order to avoid making key decisions before the first implementation effort. There are some improvements necessary:

- The sorting mechanism should be used in the report, because it can help the user find the links quickly. User could choose sorting by file name, file type, and so on.
- Dynamic pages should be handled in the future implementation, because more and more dynamic web page will show up on the WWW, such as PHP, ASP, and so on.

- The report HTML file can use Frame. This will make it user-friendlier. In the left hand side of pane, section links shall be listed; when user click the link in left side of pane, the content of that section shall be displayed in right hand side of pane.

References

1. Unknown, " SiteMapper", http://www.trellian.com/mapper/index.html, pp 1-5

2. Tim Berners-lee, "The World Wide Web: Past, Present and Future", Draft response to invitation to publish in IEEE Computer special issue of October 1996, pp 2-10

3. Anthony, "Anthony's CSS2 tutorial", "http://www.dynamicdeezign.com/css/introduction.html", March 5,2000, pp 2-10

4. NCSA, A Beginner's Guide to HTML, "http://archive.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html" Jan 23 2001, pp 4-22.

5. J. Gettys, J. Mogul. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", June 1999, pp 5-10

6. Jack Wallen, Jr., TechRepublic, "Look out Apache and IIS" 18 December 2001, pp 5-20.

7. James Clark, "Comparison of SGML and XML." By Reference identifiers: World Wide Web Consortium Note 15-December-1997, NOTE-sgml-xml-971215, pp 4-20.

Appendix

A.1 Code Segment and Explanations of WSA.java

```
public void run() {
    String strURL = textURL.getText();
    if(!strURL.startsWith("http"))
       strURL = "http://"+strURL;
  //String inputURL = strURL;
  String BASEURL =strURL;
```

Notes:
Obtain input URL and get uniform HTTP link format in case that user do not input full URL name sometimes.

```
if(strURL.endsWith("/"))
   BASEURL = strURL.substring(0,strURL.lastIndexOf("/"));
if(strURL.endsWith("html")
  || strURL.endsWith("htm")
  || strURL.endsWith("shtml")
  || strURL.endsWith("asp")
  || strURL.endsWith("jsp")
  || strURL.endsWith("php"))

BASEURL =strURL.substring(0 ,strURL.lastIndexOf("/"));
```

Notes:
Obtain the BASEURL, which will be used later.

```
// String strTargetType = choiceType.getSelectedItem();
int numberSearched = 0;
int numberFound = 0;
if (strURL.length() == 0) {
    setStatus("ERROR: must enter a starting URL");
    return;
}
// initialize search data structures
vectorToSearch.removeAllElements();
vectorSearched.removeAllElements();
listMatches.removeAll();
leafVector.removeAllElements();
urlUrlVectortable.clear();
urlUrlReferencedtable.clear();
foreinerLink.removeAllElements();
```

```
m_reportServer = null;

Notes:
Clear all data structure in very beginning in order to
remove any remaining records in previous run.

vectorToSearch.addElement(strURL);
while ((vectorToSearch.size() > 0)
        && (Thread.currentThread() == searchThread)) {
  // get the first element from the to be searched list
   strURL = (String) vectorToSearch.elementAt(0);
   vectorToSearch.removeElementAt(0);
   setStatus("searching " + strURL);
   listMatches.add(strURL);
   URL url;
   try {
     url = new URL(strURL);
   }
   catch (MalformedURLException e) {
   setStatus("ERROR: invalid URL " + strURL);
  break;
}

// mark the URL as searched (we want this one way or the
other)
if(!vectorSearched.contains(strURL))
   vectorSearched.addElement(strURL);
// can only search http: protocol URLs
if (url.getProtocol().compareTo("http") != 0)
    break;

// test to make sure it is before searching
if (!robotSafe(url))
    break;
URLConnection urlConnection;
try {
    // try opening the URL
    urlConnection = url.openConnection();

}
catch (IOException e) {
      setStatus("ERROR: couldn't open URL " + strURL);
      break;
}
urlConnection.setAllowUserInteraction(false);
InputStream urlStream;
//yuan
```

```
try{
   urlStream = url.openStream();
}
catch (Exception e)
{
   deadLinkVector.addElement(strURL);
   setStatus(strURL+" is dead link");
   continue;
}
```

Notes:
Try to open the given URL, if it failed, that mean this URL couldn't be opened (dead link).

```
if(strURL.indexOf(".html")>=0                                    ||
strURL.indexOf(".shtml")>=0
   ||              strURL.indexOf(".htm")>=0                     ||
strURL.indexOf(".asp")>=0
   ||              strURL.indexOf(".jsp")>=0                     ||
strURL.indexOf(".php")>=0
   ||strURL.endsWith("/")
   ||strURL.substring(strURL.lastIndexOf("/"),
     strURL.length()).indexOf(".") <0)
{
   byte b[] = new byte[1000];
   int numRead;
   String content=null;
   try{
      numRead = urlStream.read(b);
      content = new String(b, 0, numRead);

      while (numRead != -1) {
      if (Thread.currentThread() != searchThread)
          break;
      numRead = urlStream.read(b);
      if (numRead != -1) {
         String newContent = new String(b, 0, numRead);
         content += newContent;
     }
}
```

Notes:

Obtain the string format of HTML file, which will be passed to HTMLStringParser later.

```
 urlStream.close();
}
catch (Exception e)
{  System.err.println("Can not read content of URL");
}
if (Thread.currentThread() != searchThread)
    break;
HtmlStringParser hsp = new HtmlStringParser(content);
if(content.indexOf("URL NOT FOUND")!=-1){
   if(!deadLinkVector.contains(strURL)){
   deadLinkVector.addElement(strURL);
   }
     break;
}
```

Notes:
In some cases, if a link cannot be opened, the default "URL NOT FOUND" page will be opened instead. This code segment shows that the link will be put into deadLinkVector if "URL NOT FOUND" page is opened.

```
Vector tagVector = hsp.getTags();
Enumeration eTemp = tagVector.elements();
while (eTemp.hasMoreElements()) {
   Tag tTemp = (Tag) eTemp.nextElement();
   String uString = tTemp.URLString();
   if ((uString !=null)
       && (uString.indexOf("mailto:") == -1)
       && (uString.indexOf("gopher:") == -1))
   {
       hasOutgoinghtml = true;
```

Notes: If there are outgoing link in this HTML file, the hasOutgoinghtml flag will be changed into true

```
       // that means this is relative URL.
       if(uString.indexOf("http") !=0){
          if(uString.indexOf("/") == 0)
             uString = BASEURL+ uString;
       else
          uString= BASEURL+"/" +uString;
   }
```

Notes:
Concatenate the BASEURL with relative URL in order to obtain full URL.

```
if(uString.indexOf("http:") ==0
   && uString.indexOf("//") < 0)
   uString = BASEURL + "/" +
             uString.substring(uString.indexOf("http:")+5,
                               uString.length());
if(!leafVector.contains(uString)
   &&uString.startsWith(BASEURL)
   &&(uString.indexOf(".html")<0||
   uString.indexOf(".shtml")<0
   ||uString.indexOf(".htm")<0 || uString.indexOf(".asp")<0
   ||              uString.indexOf(".jsp")<0              ||
uString.indexOf(".php")<0
   ||!uString.endsWith("/")||
     uString.substring(strURL.lastIndexOf("/")
                       ,strURL.length()).indexOf(".")   >=0)
)

   leafVector.addElement(uString);
// we put URL which is outside base url into a vector
```

Notes:
If there are not links in the HTML file, This URL will be put into leafVector.

```
if(uString.indexOf(BASEURL)==-1
   && !foreinerLink.contains(uString))
   foreinerLink.addElement(uString);
```

Notes:
If URL does not include BASEURL, put it into foreingerLink.

```
if(uString.startsWith(BASEURL)&&
  (!vectorToSearch.contains(uString)))
{
  vectorToSearch.addElement(uString);
}
```

Notes:
If the URL is not in the vectorToSearch and not a foreigner, it will be put into vectorToSearch vector.

```
if (!urlUrlVectortable.containsKey(strURL))
{
    Vector temp =new Vector();
    temp.addElement(uString);
    urlUrlVectortable.put(strURL,temp);
}
else
{
    Vector temp =(Vector) urlUrlVectortable.get(strURL);
    if (!temp.contains(uString)){
        temp.addElement(uString);
}
```

Note:
Construct urlUrlVectortable, the key is a URL link; the value is a vector of links referenced.

```
}
}
}
}

if(strURL
  !=null&&strURL.startsWith(BASEURL)
  &&!leafVector.contains(strURL)
  && hasOutgoinghtml  == false)
  leafVector.addElement(strURL);
```

Notes:
If the HTML file does not include any the links, put it into Leafvector.

```
if((textURL1.getText()!=null)
   &&!(textURL1.getText().equals(""))
   &&!(textURL1.getText().equals("1000")))
   SEARCH_LIMIT=Integer.getInteger(textURL1.getText())
               .intValue();
   numberSearched++;
   if (numberSearched >= SEARCH_LIMIT)
       break;
   }
Enumeration eVectorSearched =vectorSearched.elements();
while (eVectorSearched.hasMoreElements()) {
       Object sTemp = eVectorSearched.nextElement();
       Enumeration eKey =urlUrlVectortable.keys();
       while(eKey.hasMoreElements()){
             Object kTemp =eKey.nextElement();
```

```
                VectorValues=(Vector)
                            urlUrlVectortable.get(kTemp);
                if(Values.contains(sTemp)) {
                if(!urlUrlReferencedtable.containsKey(sTemp))
{
                    Vector vTemp = new Vector();
                    vTemp.addElement(kTemp);
                    urlUrlReferencedtable.put(sTemp,vTemp);
                }else{
                      Vector vTemp =(Vector)
                      urlUrlReferencedtable.get(sTemp);
                      if(!vTemp.contains(kTemp))
                      vTemp.addElement(kTemp);
                      }
          }
    }
}
```

Notes:
Construct the urlUrlReferencedtable, which is obtained by analyzing the urlUrlVectortable and vectorSearched. The key is a URL; value is a vector of URL referencing the key URL.

```
m_reportServer=
new ReportServer(urlUrlVectortable,foreinerLink,leafVector,
BASEURL,deadLinkVector,urlUrlReferencedtable);
```

Notes:
Pass these parameters to ReportServer class.

```
if (numberSearched >= SEARCH_LIMIT || numberFound >=
SEARCH_LIMIT)
     setStatus("reached search limit of " + SEARCH_LIMIT);
else
     setStatus("done");
     searchThread = null;
}

void setStatus(String status) {
     labelStatus.setText(status);
}
public void actionPerformed(ActionEvent event) {
          String command = event.getActionCommand();

          if (command.compareTo(SEARCH) == 0) {
          setStatus("searching...");
```

```
            // launch a thread to do the search
                if (searchThread == null) {
                    searchThread = new Thread(this);
                }
                searchThread.start();
            }
            else if (command.compareTo(STOP) == 0) {
                    stop();
            }
            else if(command.compareTo(REPORT)== 0) {
                   m_reportServer.displayReport(

m_reportServer.generate_report_html_file());
                   m_reportServer = null;
            }
}
```

Notes:
If "Report" button is pressed, ReportServer class will generate HTML file.

```
public static void main (String argv[]){
Frame f = new Frame("Web Site Analyzer");

Wsa applet = new Wsa();
f.add("Center", applet);
f.addWindowListener(new WindowAdapter()
{ public void  windowClosing (WindowEvent e)
{
System.exit(0);
}
});
applet.init();
applet.start();
f.pack();
f.show();
}

}
```

A.2 Code segment and Explanations HtmlStringParser.java

```
public HtmlStringParser(String x)
{
   htmlString=x;
   try
  {
     int index=0;
     int index2=0;
     tagVector =new Vector();
     while(((index=x.indexOf("<", index))
            !=-1)&&((index2=x.indexOf(">", index))!=-1))
     {
            tagVector.addElement(new

Tag(x.substring(++index,index2)));
           index=index2;
     }
   }
   catch (Exception e)
   {
      e.printStackTrace();
      System.out.println(" there are something wrong
                          during
                          parsing the HtmlString");
   }
}
```

Notes:
The constructor of HTMLStringParser will convert String into a vector of tag objects.

A.3 Code Segment and Explanations of Tag.java

```
public  Tag (String tagstr)
{
    StringTokenizer st=new StringTokenizer(tagstr);
    if(st.hasMoreTokens())
       code=st.nextToken();
    if(code.charAt(0)!='/')
       isOpenTag=true;
    if(code.charAt(0)!='!')
       isComment=false;
    if(code.equalsIgnoreCase("IMG")
       ||code.equalsIgnoreCase("A")){
       while (st.hasMoreTokens()){
              String s=st.nextToken();
              while(!isGoodArgument(s))
                    s=s+st.nextToken();
              arguments.addElement(new Argument (s));
       }

}
```

Notes:
The constructor of tag class will convert <IMG ...> and <A ... > into a vector of Arguments.

```
}
public boolean isGoodArgument(String s){
   if(s.indexOf("=")== -1)
      return false;
   if(s.endsWith("="))
      return false;
   if(s.indexOf("\"")>=0
      &&s.indexOf("\"")==s.lastIndexOf("\""))
      return false;
   return true;
}
```

Notes:
In the HTML files, some arguments are not well formatted. This method will test if the argument is good or not.

```
public String URLString () {
   if (code.equalsIgnoreCase("A")){
      for(int m=0;m<arguments.size();m++){
```

```
          if(((Argument)arguments.elementAt(m))
          .getName().equalsIgnoreCase("href")) {
          String temp = ((Argument) arguments.

elementAt(m)).getValue();
          if(temp.indexOf("\"")==0)
          temp = temp.substring(1,temp.length()-1);
          if(temp.indexOf("#")!=0)
             return temp;
           }
       }
   }
   if(code.equalsIgnoreCase("IMG")){
   for(int m=0;m<arguments.size();m++){
       if(((Argument)
            this.arguments.elementAt(m)).getName()
            .equalsIgnoreCase("src")){

            String temp = ((Argument)
            arguments.elementAt(m)).getValue();
            temp = temp.substring(1,temp.length()-1);
            return temp;
       }
    }
  }
  return null;
}
```

Notes:
This method will obtain the URL string from argument objects.

A.4 Code Segment and explanations Argument.java

```
public  Argument (String str)
{
   StringTokenizer temp=new StringTokenizer(str,"=");
   if(temp.hasMoreElements())
      name=temp.nextToken();
   if(temp.hasMoreElements())
      value = temp.nextToken();
   }
public String getName ()
{
return name;
}
public String getValue ()
{
return value;
}
```

Notes:

Name and value are delimited by "=".

A.5 Code Segment and Explanations ReportServer.java

```
public class ReportServer {
   private Hashtable m_urlUrlVtable =new Hashtable();
   private Hashtable m_urlUrlRtabel =new Hashtable();
   private Vector    m_foreignerVector = new Vector();
   private Vector    m_leafVector = new Vector();
   private Vector    m_deadLink =new Vector();
   private String    m_baseURL = null;
   private FileOutputStream _fileHandle=null;
   private PrintWriter _printStream=null;
   protected static String REPORTS_DIR="c:/windows/temp";
```

Notes:
REPORTS_DIR is a constant, which is platform dependent.
User may change it when application runs in other platform.

```
public synchronized static Object[] displayReport( String
fileName )
{
   Object[] _result = new Object[2];
   _result[0] = Boolean.TRUE;
   _result[1] = null;

   // Get the browser to use ...
   System.setProperty("global.browser",
   "c:\\program files\\internet explorer\\iexplore");
   String _browserName = System.getProperty(
                                    "global.browser" );
   if( _browserName != null )
   {
      // Get the Runtime object and open the file in the
      // browser.
      String [] _cmd = new String[2];
      _cmd[0] = _browserName;
      _cmd[1] = fileName;
      try
      {
        Runtime.getRuntime().exec( _cmd );
      }
      catch( IOException e )
      {
        _result[0] = Boolean.FALSE;
        _result[1] = e.getMessage( );
      }
```

```
}
else
{
   _result[0] = Boolean.FALSE;
   _result[1] = "Browser not set";
}

return _result;
}
```

Notes:
This method will call system command, which launches IE and open specific file.

```
public String generate_report_html_file ()
{
   boolean _fileCreated = true;
   String _fileName =null;
   try
   {
     _fileName= REPORTS_DIR + "/"+ "Report_for_URL.html";
     _fileHandle = new FileOutputStream (_fileName);
     _printStream = new PrintWriter (_fileHandle);
     String _pageTitle ="The report for " + m_baseURL;
     _printStream.println("<html>\n<head><title> " +
                          _pageTitle +
                          "
</title></head><body><center>");
     _printStream.println("<hr width=100%
                           noshade>");
     _printStream.println("<h1>" +_pageTitle
                           + "</h1></center>");
     _printStream.println("<table width=\"100%\">");
     _printStream.println("<tr align=center>");
     _printStream.println("<td align=center><b>" +
     DateFormat.getDateTimeInstance( ).format(
                       new java.util.Date( ) ) + "</b>" );
     _printStream.println("<br><br></tr></table>");
     _printStream.println("<h2 align=center>
           <a href=#section1>The URL versus URL going
           out of it  </a></h2>");
     _printStream.println("<h2 align=center>
              <a href=#section2>The URL versus URL
              referenced</pre>  </a></h2>");
     _printStream.println("<h2 align=center>
          <a                href=#section3>The     foreigner
URL</a></h2>");
```

```
      _printStream.println("<h2 align=center>
            <a href=#section4>The Leaf URL</a></h2>");
      _printStream.println("<h2 align=center>
                <a href=#section5>The dead URL </a></h2>");
      _printStream.println("<hr width=100%
                 size=5  noshade>");
      _printStream.println("<h2 align=left>
                <a name=section1>The URL versus URL going
               out of it:  </a></h2>");
       _printStream.println("<hr width=100%
                              size=5  noshade>");
       addUrlToUrlVector ();
      _printStream.println("<hr width=100%
                            size=5  noshade>");
      _printStream.println("<h2 align=left>
              <a    name=section2>The    URL    versus    URL
referenced:
             </a></h2> ");
      _printStream.println("<hr width=100%
                            size=5  noshade>");
      addUrlToRUrlVector ();
      _printStream.println("<hr width=100% size=5
                             noshade>");
      _printStream.println("<h2 align=left>
          <a name=section3> The foreigner URL: </a></h2>");
      _printStream.println("<hr width=100%
                             size=5  noshade>");
      addForeignerVectorToHtml();
      _printStream.println("<h2 align=left>
                <a  name=section4>  The  Leaf  URL:  </a></h2>
");
      _printStream.println("<hr width=100%
                            size=5  noshade>");
                            addLeafVectorToHtml();
      _printStream.println("<hr width=100%
                            size=5  noshade>");
      _printStream.println("<h2 align=left>
            <a name=section5> The dead URL: </a></h2>");
      _printStream.println("<hr width=100%
                           size=5  noshade>");
      addDeadLinkVectorToHtml();
      _printStream.println("</body></html>");
      _printStream.close();
      _fileHandle.close();
}
catch (Exception e)
{
```

```
      System.err.println("The report can not be generated");
}
   return _fileName;
  }
}
public void  addUrlToUrlVector ()
{
if ( _printStream != null ){
     if(!m_urlUrlVtable.isEmpty()){
     int key_num=m_urlUrlVtable.size();
     Enumeration eKeys=m_urlUrlVtable.keys();
     _printStream.print("<ul>");
     while(eKeys.hasMoreElements()){
     String uString =(String)eKeys.nextElement();
     HTMLutils HTMLutils_instance = new HTMLutils();

     _printStream.print("<li><h3 align=left>" +
                          "<A HREF=\" "
              +HTMLutils_instance.stringToHTML(uString)
              +"\""+">" +HTMLutils_instance.stringToHTML
              ("<"+uString+">")
               +"</A>" + ": "+"</h3>");
     Vector vTemp=(Vector) m_urlUrlVtable.get(uString);
     int size =vTemp.size();
     Enumeration urlEnum=vTemp.elements();
     _printStream.print("<ol>");
     while(urlEnum.hasMoreElements()){
        String sTemp= (String) urlEnum.nextElement();
        _printStream.print("<li>");
        _printStream.print("<A HREF=\" "+
               HTMLutils_instance.stringToHTML(sTemp)+"\""
              +">" +HTMLutils_instance.stringToHTML
              ("<"+sTemp+">")
              + "</a>" );
      }
     _printStream.print("</ol>");

    }
     _printStream.print("</ul>");
  }
   else {
_    printStream.println("<b> It is empty </b>");
  }
 }
}
```

Notes:

Those methods will print the well-formatted HTML file.

A.6 Code Segment and Explanations of HTMLutils.java

```
// Example: if an input string contains the '<' character,
this gets
// converted to '&lt;'
public String stringToHTML( String inputString )
{
   String newstr, newstr1;

   // replace < with &gt;
   newstr = new String(
            replaceCharWithString( '>',
                                   inputString, "&gt;" ) );

   // replace > with &lt;
    newstr1 = new String( replaceCharWithString(
                                  '<', newstr, "&lt;" ) );
    newstr = newstr1;

    return newstr;
}

// Converts a character int a string with a string
// used byte stringToHTML
private  String  replaceCharWithString(  char  c,  String
originalString, String substring )
{
    String temp = new String();
    String temp1;
    String loopString = new String( originalString );
    int index;

    while( (index = loopString.indexOf( c )) != -1 ) {

    // create a substring up to the first character c
    if( index > 0 ) {
       temp1 = loopString.substring( 0, index );
       temp = temp1;
    }

    // add the substring replacing the character c
    temp1 = temp.concat( substring );
    temp = temp1;
```

```
    // add the rest of the string after character c
    temp1 = temp.concat( loopString.substring
                         ( index + 1 ) );
    temp = temp1;

    loopString = temp;
   }

    return loopString;
  }
}
```

Note:
It converts an input string to HTML-friendly format for display in a browser