International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 12 | Dec-2017 www.irjet.net p-ISSN: 2395-0072

Transformation of Java Server Pages: A Modern Approach

A.K.Ratha[1], S.Padhan[2], S.Mohanty[3]

[1]Asst.Prof, Dept of CSE, Vikash Institute of Technology, Bargarh, Odisha, INDIA
[2,3]Student Researcher, Vikash Institute of Technology, Bargarh, Odisha, INDIA

Abstract - *A web application maintains two things: presentation and the logic. The maintainability is improved by making it separate which makes the web application easier to evolve and maintain. It also allows developers with different skills to cooperate more efficiently. There are custom tag in JSP which provides separation. In this paper , we approach a modern transformation to restructure JSP by moving embedded Java Code into custom tags without changing the original application. It helps in reducing complexity which makes the application more maintainable.*

Key Words: scriptlet, tag, elements, transformation, javabean.

1. INTRODUCTION

A web application can be develop with the help of HTML code and application code which is written in different language like Visual basic, java ,JavaScript. HTML is being embedded by application code and setout of data has been used by HTML.

For creating dynamically generated web pages a technology is used known as Java Server pages(JSP).java is used as scripting language by JSP. The form <%Java code %> is used by the scriptlets, that are used to embed java code within HTML. A part of Java-code embedded in the HTML like JSP code. Everything inside the <% %> tags is called as scriptlet .It accelerate prototyping by it's explicit use and establish complexity into the implementation.HTML associate with java code by the help of this scriptlets, which leads to problems in code authentication and debugging, that faces difficulties in software maintenance and evolution. As these scriptlets are not reclaimable. so there is a chances of duplication while cuts and paste edits between pages, which gives us error environment.

The combination of HTML and Java helps to differentiate the presentation and business logic. This type of separation helps in both easier maintenance of web application and allows different skills developers to co operate effectively. A user defined JSP language element is commonly known as custom tag.

2. BASICS ABOUT JSP

JSP elements and templates make JSP pages. Template texts are those content which are not a JSP elements. They may be any text like HTML, XML, WML or plain text that can directly passed through to browser. Dynamic content can be constructed with the help of JSP elements. It has custom actions, standards tag library tag(JSTL), directives, standard actions, scripting elements and JavaBeans components. Prefix JSP like <jsp:useBean> and <jsp.getProperty> action are uses by the standard actions. For creating bean , access bean property and invoke other pages these actions are used . Many actions can also covered by custom action and JSTL. Custom tag library has two components i.e. XML file and implementation of tags in java. The XML file is also known as tag library descriptor. Jsp page is translate in to servlet when it contain custom tag and the tag is converted to operation on an object called tag handler.

Tag handler handles the behavior of the tags that must implemented on one of the interfaces defined in the package like javax.servlet.jsp.tagext.

Mapping is done in between tag library and each tag to the appropriate tag handler class. With the help of taglib directives, tags are made available within a jsp page. In fig1 we draw the relationship between taglib directives in a jsp file,TLD and tag handler class.

In the figure, the taglibbrary xxxlib has a local name within the JSP file of mylib and the custom tag is named cdtitle. The implementation of the tag is in the file CdtitleTag.class.

3. THE TRANSFORMATION

The section presents comes nearer to reconstructing jsp pages by transforming interweaved java code into tag to modernize existing jsp web application.

3.1. Requirements

Three major requirements are there, in the implementation of the reconstructing whose details are as follow,

The functionality of web application should not change by the transformational restructuring. To meet this necessary condition static analysis information such as data dependence and control flow must be concentrated.

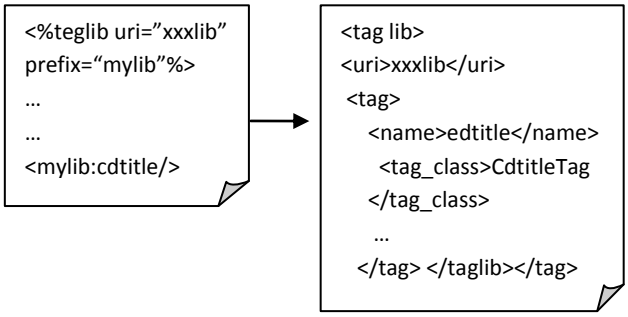

Fig-1:The taglib directive

The complex instruction among different parts of a jsp page should still be express even after the parts written in java are progress into custom tags.

The user interface of the application which is determining in part by the HTML and in part by the java and JavaScript should remain fixed.

All code comments should be store for future continuance it is unavoidable that the application will require to be continued and develop after it is transformed, such as attaching new features and making extra improvements. Thus the improved application code with comments clean out of the original application may not be preferable.

The code comments must be store in the same place comparative to the original source code in order to make sure that the code is still understandable.

3.2 A multilingual parser

Parsing can be done before the transformation of source code. Most of the browsers ignored minor syntax error which is carried out by source code due to mixing of multiple languages. For this we consider a parsing technique which is developed by N.Synytskyy et al. This is a powerful multilingual parsing technique for ASP applications which is based on island grammars, and extended to JSP by Arial Li. This multilingual parse used to parse multiple language in to a single parse tree ,that helps in code Analysis fact extraction & transformation.

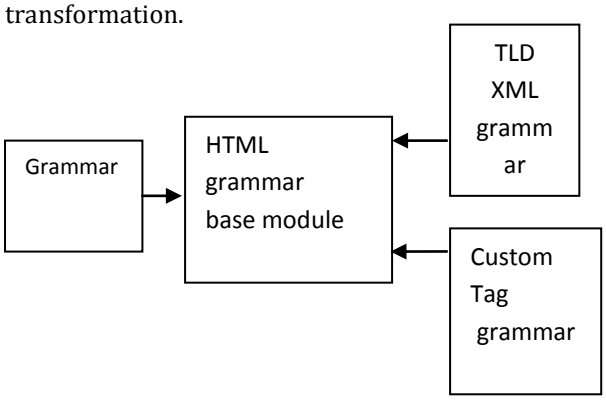

Fig-2:Multilingual Parser

Relationship between grammars is shown in fig 2 HTML grammar is the root grammar which describes the structural element of HTML. It consider different HTML element i.e table, style & appearance element, Anchor element, Image element etc. In JSP grammar, In modular form the TLD, XML grammar & custom tag grammar are written.

Maintaining the relationship between scopes of multiple source languages is the major characteristics of the grammar. For example, A loop statement that consist of HTML text within a block is parsed as single scriptlets not as two separate scriptlets. Here the '%>'tag starts HTML text and '<%'tag ends HTML text.

4. The Approach

Here we consider 4 approach, they are ,

a) JSP Scripting Elements
b) HTML content embedded in out.print()
c) JavaBeans action elements
d) JSP page directives

JSP Web application is the existing one which is tightly coupled with HTML code & Java code. Java code is embedded in HTML code & HTML tags are embedded in Java code. First we have to cheek the code transformation that, which elements are being affected by the transformation.

a) JSP Scripting Elements

Inserting of java code into servlet that will be generated from jsp page can be authorized by jsp scripting elements. These are 3 forms,

- Formal declaration: Declaration can be done by either methods or variables.
- Expressions: A value is return by the expression elements which is written to HTML page
- Scriptlets: It consists of a number of languages elements, variables or method declaration.

b) HTML content embedded in out.print()

For generating HTML content from within a scriptlets a method is used i.e. out.print(....);. It has two disadvantages,

I. Extra effort is needed to create and maintain HTML pages for application programmers

II. Embedded code must be understood by the web page designer

We extract HTML content from out.print() and out.println() statements before we go for transformation and then put the content directly into the page.

c) JavaBeans action elements

A no argument constructor is used in jsp i.e. a java bean component. It is a java class. Jsp

Standard action elements permits the developers to use JavaBeans components which are categorized as 3 types,

1- *<jsp:useBean>* :it represent a JavaBean and makes it accessible in a page.

2- *<jsp:get Property>:* it gets a property value from a JavaBean and adds it into the response .

3- *<jsp:setProperty>:* it is used to set all properties valuesin a JavaBean. It matches the names of the parameters received from the request. It is not mandatory to know how to set properties of the bean ,by the page designer. The page designer only focus on JavaBean components in application. In which a different servlet represent a bean and passes it into jsp pages for display.
Before the general transformation we change the 3 action elements into valid java code enclosed within scriplet tags <%.....%>

4- *jsp page directives*
These are present at the top of a jsp page and enclosed within directive tags (<%@.....%>). The number of page directives is not fixed in jsp pages but it should have unique attribute/value pair. The transformation is affected by the cases are the case where a java packages is inserted by page.

Eg.

<%@ page import ="java.sql.ResultSet" %>

This directive imports the class ResultSet from the package java.sql into the page. As this class should also be imported into our newly created custom tags, we change the page directive into a scriptlet. For example, the directive above becomes <% import java.sql.ResultSet;%>. We can then transform this valid import statement as part of the general transform.

5. THE GENERAL TRANSFORMATION

The segment looks at the general transformation needed to convert installed java into custom tags we present the 4 general expression that contain the transforms.

Before the general transformation the original source code must be rearranged by modifying non scripting elements (bean, page directives) that need to be relocated into jsp scripting elements extracting HTML content from out.print.statements and merging adjacent scriptlets.

After the modify transform all java code is fixed inside jsp scripting elements. The next goal is to eliminate jsp scripting elements from the normalized source code by replacing them with jsp custom tags.

The intermix of HTML and code presents a challenge not only for parsing but also for transformation. By using multiple programming and technology the dataflow analysis through various software pieces is complicated. For the transformation of embedded java code into appropriate custom tags. We have to find out how to divide the processes of jsp pages into smaller components.

(1) How to use new custom tags for better control of dynamic content,

(2) How to name the tags,

(3) How to pass input through by placing data either in tag attributes or between opening and closing

(4) How to devise the tags that focus on the "*what*" and hide the "*how*" to make the transformed web pages and resulted tags maintainable

Transformation strategy categorized into 4 basic cases,

Case 1

Only one custom tag is needed

In figure 3, one block of java code is enclosed with HTML but no nesting is done in HTML segment. Here only one custom tag has to be created into which all java code is relocate. It has an empty body. In the eg. Java code is handle by the user session management. Hence it can be named as <mylib:user Session> and java class havingthe name as UserSession tag. In figure 4, the transformed page after javacode is migrate into the custom tag <mylib:UserSession>.In figure5 two block of java code in HTML has been shown. These two blocks are separated by HTML water elements("welcome!"). it's parse tree demonstrate that both two block are at same scope level. These two java code blocks can be merge into one custom tag , which has a non empty body and consist of HTML water elements, which is shown in figure6.

Case2

Two nested custom tag are needed

Now consider two custom tags .in figure7, a jsp expression is used i.e. <%=userName%>.here a variable is used within the expression that is defined in the javacode block. These two nested custom tags are created in such a way that one tag is nested within the body of another tag. The java code in the block will be move into outer custom tag and the expression will be migrating into simple tag.

```
<html><body>
<%
HttpSession s = request.getsession (true);
Octs2.webLogUserImp entry = new octs2.WebLogUserImp ();
String UserId = ((string s.getValue ("userId")).trim ();
String username = entry.getUser (userId);
s.putValue ("UserName",UserName)
%>
Welcome!
</body></html>
```

Fig-3:Block of Java Code in HTML

```
<html><body>
    <my lib:userSession></my lib:userSession>
  Welcome!
  </body></html>
```

Fig-4:Block After Transformation

After scriptlet and expression are relocated into custom tags <mylib:userSession> and <mylib:userName>respectively ,a transformed page is generated which is shown in the figure 8. Here only nesting of tags are done. While the classes that implement the tags are not. To maintain the relationship between parents and child class gets and set method must provide by parents for variables that are accessed by child class.

The code is generated in child tag class for obtaining a reference to instance of class for parent tag. By using gets and set method of parent class the child can access the variables.

The outer custom tag is same as case1, expert that this outer tag has a body which contains not only a segment of HTML code but also another tag <mylib:userName>.

Case 3

HTML content depends on choice

In figure 9, a jsp page is present in which a choice statement determines if some HTML content is displayed. As we know a choice statement maybe an if-else statement or a switch case statements. The HTML content is nested within choice and enclosed between a end tag(%>) and start tag(<%).

The nested HTML content after parsing will be used by our grammar as an entity within the if-else statement. The parse will found one block of java code at the top scope in figure 10,that start from first "<%" to last "%>" tag.

For this case one parent tag and two children tags are created. Within the body of parent tag the two children tags are nested and as they are sibling they deployed side by side in the body of parent tag. The then part of choice and else part will handle by one tag and another child tag respectively. If the condition is true then child will allow the HTML content.

```
<html><body>
<%
HttpSession s = reqest.getsession   (true);
Octs2.WebLoqUserImp entry =new octs2webLogUsersImp();
String userId = ((string s.getValue (:userId")).trim();
String UserName = entry.getUser (UserId);
%>
Welcome!
<%
s.putvalue ("username",username);
%>
</body></html>
```

Fig-5: Two blocks of Java Code in HTML

```
<html><body>
    <my lib:UserSession>
  Welcome!
    </my lib:userSession>
</body></html>
```

Fig-6:Two block After Transformation

At the top scope or parent tag a new Boolean variable is declared that will store the evaluation of choice expression. By using this Boolean variable if statement is replaced by two if statement. For this variable a get method is also declared in parent tag. At last all java code in top scope that include all declaration of new variable is move into parent tag (i.e <mylib:chock login>). The java code wrapped with an appropriate if condition expression should enter its children tags (<mylib:invalid login> and <mylib:valid login>)

```
<html><body>
<%
HttpSession s = request.getsession (true);
Octs2webLogUserImp entry = new octs2.webLogUsersImp ();
String user Id = ((string) s.getValue  ("userId").trim();
String username  = entey.getuser (userId);
%>
Welcome!    <%= username  %>
</body></html>
```

Fig-7: A Page with an Expression

```
<html><body>
Welcome!
</body></html>
<mylib:userSession>
<mylib:userName/>
</mylib:userSession>
```

Fig-8 Expression Page After Transformation

Another feature of transformation is shown in figure 11. Two of java statements retrieve parameters from page request. The information that provided by browser is the part of web design. It needs to be visible to web page designer so that the tag is parameterized by attributes. All cases can used this parameterization, where the code to be replace to custom tag and access the page request.

Case 4

HTML content inside a loop

Figure 12 shows a JSP page in which a block of Java code is followed by a table and the table body content is controlled by a loop. A loop can be a while-loop statement or a for loop statement. In the figure, the outermost layer of the table is a HTML table tag. It has an opening tag <table border> and a closing tag </table>. A block of Java code containing a while-loop is contained inside the table tag. The code generates the table body. Moreover the body of the loop (the table body) is made up of HTML text, scriplets and expressions. In the grammar, the interesting elements cover not only JSP elements such as JSP scripting elements, but also interesting HTML elements such as table tags, form tags and anchor tags. As a result, our parser will detect two interesting elements at the top-most scope level for this case. The first element is a scriptlet which contains a block of Java code for the code initialization (i.e. where rs1 is assigned an initial value). The second element is an HTML Table Tag which contains a mixture of HTML and Java code to generate the body of the table. We create two custom tags with a parent-child relationship. The Java code in the top-most scope is migrated into the parent tag mylib:information> , and the other Java code in the sub-scope for the table content generation is migrated into the child tag <mylib:searchResultDisplay>. The child tag, in turn, has children representing the scriptlets that are nested within the body of the loop. Figure 13 shows the transformed page after Java code is migrated into custom tags. The child tag in this case is an iteration action, which means this tag will evaluate its body content (the table cells and table rows) repeatedly until some condition becomes true (i.e. the collection rs1 is empty). This involves implementing the IterationTag interface by the tag class. The code within the loop body makes a reference to the result set that is in the top level tag. This can be simplified by adding a local variable to the child tag for the loop that is initialized as the start of the loop. Then the tags created for the inner children can then easily reference the value from the tag that implements the loop.

```
<html><body>
</body></html>
<%
//more Java code would be here
String userId =request.getParameter("userId").trim();
String password =request.getParameter("Password").trim();
if (! entry.checkLogin (userId, password))
{
%>
<%
}
else
{
s.putValue ("userId", userId);
response.sendRedirect ("postWebLog.jsp");
}
%>
<H1 > Invalid Login</H1 >
<form action="signIn.jsp" method="POST" >
<input type="submit" value="Try Again" >
</form>

<%
}
else
{
s.putValue ("userId", userId);
response.sendRedirect ("postWebLog.jsp");
}
%></body></html>
```

Fig-9: Page With a Choice Statement

```
<html><body>
//more Java code would be here
String userId = request.getParameter ("userId").trim ();
String password = request.getParameter ("Password").trim ();
boolean choice = ! entry.checkLogin (userId, password);
if (choice)
{
%>
<%
}
if (!choice)
{
s.putValue ("userId", userId);
response.sendRedirect ("postWebLog.jsp");
}
%>
<H1 > Invalid Login</H1 >
<form action="signIn.jsp" method="POST" >
<input type="submit" value="Try Again" >
</form>
<%
}
if (!choice)
{
s.putValue ("userId", userId);
response.sendRedirect ("postWebLog.jsp");
}
%>
<html><body>
</body></html>
```

Fig-10:Simplified if Statement

```
<html><body>
<mylib:checkLogin attr1="userId" attr2=”Password”>
<mylib:invalidLogin>
<H1 > Invalid Login</H1 >
<form action="signIn.jsp" method="POST" >
<input type="submit" value="Try Again" >
</form>
</mylib:invalidLogin>
<mylib:validLogin redirectPage=”postWebLog.jsp”>
</mylib:validLogin>
</mylib:checkLogin>
<H1 > Invalid Login</H1 >
<form action="signIn.jsp" method="POST" >
<input type="submit" value="Try Again" >
</form>
```

Fig-11: Simplified If Statement After Transformation

Composing Cases. Obviously not all JSP pages fit into simple choice or loop cases. The cases must compose if we are to handle JSP web pages in general. The example used for case four (the loop case) also llustrated the problem, having simple code nested within the loop, and the loop itself nested within the top level code. Nested code may refer to variables in higher up in the scope hierarchy. Just as the scopes are

```
<html><body>
<%
//more Java code here
String i = request.getParameter ("item");
ResultSet rs1 =CDStoreDB.searchByUPC(Integer.parseInt
(i));
%>
<table border>
<%
String title;
String price;
while (rs1.next ())
{
%>
<tr>
<%
title = rs1.getString (1);
%>
<td> Title </td >
<td> <%= title %> </td >
</tr>
<tr>
<%
price = rs1.getString (2);
%>
<td > Price </td >
<td > <%= currency.format(price)%> </td >
</tr >
</table>
</body ></html >
```

Fig-12: A loop generating a table

```
<html><body>
<mylib:information attr1=”item”>
<table border>
<mylib:searchResultDisplay
<tr >
<td > Title </td >
<td > <mylib:CDTitle/> </td >
</tr >
<tr >
<td > Price </td >
<td ><mylib:CDPrice/>
 </td >
</tr >
</mylib:searchResultDisplay>

</table>
</body></html>
</mylib:information>
</body></html>
```

Fig-13: Page after transformation

nested, the resulting tags will also be nested. So if we were to modify the example shown in Figure 10 to include a loop generating a table inside of the form, the tags generated as part of the loop solution would be nested within the invalidLogin tag. Similarly, if a choice existed within a loop, the tags generated as part of the choice would be nested within the loops.

Our transform performs data flow analysis between the scoping levels, identifying the scope to which a referenced variable belongs. Code is generated using the JSP API to find the instance of the class implementing the parent (or ancestor) tag and invokes the appropriate get or set method to access the variable.

6. IMPLEMENTATION

In this section, we briefly describe the implementation of the transformation. Most of the process is implemented using the TXL language, which is a pure functional programming language particularly designed to support rule-based source-to-source transformation [4]. Figure 14 illustrates our transformation process, which includes five phases. The preprocessing phase normalizes the source code as described in section 3. It also performs some comment and lexical preprocessing. The grouping phase performs an analysis and annotates each line of normalized source code with a tag id identifying the custom tag to which the source code will belong.

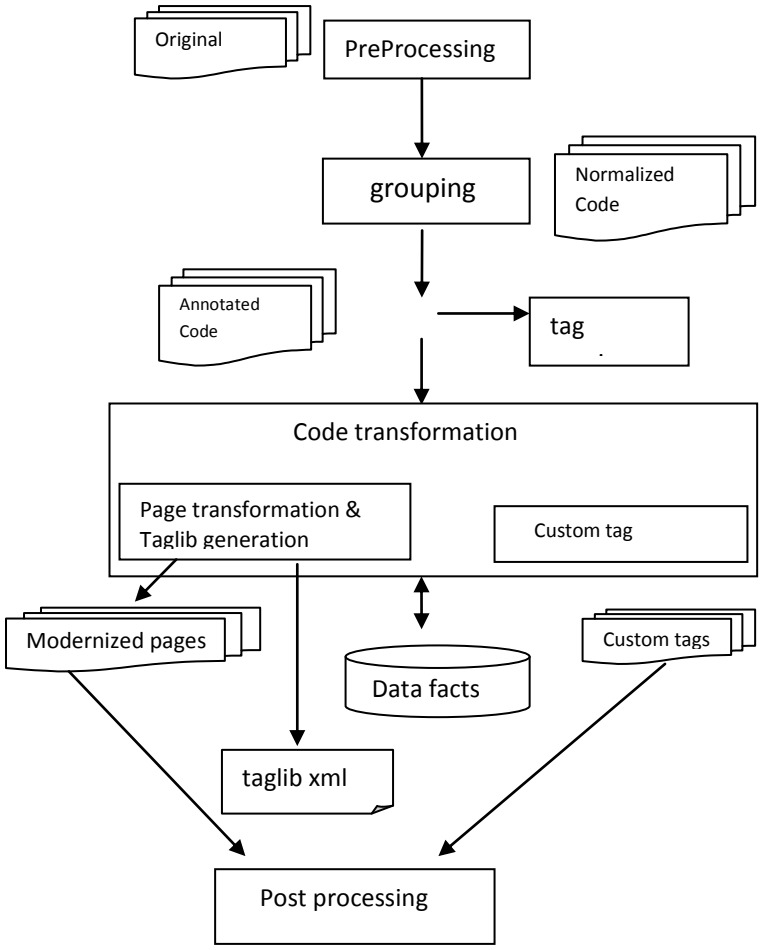

Fig-14: Transformation Process

The first part of the analysis identifies the cases that we have identified in this paper. It identifies control statements that contain HTML/JavaScript and the first statement of Java sequences within the template text that must be given their own tags. Thus, the basic structure of the tags is identified. The second phase of grouping assigns the remaining statements to one of the identified tags. The tag id generated in the group phase is mapped to a reasonable user name such as invalid Login or CD Title by a web interface in the tag naming phase. The code transformation phase uses the markup from the group phase and the mapping from the tag naming phase to generate the three outputs of the process. These are the modernized JSP pages, the tag library description file, and the custom tag classes. The final post-processing phase deals with final touchups such as fixing ommon The whole process is automated, except the tag naming phase where the human assistance is required. The details of the implementation, particularly, the markup approach and the transformations are described elsewhere .We have tested our system on 3 small systems to date consisting of an online music store, a mini weblog application and a guest book application. Two were obtained from within Queen's, the other is a sample system downloaded from the internet. The systems comprise a total of 14 JSP pages containing a total of 682 lines of mixed JSP and HTML. The resulting pages contain 362 lines of tags and HTML. 74 custom tag classes were generated.

Currently each JSP expression is translated into its own custom tag. A simple optimization is to fold the simple JSP expressions into one simple tag. This would liminate 23 custom tag classes.

7. RELATED WORK

There has been considerable investigation into the evolution of web sites. This includes results in program understanding and architectural modeling [6,7,11], clone detection and removal [3,12], restructuring and refactoring and migration [2,8,9,10,14,].Ricca et al. present an approach of using rewrite rules to improve the quality of web applications. The HTML transformations cover both interpage and intrapage transformations and can identify six cases. Further work illustrates a semi-automatic process to identify static pages that can be transformed into dynamic pagesusing clustering techniques. Jiang et al. [9] present a method to migrate a web application to a web service by examining the generated pages and using pattern matching techniques to infer the services.

Hassan et al. [8] propose a framework for migrating web applications between different development frameworks based on water transformations, an extension of island grammars. Ping et al. [14] present an approach for migrating web applications from IBM Net. Data into JSP, separating database access functionality from presentation logic. Lau et al. [10] present a migration methodology, The Modified Table Generation Code supported by a tool developed for the IBM WebSphere Commerce Suite and released on IBM's alpha Works. An alternate approach by Ping et al. restructures JSP based web applications by adapting them to a controller centric architecture. Tilley et al. renovate web applications by reengineering transactions with a user-centered approach.

8. CONCLUSIONS

The implementation of our transformation is a greedy approach. It attempts to group as many statements as possible into each tag. Each web page is also processed independently. One potentially extension is to identify clones between pages, separating them in to separate tags.One example is session management code ommon to multiple pages. In this paper, we have presented a set of transforms that can be used to implement the separation of the presentation and business logic for existing JSP-based webapplications. The transforms restructure the web applications by moving Java code embedded in JSP pages intocustom tags without hanging the original functionalities and user interfaces of the applications. The interesting information required for this restructuring is contained not only in the multiple languages themselves but also in the way they are coupled.

An advantage of our Java code transformation is that all business logic intensive Java code in JSP pages is moved and encapsulated into custom tags and all elements for presentation are kept in pages, which helps to reduce the complexity of web applications and helps make the restructured applications more reusable and maintainable.

9. REFERENCES

[1] Hans Bergsten, JavaServer Pages, O'Reilly 2002.

[2] T. Bodhuin, E. Guardabascio, M.Tortorella, "Migrating COBOL Systems to the WEB by Using the MVC Design Pattern", Proc Working conference on Reverse Engineering, Richmond, Virginia, pp 329-338.

[3] C. Boldyreff, R. Kewish, "Reverse Engineering to Achieve Maintainable WWW Sites", Working Conference on Reverse Engineering, Stuttgart, Germany, Oct. 2001, pp. 249- 257

[4] J. Cordy, "TXL – A Language for Programming Language Tools and Applications", Proc ACM International Workshop on Language Descriptions, Tools and Applications, Edinburg, Scotland, January 2005, pp. 3-31.

[5] A. van Deursen, T. Kuipers, "Building Documentation Generators", Proc International Conference on Software Maintenance, Oxford, England, 1999, pp 40-49.

[6] D. Draheim, E. Fehr, G. Weber, "JSPick – A Server Pages Design Recovery Tool", Proc 7th European Conference on Software Maintenance and Reengineering, Benevento, Italy, March 2003, pp 230-238.

[7] A. Hassan, R. Holt, "Architecture Recovery of Web Applictions", Proc International Conference on Software Engineering, Orlando, Florida, May 2002, p 19-25.

[8] A. Hassan, R. Holt, "Migrating Web Frameworks Using Water Transformations", Proc International Computer Software and Application Conference, Dallas, Nov. 2003 pp 296-303

[9] J. Jian, E. Stroulia, "Towards Reengineering Web Sites to Web Service Providers", Proc European Conference on Software Maintenance and Reengineering, Tampere, Finland, March 2004.

[10] T. Lau, J. Lu, J. Mylopoulos, K. Kontogiannis, "Migrating E-commerce Database Applications to an Enterprise Java Environment", Information Systems Frontiers, Vol 5, N 2, 2003, pp. 149-160.

[11] G. Di Lucca, A. Fasolino, P. Tramontana, C. Visaggio, "Towards the Definition of a Maintainability Model for Web Applications", Proc European Conference on Software Maintenance and Reengineering, Tampere, Finland, March 2004, pp 279-287

[12] F. Lanubile, T. Mallardo, "Finding Function Clones in Web Applications", Proc European Conference on Software Maintenance and Reengineering, Benevento, Italy, March 2003, pp 379-388.