Hyperlinks Analysis of Dynamic Web Applications

Angle Hsieh and Dowming Yeh

National Kaoshiung Normal University
Kaoshiung, Taiwan 802, Republic of China
AngleHsieh@ttnt.com.tw, dmyeh@nknucc.nknu.edu.tw

Abstract. There are several approaches available to implement complex and adaptive Web Application. One of the most favorite approaches is dynamic Web page technique. Many structural problems such as void links and broken links may be hidden in a web site with dynamic pages. Thus, techniques to aid in understanding and restructuring of a Web site can help Web masters significantly. This work proposes a set of structural analysis methods for diagnosis of some common problems to meet real maintenance needs. Systemized analyses are employed to gather key attributes of the structure in a Web site, and the structural information gathered can be utilized to analyze the quality of the site structure easily.

1 Introduction

Maintaining a Web-based application is not easier than a traditional application. Numerous arts, page layouts and hyperlink designs are used and slacked in designing Web-based application, aside from database connection and program code. There are several approaches available to implement complex and adaptive Web Application. One of the most favorite approaches is to produce Web pages dynamically using server-side script languages such as ASP. This gives developers flexibility and adaptability to construct more powerful Web applications. A Web page produced by the script is a dynamic Web page. Web sites or Web applications with dynamic pages are thus called dynamic Web sites or applications.

There are few studies addressing the issues of Web site understanding and restructuring [1]. Ricca and Tonella completely tendered a method for structure readjusting and implemented tools, *ReWeb* and *TestWeb*, for modeling and understanding Web sites [4] [5]. These tools can analyze Web sites, and display the outcome of structural analyses. However, they only can analyze sites consisted of static pages with frames.

Di Lucca et. al. also present a tool for reverse engineering Web applications into UML diagrams [2]. They extract the attributes of Web pages into an Intermediate Representation Form, and then translate this Form into a relational database. The class diagrams can be produced directly from the database. Other diagrams such as use case diagram and sequence diagram may be constructed after performing some queries on the database. There is little discussion regarding dynamic page issues in these works.

J.M. Cueva Lovelle et al. (Eds.): ICWE 2003, LNCS 2722, pp. 349–352, 2003.

2 Algorithms for Gathering Hyperlinks

The *StruGatherer* algorithm processes all Web pages in a target site, and the results are represented in the form of page objects and link objects. A page object may own many link objects, but a link object can only belong to one page object. A page object *P<p, lo, d)>* has attributes "page name", "page location", "page type". The page name *p* with page location *lo* attributes can identify a unique page in the site. In the following discussion, the Active Server Page (ASP) will be used to explain the concept of our methods and how it works.

A page object may own zero to *n* link objects, *L<q, llo, lvn, lvp>*. A link object has attributes "target page name", "target page location", "identification number", and "variable location". The attribute *lvn* is an identification number to identify links that are created from the same hyperlink through variable assignments. The name of the page containing the link will be saved in *lvp*. Each link object will be conjoined with its containing page object upon creation.

The pseudo-code of the algorithm is shown in Figure 1. *PageInfo* creates a new page object *P(v)* corresponding to the page *v*. The *DecomposePage* procedure separates the dynamic blocks B^D from the static blocks [3]. Similar to *PageInfo, LinkInfo* extracts the essential information from a hyperlink and creates a link object. Then the target page name is examined to determine whether it is a regular link. If it is a variable, the link object *L(l)* becomes a *virtual* link object by setting its identification number *lvn* to *i*. The number *i* is a unique number beginning with zero and increases progressively when new variables are processed. This number is to identify that different virtual link objects are in fact assigned with the same hyperlink.

```
Algorithm StruGatherer (D)
Input: D = {all pages in home directory, sub-directory, virtual directory}
1.    i ← 1
2.    for each pages v in D do  //visit all of pages
3.          P(v) ← PageInfo(v)
4.          B^D (v)← DecomposePage(v)
5.          for each link l in B^D(v) do
6.                P(v).L(l) ← LinkInfo(l)
7.                if P(v).L(l).q is not a regular link // variable
8.                   P(v). lvn ← i
9.                   Call GatherLinkVariable (D, P, v, P(v).L(l).q, l)
10.                  i ← i + 1
11.               end if
12.         loop
13.   loop
14.   return P     // Return the set of page objects
```

Fig. 1. The *StruGatherer* algorithm

The pseudo-code of algorithm *GatherLinkVariable* is divided into three segments depending on the kind of the variable under analysis: dynamic and static local variable, inquiry variable, and global variable. We do not consider database objects since for a database field object, all possible links can be determined from the database field.

```
Algorithm GatherLinkVariable (D, P, v, b, l)
Input: D = { all pages in home directory, sub-directory, virtual directory}, the page
object set P, current page v, the variable name b, the link l
 1.   select case type(b)
 2.   case local variable
 3.      fv ← open(v)
 4.      while not fv.EOF and MatchLHS(b) do
 5.            r ← FetchRHS() //right hand side of "b="
 6.            if r is a variable then Call GatherLinkVariable(D, P, v, r, l)
 7.            if r is a page then Call CreateActualLinkObject(r, P(v).L(l))
 8.      loop
 9.   case inquiry variable
10.      for each pages cp in D do
11.         fcp ← open(fcp)
12.         while not fcp.EOF and MatchLHS(FetchArg(b)) do
13.            if SamePageName(cp, P(v).p))
14.               r ← FetchRHS()
15.               if r is a variable then Call GatherLinkVariable(D, P, v, r, l)
16.               if r is a page then  Call CreateActualLinkObject(r, P(v).L(l))
17.            end if
18.         loop
19.      loop
20.   case global variable
21.      for each pages cp in D do
22.         similar procedure to local variable
23.    loop
24.   end select
```

Fig. 2. The *GatherLinkVariable* algorithm

The scope of a local variable is the containing page. Therefore, all possible link values can be extracted from *v*. In the loop, all tokens at the left hand side of an assignment statement and matching the variable *b* are successively discovered and the value at the right hand side of the assignment statement is extracted. If an actual page is found, a new link object will be created by *CreateActualLinkObject* and added into the page object *P(v)*. This algorithm may be called recursively since variable may be assigned values by the other variables.

Inquiry variables act like parameter passing for conventional programs and appear as parameters for a *request* method in the target page. According to the transmission principle of inquiry variable, the source of an inquiry variable could be any page. Therefore, the outer loop processes every page including itself and the link objects are extracted in the inner loop. The inquiry variable must be extracted by *FetchArg* forasmuch it is an argument in a *request* method. An important point is to ascertain that the target page of the submitted value is the same as the current page *P(v).p* when the variable name is found. This is accomplished by the *SamePageName* procedure.

Global variables are visible to all pages in a site so that a page can communicate with any other page without passing arguments. In the case of ASP, the lifetime of global variables can exist in a user session or the lifetime of the application.

Many structural problems, such as broken link, long-distance path, different root, etc., may be hidden in a dynamic Web site. If a hyperlink is composed of a variable, but the variable is not given any page name or the target of a regular link is not specified, it is called a void link. A void link involving variables is represented by a virtual link object without corresponding link objects with actual target page. Recalls that any link objects created by *GatherLinkVariable* must correspond to a virtual link object with the same ID number in attribute *lvn*. Therefore, by examining the attribute *lvn* of virtual link objects, if a ID number appears only once, it is a void link. Algorithms to detect void link and broken link errors in a Web site are described in [3]. Broken links result from unavailable target pages.

3 Conclusions

A real running Web site is studied to test the feasibility of our methods. The site under study contains a total of 85 pages, and more than 80 percent of them are dynamic pages. The main scripting languages are ASP and JavaScript. There are a total of 325 hyperlinks in this site. Most are dynamic links. After analysis, two hyperlinks are found void. There are five broken links, three dynamic links and two static links. The analysis result gives Web master some helpful suggestions to improve the site.

Dynamic pages are becoming more popular because pages can reflect most updated information and can be customized much easier than a static page. However, the maintenance issue of a dynamic Web site has presented new challenges to the research community. Our contribution to Web site maintenance is in defining a set of structural analysis methods for diagnosis of some common problems to meet real maintenance needs. In the future, we plan to implement these algorithms to be able to gather and analyze dynamic Web sites automatically.

References

1. S. Chung and Y. S. Lee, "Reverse software engineering with uml for web site maintenance", in *Proceedings of the 1st International Conference on Web Information Systems Engineering*, Hong-Kong, China, June 2001.
2. G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, U. De Carlini, "WARE: a tool for the Reverse Engineering of Web Applications", in *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering,* Budapest, Hungary, 2002.
3. A. Hsieh, "StruWeb: A Mehthod for Understanding and Restructuring Dynamic Web Sites", *Master Thesis*, MIS Dept. National Pingtung University of Science and Technology, July 2002.
4. F. Ricca and P. Tonella, "Understanding and Restructuring Web Sites with ReWeb", IEEE *MultiMedia*, April-June 2001.
5. F. Ricca and P. Tonella, "Analysis and Testing of Web Applications", in *Proceedings of the 23rd IEEE International Conference on Software Engineering*, Toronto, Ontario, Canada, May 12–19, 2001.