Refactoring Web sites to the Controller-Centric Architecture

Yu Ping and Kostas Kontogiannis
Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, ON, N2L 3G1, Canada
{yping, kostas}@swen.uwaterloo.ca

Abstract

A Web site is a hyperlinked network environment, which consists of hundreds of inter-connected pages, usually without an engineered architecture. This is often a large, complex Web site that is difficult to understand and maintain. In this paper, we propose an approach that aims to restructure an existing Web site by adapting them to a controller-centric architecture. In particular, this approach is twofold. First, it defines a domain model to represent dependencies between Web pages in order to abstract current structure of the Web site. Second, it designs a system architecture as a reference model for restructuring the Web site to the new structure. These principles will be illustrated through a case study using a reengineering tool that implements the refactoring process for a JSP-based Web site.

Keywords: Refactoring, controller-centric, link type, page flow, JSP.

1. Introduction

Just a few years ago, web sites were mainly composed by a few simple and static HTML pages, which were used to share information over the Internet. The structure or the topology of the web site was easy to be discovered and maintained manually. Today, however, the rapid growth of the Internet and World Wide Web in their contents and scopes has significantly increased the complexity of the web sites. Because of such a decentralized nature of their growth, conventional Web sites usually lack a well-engineered architecture [11]. This has caused several problems for managing and maintaining such Web sites, where broken links and inconsistent page contents pose a major operational problem [2].

Therefore, to improve the maintainability and scalability of existing Web sites, a common effort is to adapt them into well-engineered architectures supported by open standards and technologies that are introduced as part of the reengineering process. By applying these open architectures, the new Web site would be more scalable and maintainable than the original one.

In this respect, to facilitate the reengineering of existing Web sites with new open structures, we present a technique to support the refactoring of Web sites towards a controller-centric architecture. The proposed technique is based on a refactoring strategy concerning page dependency analysis, page flow modeling, and implementation architecture. Specifically, we extract and classify the link information from the Web page analysis. Consequently, we group the extracted links together to form the page flow, which is represented in a XML format. Moreover, a system architecture is proposed with the aim of providing a reference model for the implementation of the refactoring task.

The rest of this paper is organized as follows. Section 2 discusses relevant work in this area. Section 3 introduces the concept about the controller-centric architecture. Our proposal refactoring strategy is described in Section 4. Section 5 provides the link classification. In Section 6, we define an XML representation for modeling the page flow according to the link classification. A general architecture of the system to implement the refactoring process is introduced in Section 7. Section 8 presents a case study used for evaluating our techniques. We summarize our conclusions in section 9 and close with pointers to future research directions.

2. Related Work

Several works on the Web site restructuring have been proposed in the literature. Ricca et. al. [5] sketch several possible Web site transforms with the aim of improving their qualities. They have classified HTML transformations into six categories: syntactic clean up, page restructuring, style renovation and grouping, improving accessibility, update to new standards, and design restructuring. A case study has been provided based on transforming original navigation structure into

HTML frames by utilizing the DMS Software Reengineering Toolkit, an integrated tools infrastructure for automating customized source program analysis and modification of large-scale software systems [19]. In another paper, Ricca and Tonella [4] propose a migration process aimed at restructuring static Web sites into dynamic ones using the software clustering technique, where a common template is extracted from the HTML pages in the same cluster, and the variable information is isolated from the template and then moved into a database. They introduce a re-engineering tool, called ReWeb, which is able to perform source code analysis and graph representation on Web sites.

These works focus on the analysis of static Web sites in which the Web pages are not generated by the server-side scripting language. By contrast, our approach includes the analysis of dynamic aspects of Web sites. We share with [5] the idea of using the reengineering technique to transform the Web site to a new structure. However, we aim to restructure the Web site by adopting a controller-centric architecture instead of the frame-based one described in [5].

There have been also some considerable activities and prototype tools on the analysis and representation the Web site by means of XML [9,15,16,20] or UML [6,7,10,18] models. The referred contributions mostly focus on modeling the entire Web site. However, much like [15,16], our approach only aims at representing relationships among Web pages. In contrast to [15,16], we support the extraction of the link information from the server side, while [15,16] only concerns the HTML documents at the client side.

Finally, there has been extensive research work conducted on the Web page/link management, such as the author-oriented link management [14] and the connectivity server [12,13]. The main difference with our work is that the approaches presented in [12,13,14] keep the Web structure untouched. Our approach achieves the page/link management in a way that restructures the Web site to a well-engineered architecture by connecting Web pages to controllers.

3. Controller-centric Architecture

The controller-centric architecture, as illustrated in Figure 1, is an architectural approach based on the Model-View-Controller (MVC) design pattern. In this approach, a controller is built on the top of a Web system to perform the central management for the client request processing and Web page forwarding. More specifically, the front-end controller provides a single entry point for intercepting HTTP requests coming from end users. It takes control of the page flow that is originally managed by individual Web pages. In other words, each Web page is not directly linked to another page. Instead, it is connected to its associated controller, and the controller then forwards the request to the target page. The page control flow is often stored either in a flat file or a database, which can be accessed by the controller in order to select corresponding Web pages to forward the request to.

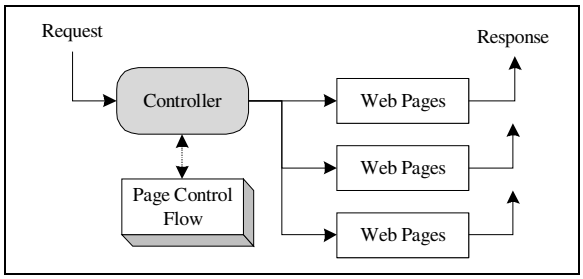

Figure 1: Controller-centric Architecture

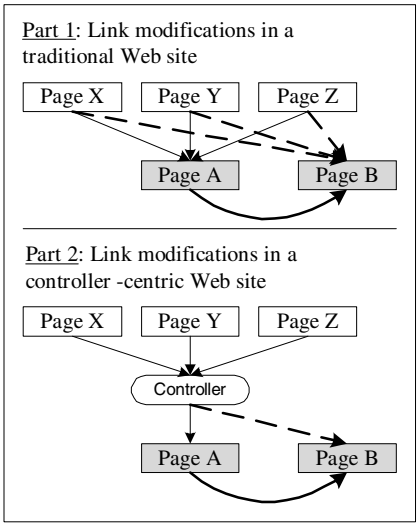

Figure 2: Page Management

To this end, the page management becomes a simpler and easier task. For example, when Page A is changed to Page B, all the links to Page A are broken until these links are updated. On a traditional Web site, as illustrated in Part 1 of Figure 2, link references are maintained by individual Web pages, where we need to search all the pages (Page X to Z) that have link references to Page A and updating these links to Page B. However, on a controller-centric Web site, as shown in Part 2 of Figure 2, we do not need to find and modify all the pages (Page X to Z) that have link references to the old page (Page A). Instead, we only need to update the page control flow information maintained by the controller, and change the corresponding link reference to the new page (Page B). Comparing to Part 1, Part 2 significantly simplifies the page modification process as well as eliminate the broken

link or the missing link, particularly in a large scale Web site.

In addition to providing the central control for the page flow, adapting Web sites to the controller-centric architecture allows us to remove potential duplication codes from Web pages. This can be achieved by including common services, such as page access authentication, to the controller. Moreover, this refactoring approach can facilitate the migration of an existing Web site to the MVC architecture [22,23], which separates business rules, presentation logic, and application control. As a result, a Web system is decoupled into three core components: the model, the view, and the controller.

4. Refactoring Strategy

To ensure that the transformation process is organized in a systematic way, a refactoring strategy for restructuring Web sites to a controller-centric architecture is formulated, as depicted in Figure 3. The activities described in the refactoring strategy are performed in a sequential manner and the whole process is divided into three major phases.

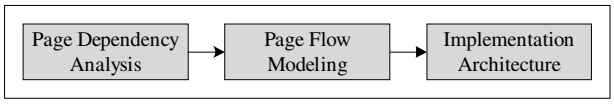

Figure 3: The Refactroring Strategy

Phase 1: Analyzing and classifying the Web page dependency

Pages and links are two fundamental elements in a Web site in which pages are interconnected by links. Each link represents a relation or dependency between Web pages. In the first phase, we focus on analyzing link information in a Web site. Furthermore, we classify the link information into several link types, such as reference link type and condition link type. With the aid of link types, Web pages can be easily grouped into clusters, and thus to provide a roadmap for the generation of controller components. Moreover, links with different types may have different transformation rules to direct Web pages to the associate controller in the new system.

Phase 2: Modeling and representing the page flow in a XML format

The preliminary requirement to reengineer a software system to a new platform is representing the source of the system being analyzed at an appropriate level of abstraction. In this context, we model the structure of an existing Web system by using an XML representation, denoted in the form of a page flow. A page flow model represents a set of relations between all the pages in a Web site. It describes the navigation path from the main pages to their various subroutines that could be branched to. In our proposed approach, a page flow model tends to focus on page links rather than page contents, where the link type is one of major elements to compose the page flow.

It is important to note that a page flow is a different concept with a page control flow because the latter is used by the controller to forward the request to the target page.

Phase 3: Designing and developing an implementation architecture

The objective of our work is for the new Web system to be adapted to modern and customizable architectural patterns such as the controller-centric architecture. Hence, in the third phase, we design and develop a system architecture with the aim of providing a reference model for the transformation of existing Web sites to a controller-centric structure. This general refactoring architecture consists of four implementation steps, including page preprocess, link information extraction, control integration, and link rewriting.

In the following sections, we provide further discussion about page dependency analysis, page flow modeling, and implementation architecture, respectively.

5. Page Dependency Analysis

A link represents a connection from one Web page to another. It is directed and thus has two endpoints, the source from which the link starts, and the target where the link ends [8]. A link type is used to describe a relation or a dependency between the source and the target of a link.

Moreover, a Web page may contain several intertwined languages dealing with different aspects of the system. These typically involve client side elements processed on the browser, such as HTML, plug-ins, and JavaScript, as well as server side elements running on the Web server to generate dynamic contents, using JavaServer Pages (JSP) or Active Server Pages (ASP) for instance. Therefore, the actual implementation method for each link type is really dependent of which programming languages are applied to the Web system.

In this section, we describe two categories of links, which are reference link and condition link, as well as possible implementation methods used in either static or dynamic Web pages.

5.1 Reference Link

We classify links into the reference type according to how a target page is referenced by a source page. The

reference link has four different types: inner link, handover link, include link, and invocation link (Figure 4).

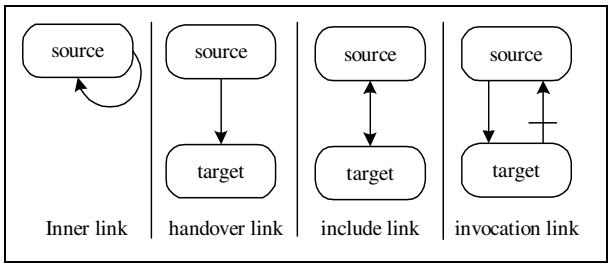

Figure 4: Reference Link

Inner Link

An inner link represents that both source page and target page are a same Web page. When an inner link is selected in the source page, it does not connect to a new page. Instead, the target page that is the destination of the inner link is still the same page as the source page, but most likely, may have different output to the client based on the request, or just moves the focus from one position to another one within the same Web page.

A typical link of this sort in a Web page is the link (we name it as static inner link), which starts at the source page and points to a new position in the same page. For instance, the links connect a table of contents to each chapter that following it in the same page. Another type of the inner link is a dynamic inner link. When activating a dynamic inner link, the source page actually sends a request to the server, even though the destination of the link is the same page as the calling page. An example of a dynamic inner link is a link in a JSP page, which serves to validate the client input in the server side. The JSP page points to itself each time after clients select the link to submit their data. It checks the data and displays the error message if any until there is no mistakes in the input.

The static inner link can be ignored in the refactoring process because the link action only makes the windows to focus on a different position in the current page, and the page itself keeps static all the time. The dynamic inner link has different behavior. It involves the server side process and changes the state of a Web page. Thus, we include the dynamic inner link in the prefactoring process.

Handover Link

A handover link represents that a source page transfers the page control to a target page when activating a link. The target page cannot be the same one as the source page or a part of the source page. It disconnects any connections with the source page after the handover link is selected, and thereafter, takes over the page or application control and starts to process the user or program's request. In another hand, the possession in the source page is permanently terminated as soon as the page is connected to the target page, and the source page does not expect any response from the target page.

The handover link is a very common link type used in Web pages. There are some sample implementations of the handover link:

Navigation: Navigation link is a link defined by the <*A*> element with specifying a URL value for the *href* attribute in a HTML document. For example, a source page has a handover link to target.html: <A href = "target.html"> Handover Link </A>.

HTML Form: A HTML Form, <FORM action = *relativeURL* method = *MethodName*>, allows the user to enter the information and sends them to the target page specified by the *action* attribute for processing.

JSP Forward Action: A JSP Forward Action, <jsp:forward Page = "*relativeURL*" />, dispatch the client request information from the current page to another resource, which can be a HTML page, a JSP page or a Servlet in the same context as the forwarding JSP page. The forward action effectively terminates the execution of the current page, and the source codes after the <*jsp:forward*> element are not processed further.

Include Link

An include link represents a relation that a source page includes a target page when activating the link. The target page can be a static page (HTML file) or a dynamic page (JSP file). If the target page is static, its whole content is included in the source page. If the target page is dynamic, it acts on a request and sends back the results to be included in the source page. When the include action is finished, the source page continues processing the remainder of the page.

A static include link in a JSP page is an Include Directive. By contrast, an example of dynamic include link in a JSP page is an Include Action.

Invocation Link

An invocation link represents a communication link between a source page and a target page. The link information is actually encapsulated in a function or a class, which can be called by a source page in order to read from and to write to a target page.

In general, an Internet connection is initialized before a actual connection to a target page is made. The possession of the invocation link is similar to the dynamic include link: a source page sends the request to a target page by invocation of a function or a class which implementing the invocation link, and the target page acts on the request

and sends back the results to the calling page. The source page continues processing the remainder of the page after the connection is made to the target page.

An example implementation of an invocation link is a link that sets up an http connection in a JSP page by using the Java API, *HttpURLConnection.*

5.2 Condition Link

Another classification of links, condition links, describes the relations according to whether or not the connection between a source page and a target page depends on certain conditions. Therefore, we can define two link types of the condition link: the conditional link and non-conditional link (Figure 5).

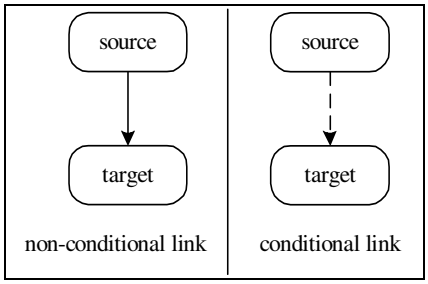

Figure 5: Condition Link

Conditional Link

The conditional link represents that there is a condition or some conditions on which a source page depends to make a connection to a target page. In the other word, the conditional link in a source page is activated only when meeting certain criteria. It is obviously that all of the links within a control statement, such as IF-ELSE, SWITCH-CASE and DO-WHILE, are conditional links. Whether or not a source page includes a link to a target page is decided at run time.

Non-conditional Link

The non-conditional link represents that there is no any conditions, where the source page depends on to make the connection to the target page. The source page always contains the link to the target page. The non-conditional link should not be found within any control statements.

In addition, the classification and the relationship of the link type can be represented in a UML model shown in Figure 6.

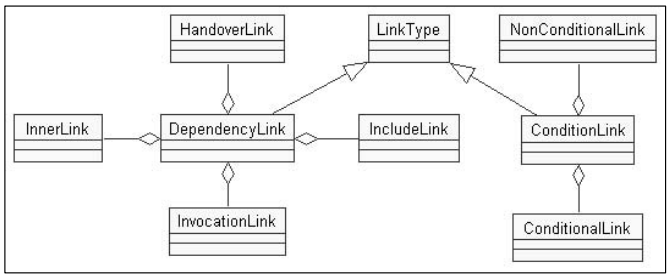

Figure 6: UML Model of Link Type

6. XML Representation Model for Page Flows

A basic consideration of building an information model for the page flow is to express it in a language that can describe the interconnection between Web pages in a general and formal way. XML has been widely recognized as the representation of the Metadata on the Web. It provides a unified framework for annotating structured data. It is extensible, and can define meaningful tags that link syntax with the semantics of the entities of a given domain. Therefore, we use XML as the language to represent the page flow in a Web site.

Another consideration for the representation model is to describe it in a standard way. This can be accomplished by creating an XML DTD document that defines the XML structure with a list of legal elements and attributes. Figure 7 illustrates the page flow DTD and its elements and attributes are explained in the following.

From the definition of the page flow, the page and the link are two basic building blocks of the page flow. As a result, a page flow DTD document mainly contains the information about pages and links.

The root of a page flow document is a *pageflow* element. It can have an arbitrary number of *sourceset* sub-elements and no attributes. The use of an asterisk * in the DTD reflects the non-mandatory constraints for these roles.

The *sourceset* element specifies a collection of source pages where the links start. The source pages are grouped together into a single source set if they reside in the same directory. The information about these source pages is described in the *source* element, the sub-element of the *sourceset* element. The *sourceset* element includes two attributes that declare the file directory: the *path* attribute that represents a local directory, and the *uri* attribute that represents a Web directory (URI) mapping to the local directory.

Using the directory structure to cluster the source pages induces two benefits. First, it allows shorter path name contained in the sub-elements of the *sourceset* element. In this context, the corresponding attributes in the sub-elements use relative path names instead of

absolute path names. The relative path name need not specify all the components of a local path or a URI. The missing value is inherited from the fully specified path (the path attribute in the *sourseset* element) or URI (the *uri* attribute in the *sourseset* element). This approach can significantly reduce the size of generated XML files. Second, it provides a possible solution to build up the controller components based on the directory hierarchy. A controller can be created to manage the Web pages either in the same directory or having a common parent directory.

```
<!ELEMENT pageflow(sourceset*)>

<!ELEMENT sourceset(source*)>
<!ATTLIST sourceset
                path        CDATA        #REQUIRED
                uri         CDATA        #REQUIRED>

<!ELEMENT source(target*)>
<!ATTLIST source
                name        CDATA        #REQUIRED>

<!ELEMENT target EMPTY>
<!ATTLIST target
                name        CDATA        #REQUIRED
                uri         CDATA        #REQUIRED
                condition   ( yes
                            | no )       "no"
                reference
                            (inner
                            | handover
                            | include
                            | invocation )   #IMPLIED
                method      ( link
                            | form
                            | frame
                            | object
                            | javaconnection
                            | jspforward
                            | errorpage
                            | include
                            | jspinclude )   #IMPLIED>
```

Figure 7: Page Flow DTD

The *source* element specifies the page and link information for a Web page that exists in the directory denoted by the *sourceset* element. It has a *name* attribute whose value is the local file name of the source page. For example, if the directory of the source set is "C:\workspace\myweb\jsp", the value of the *name* attribute would be "index.jsp" that is a shorthand for the full path name "C:\workspace\myweb\jsp\index.jsp".

The *source* element has a *target* sub-element, which represents a collection of the target pages that the specified source page points to. The *target* element has five attributes: *name*, *uri*, *condition*, *reference*, and *method*. The *name* attribute indicates the file name of the target page without including the Web directory. The Web directory, where the target page resides, is specified by the *uri* attribute. It can be either a relative path name or an absolute path name depending on whether the target page is in the same context as the source page. The relation or the link type between the source and target pages is declared in the *condition* attribute representing the condition link, and the *reference* attribute representing the reference link. If the link is the conditional type, the value of the *condition* attribute will be "yes", whereas the value will be "no" (default) if the link is the non-conditional type. The *reference* attribute can have a pre-defined value that is, "inner", "include", "handover", or "invocation". The actual use of the link types is specified by the *method* attribute that has a set of pre-defined values. Figure 8 illustrates a sample XML document using the page flow DTD.

```
<?xml version="1.0" ?>
<pageflow>
    <sourceset path="c:\workspace\myweb\jsp"
               uri="http://localhost/myweb/jsp">
      <source name="category-edit.jsp">
        <target name="category-index.jsp" uri="*"
                condition="yes" reference="handover" method="link" />
        <target name="category-edit.jsp" uri="*"
                condition="no" reference="inner" method="form" />
        <target name="header.jsp" uri="\commonfiles\"
                condition="no" reference="include" method="include" />
        <target name="header.jsp" uri="\commonfiles\"
                condition="no" reference="include" method="include" />
      </source>
    </sourceset>
</pageflow>
```

Figure 8: Page Flow XML

7. System Architecture

In this section, we describe the system architecture for the overall refactoring process. The proposed architecture aims to provide a reference model presenting major components and implementation phases to be applied for refactoring a Web site towards a new controller-based system. As a result, the target system would be more maintainable than the original system. In the following we will discuss the general architecture of the refactoring system as this is depicted in the Figure 9.

The first step, the Page Preprocess, in our approach is to perform preliminary processing on a Web site in order to make Web pages ready to be analyzed in the second step. In particular, this step focuses on identifying the control block in Web pages. The control block can be recognized as either control statements such as IF-statement, WHILE-statement, and SWITCH-statement, or custom tags that implement the control logic. Any links found in the control blocks are conditional links according to the categorization of the link type. Moreover, it discovers any implicit link information that cannot be directly handled by using the pattern matching technique in the following steps, specifically, the links inside Java codes, VBScript or JavaScript. Then it makes these

implicit links to be explicitly specified by using a pre-defined method.

There are some possible solutions to implement Web page preprocessing tasks. One possibility is to create a page analyzer, which usually contains a parser and a lexical analyzer for the Web programming language [1]. It aims to represent the page source at an appropriate level of abstraction, for instance, in the form of an Abstract Syntax Tree (AST) [21]. Therefore, the linkage information can be easily obtained by accessing the generated AST. JSP pages involve several intertwined languages dealing with different aspects of the system, such as JSP elements, Java codes, HTML and JavaScript. This makes the implementation of a page analyzer a tedious and complicate work.

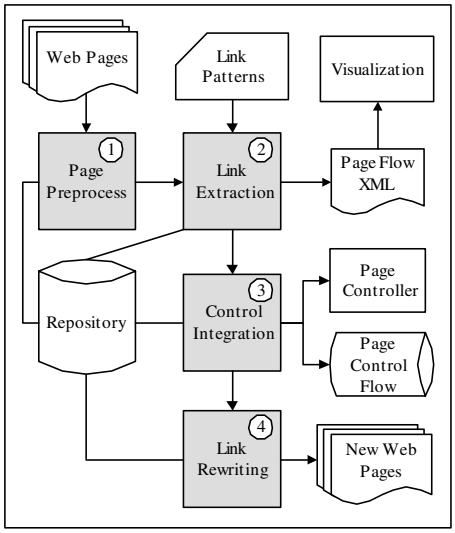

Figure 9: System Architecture

Another solution to preprocess Web pages is to use a lightweight preprocessor instead of a heavyweight analyzer. Since the page preprocessing aims to identify the control statements and locate implicit links, a page preprocessor can be developed by focusing on the analysis of this information included in Web pages and ignoring irrelevant elements. Generally, the denoted page preprocessor will bypass HTML and custom tags in the Web pages because the control statements and linkage information contained in these objects can be easily matched by the specific lexical patterns, which are used in the link information extraction in the next step. But such information cannot be directly mapped to the pre-defined patterns if they are included in Java codes, VBScript, or JavaScript. Therefore, the page preprocessor only need to recognize the corresponding representation of the control statements and implicit links inside the Java codes, VBScript, or JavaScript, and consequently make them pattern-ready format. Thus the generation of such a page preprocessor is much simpler than that of a full-function page analyzer.

The Link Information Extraction step aims to analyze the preprocessed Web pages and to extract the embedded link information by using the pattern-matching technique. The lexical patterns, which are used to represent link relations, are stored in the Link Pattern component, and may be defined in different forms of expressions according to the actual implementation of Web programming language specifications. Figure 10 shows some examples of link patterns used in a JSP-based Web site. The Link Information Extractor, the implementation program of the Link Information Extraction step, will create an XML file to include the discovered link objects for each Web page. Then it will combine these newly generated XML file into a single XML file to represent the page flow for the whole Web site. Such XML file should conform to the page flow DTD document defined in the previous section. As an extension of the Link Information Extraction step, the page flow XML file can be translated to an appropriate graphic language and thus the current Web site structure can be displayed by using the associated graph visualization tool.

```
//JSP LinkPatterns

//JSP Forward
jspforward=<jsp:forward[\\s]+page=[\"][\\s]*[^\"]+[\"][\\s]*((>)|(/>))

//JSP Include
jspinclude = <jsp:include[\\s]+page=[\"][\\s]*[^\"]+[\"][\\s]+flush=
             [\"][\\s]*[^\"]+[\"][\\s]*((>)|(/>))

//Java Bean
javabean = <jsp:useBean[\\s]+id=[\"][\\s]*[^\"]+[\"][\\s]+scope=
           [\"][\\s]*[^\"]+[\"][\\s]+class=[\"][\\s]*[^\"]+[\"][\\s]*((>)|(/>))

//Include
include =<%@[\\s]+include[\\s]+file=[\"][\\s]*[^\"]+[\"][\\s]*%>

//TagLib
taglib = <jsptag:choose>.*?</jsptag:choose>

//JSP Plug-in
jspplugin = <jsp:plugin.*?</jsp:plugin>

//HTMLHref
href=<a[\\s]+href=[\"][\\s]*[^\"]+[\"][\\s]*>

//HTMLForm
form=<form[\\s]+method=[\"][\\s]*[^\"]+[\"][\\s]+action=[\"][\\s]*[^\"]+[\"][\\s]*>

//HTMLFrame
frame=<frame[\\s]+src=[\"][\\s]*[^\"]+[\"][\\s]*>
```

Figure 10: JSP Link Patterns

During the Control Integration step, controller components of the new architecture is generated, as well as a corresponding page control flow is created according to the control structure. This can be accomplished by applying certain software clustering techniques based on the analysis of the page flow identified in the Link Information Extraction step. A server side program, such as Java Servlet, then implements the functionalities of the controller components. The newly created page control

flow can be stored in either a flat file in an XML format or a database system.

The last step of the refactoring process is referred to the rewriting of the original Web pages by conforming to the generated control-centric architecture. This is performed by modifying the link information included in Web pages. As a result the destination of the link in a source page is replaced by an associate controller specified in the page control flow.

The above components store and share the results produced over the refactoring process in the Repository. In addition, the Repository contains the information related to the file directory, the Web site, the custom tag library, and other useful data, which are input from users or generated by other tools. The Repository can be either a database system or a set of collections of flat files.

8. A Case Study

The strategy and the architecture described in previous sections have been used in an experiment to restructure a JSP-based Web site, which was originally developed to publish photography information to the Web. The Web site allows the user to browse and manage photographs by the categories or the location. It is implemented by using a free software, called PhotoDB, which is mainly composed of JSP pages and HTML pages [17].

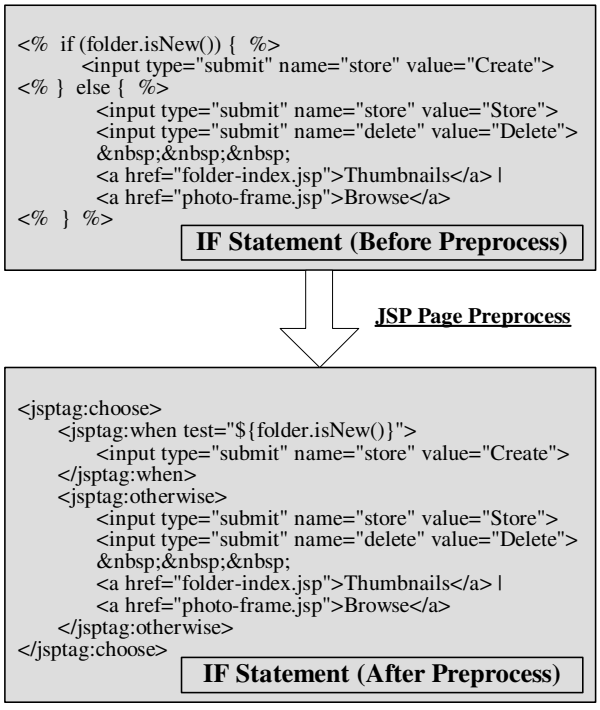

Figure 11: JSP Page Preprocess

We developed a reengineering tool used to support the refactoring process. Such a tool consists of four components, JSP page preprocessor, link information extractor, control integrator, and link rewriter, which are designed in accordance with the system architecture previously presented.

During the step of the JSP page preprocess, the source code was analyzed for identifying the Java control statement included in the JSP page. We converted these statements to predefined JSP control tags, for instance, using *<jsptag:choose>* tags to replace JSP IF-Statements (Figure 11). Therefore, we can use a simple pattern form to match the control statement in the link information extraction step.

The link information extractor firstly examined conditional and non-conditional links, where the former was inside control blocks marked by JSP control tags, and the latter as not. Then, it analyzed each link in order to determine the link type and the implementation method. Consequently, it exported results into a page flow XML file. Moreover, by using XSLT, we transformed the page flow file to the dotty format, which is a graph layout representation from AT&T Lab [3].

```
<?xml version="1.0" ?>
<controlflow>
  <controller uri="/servlet/WebController">
    <sourcepage id="001" uri="/jsp/folder-edit.jsp">
      <targetpage id="002" uri="/jsp/header.jsp"/>
      <targetpage id="006" uri="/jsp/category-edit.jsp"/>
      <targetpage id="008" uri="/jsp/login.jsp"/>
    </sourcepage>
  </controller>
</controlflow>
```

Figure 12: Page Control Flow

In the third step of the process, JSP pages need to be clustered together when they share common attributes, such as the structure, the connectivity, or the content. However, the clustering technique for JSP pages is outside the scope of this paper. Thus, we did not include any clustering algorithms in the refectorying tool. Instead, we grouped all JSP pages into one cluster, which was managed by a single controller, namely, WebController. In addition, the tool assigned each page a unique id that was used by the controller to find the corresponding target page. Figure 12 shows a sample control flow file for the testing Web site.

In the last step of process, the tool reviewed the linkage information included in the JSP page and replaced the target page with the corresponding controller along

with a parameter to indicate the page id. An example of these changes is illustrated in Figure 13.

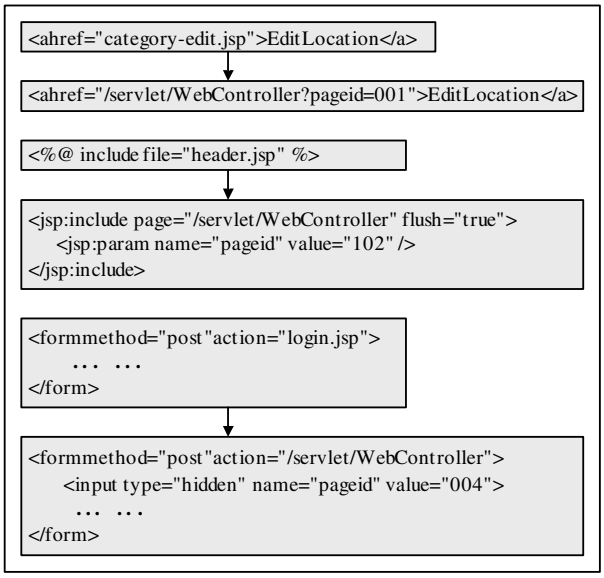

Figure 13: Link Rewriting

In conclusion, the reengineering tool can successfully refactore the Web site to a controller-centric one. Specifically, from the end user's point of view, the resulting Web pages created by the new system are exactly the same as the original ones in the original environment. In addition, this case study illustrates that the proposed framework can adapt to real life Web sites. The whole refactoring process is algorithmic and automated, except the JSP pages preprocessing and the generation of JSP pages clusters, where the human assistance is required to identify dynamic links and possible JSP page groups. However, the proposed tool can be further extended to support dynamic link analysis and page clustering in the refactoring process. Finally, the new Web site has been simplified from the original one. The structure comparison between the new Web site and the old one is not presented here because of the space limitation.

9. Conclusions and Future Work

The accelerated development of Web sites and the fast growth of associated Web technologies have resulted in a variety of maintenance concerns. One of the major maintenance problems is the adaptation of existing Web sites into a well-engineered architecture. In this paper, we have addressed this problem by applying reengineering techniques to restructure Web sites towards a controller-centric architecture. Specifically, we extracted the link information from Web pages and classified into difference categories, and consequently represented as XML models. In addition, we proposed a system architecture to provide a reference model for the implementation of the refactoring task. Moreover, the technique discussed in this paper has been evaluated with a reengineering tool that was implemented and tested on a JSP-based Web site.

Future extensions on the work presented in this paper may focus on the following two directions. The first direction is on investigating the usage of a software clustering technique with the purpose of identifying cohesive groups of Web pages. A particular clustering algorithm needs to be developed in order to provide the support for the control integration of the migrant applications. The second direction is on developing a technique for automatic implementation of analyzing dynamic Web components that can only be formed at run time, such as HTML form filling, and database querying.

Acknowledgments

We would like to thank Ying Zou of Queen's University, Terry Lau and Tack Tong of IBM Canada Laboratory, Jianguo Lu of the University of Windsor, Joe Wigglesworth of IBM CAS, John Mylopoulos of the University of Toronto, as well as all the anonymous reviewers for their valuable suggestions, comments, and insights.

This work was funded by IBM Canada Ltd. through IBM Centre for Advanced Studies and by the Natural Sciences and Engineering Research Council of Canada.

Trademarks

References

[1] CAST, CAST JSP Analyzer, http://www.castsoftware.com

[2] Cornelia Boldyreff and Richard Kewish, "Reverse Engineering to Achieve Maintainable WWW sites", IEEE 2001.

[3] Dotty, AT&T, http://www.research.att.com/sw/tools/graphviz/

[4] Filippo Ricca and Paolo Tonella, "Using Clustering to Support the Migration from Static to Dynamic Web Pages", in Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03), May 2003, pp. 207-216.

[5] Filippo Ricca, Paolo Tonella, and Ira D. Baxter, "Web Application Transformations Based on Rewrite Rules", *Journal Information and Software Technology*, Vol. 44(13), 2002, pp. 811-825.
[6] G.A.Di Lucca, A.R.Fasolino, F.Pace, P.Tramontana, and 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 (CSMR'02), 2002.
[7] G.A.Di Lucca, M.Di Penta, G.Antoniol, and G.Casazza, "An approach for reverse engineering of web-based applications", IEEE 2001, pp. 231–240.
[8] HTML 4.01 Specification, W3C Recommendation. http://www.w3.org/TR/html4/.
[9] I.Varlamis and M.Vazirgiannis, "Web document searching using en-hanced hyperlink semantics based on xml", IEEE 2001, pp. 34–43.
[10] Jim Conallen, "Modeling web applications architecture with uml". White paper, Rotional Software, June 1999.
[11] Jon Kleinberg and Steve Lawrence, "The Structure of the Web", *Journal Science*, Vol. 294, November 2001, pp. 1849-1850.
[12] Keith H. Randall, Raymie Stata, Rajiv G. Wickremesinhe, and Janet L. Wiener, "The Link Database: Fast Access to Graphs of the Web", in Proceedings of the Data Compression Conference, 2002.
[13] Krishna Bharat, Andrei Broder, Monika henzinger, puneet Kumar, and Suresh Venkatasubramanian, "The Connectivity Server: Fast Access to Linkage Information on the Web", *Computer networks and ISDN Systems*, Vol. 30, 1998, pp. 469-477.
[14] Michael L. Creech, "Author-oriented Link Management", *Computer networks and ISDN Systems*, Vol. 28, 1996, pp. 1015-1025
[15] Olga De Troyer and Tom Decruyenaere, "Conceptual modelling of web sites for end-users", World Wide Web, Vol. 3, 2000, pp. 27–42.
[16] Olivier Liechti, Mark J. Sifer, and Tadao Ichikawa, "Structured graph format: Xml metadata for describing web site structure", *Computer Networks and ISDN Systems*, Vol. 30, 1998, pp. 11–21.
[17] PhotoDB, http://www.magiccookie.com.
[18] 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, June 2001.
[19] Semantic Designs, Inc., DMS Software Reengineering Toolkit, http://www.semdesigns.com /Products/DMS/DMSToolkit.html
[20] Stefano Ceri, Piero Fraternali, and Aldo Bongio, "Web modeling lan-guage (webml): a modeling language for designing web sites", 2000.
[21] Ying Zou, Kostas Kontogiannis, "A Framework for Migrating Procedural Code to Object-Oriented Platforms", in Proceedings of the 8th IEEE Asia-Pacific Software Engineering Conference (APSEC), Macau, China, December 2001, pp. 408-418.
[22] Yu Ping, Jianguo Lu, Terence C. Lau, Kostas Kontogiannis, Tack Tong, and Bo Yi, "Migration of Legacy Web Applications to Enterprise Java Environments – Net.Data to JSP Transformation", in Proceedings of CASCON 2003, October 2003, pp. 121-135.
[23] Yu Ping, Kostas Kontogiannis, and Terence C. Lau, "Transforming Legacy Web Applications to the MVC Architecture", in Proceedings of the Software Technology and Engineering Practice (STEP 2003) International Conference, September 2003.