An Automatic Approach to Reengineering Common Website with AJAX

Qingling Wang[1], Qin Liu[2], Na Li[3], Yan Liu[1]
School of Software Engineering, Tongji University
Shanghai 200092, China
{wqingling, giveme areason}@gmail.com[1], qin.liu@mail.tongji.edu.cn[2], lina_sh@yahoo.cn[3]

Abstract

Web applications are suffering from poor interactivity and responsiveness towards end users since the conventional browser-based Web applications follow a request-wait response-wait pattern. This pattern leads to high network latency, interface complexity and slow server responsiveness which impair the user experience, decrease customer satisfaction, cut down the number of visits, and, ultimately, reduce revenue to e-businesses. AJAX (Asynchronous JavaScript and XML) is a standards-based programming technique designed to make Web-based applications more responsive, interactive, and customizable. It is becoming increasingly important to reengineer the traditional web project with AJAX. In this paper, we present a common process to reengineer the traditional web project with AJAX and develop a tool to do the work swiftly, automatically and efficiently. Compared to reengineer a traditional web site with AJAX of medium size, our method will save about 90% workload. This is not only useful for the project manager but also beneficial to the customers.

Keywords

AJAX, Automatically Reengineering

1. Introduction

Web applications are suffering from poor interactivity and responsiveness towards end users since the conventional browser-based Web applications follow a request-wait response-wait pattern which requires the user to submit a request to the server, wait for the server to process the request and generate a response, and then wait for the browser to update the interface with the results. This pattern leads to high network latency, interface complexity and slow server responsiveness which impair the user experience, decrease customer satisfaction, cut down the number of visits, and, ultimately, reduce revenue to e-businesses.

The creation of the programming language, AJAX (Asynchronous JavaScript and XML) [1], is helping web applications advance greatly from this plight. AJAX is a combination of XML and JavaScript to help improve the performance and attractiveness of websites by updating only part of the page at a time. AJAX can increase the levels of interactivity, reduce responsiveness time and improve user experience. Well known examples of AJAX web applications include Google Suggest, Google Maps, Meebo and Gmail. AJAX web applications have attracted a strong interest in the web development community.

Considering these benefits of AJAX, many organizations are beginning to consider migration to this new paradigm. Therefore, in front of enormous amounts of existing web applications, how to reengineer these web applications to AJAX applications swiftly, efficiently and automatically or semi-automatically is becoming increasingly important.

In this paper, we present a common process to convert a traditional web project to AJAX project, in a swift way. We give an automatic convert tool (we call it ACT) to do this work. It is very helpful for the project managers because they could get a new AJAX project in only one day. It's possible that the new AJAX project gets some problems. But it can still make a big improvement of the user experience.

The rest of this paper is organized as follows. We briefly introduce AJAX in section 2. Our swift converting process is shown in section 3. And the detailed steps are presented in Section 4. In section 5 we estimate the workload of the whole process for a sample project. Section 6 is about conclusion and future work.

978-0-7695-3455-8/08 $25.00 © 2008 IEEE
DOI 10.1109/NWeSP.2008.34

2. Problem Definitions

AJAX is an abbreviation for Asynchronous JavaScript and XML used to describe a framework incorporating several technologies and aimed to update web pages without doing a browser refresh [1]. Web applications modeling by using AJAX offer better resource usage than traditional architectures. [2] gives Some data that is relative to the performance improvement of the Ajax architecture over the traditional client/server architecture. We can see that the Ajax architecture outperforms the client/server architecture in some respects, the data transfer volume, data transfer time and overall transmission time. Taking the AJAX architecture on e-commerce could save a considerable amount of money for the company because compared to the client/server architecture the server spends less time on user requests and processes fewer bytes of information.

Attracted by the huge profit, more and more organizations are eager to substitute AJAX architecture for the traditional one. Reengineering the traditional web system to AJAX rashly will not only take up a lot of time and energy but also get unexpected results. The ACT (the automatic convert tool) can solve this dilemma. It can reengineer a traditional web system with AJAX automatically, efficiently and swiftly. You can see the converting results in even only one day. Therefore there is no need to worry about time and money wasting. It is ok to keep the original web system if you are not satisfied with the new one. The great contribution of the ACT presented in this paper is its swift and automatic convert process.

3. The Swift Converting Process

Our goal is giving a common process to reengineer an existing traditional web project with AJAX swiftly. In this section we will show the general steps.

Figure 1 depicts an overview of the process which consists of the following five steps.

- Extract Template
- Generate Skeleton
- Rewrite Index Page
- Rewrite Front Controller
- Convert JSP to JAVA code automatically
- Merge the result above

Next we will discuss each step in detail. The main focus of this paper is on step five, converting JSP to JAVA code which is done by the introduced tool ACT automatically.

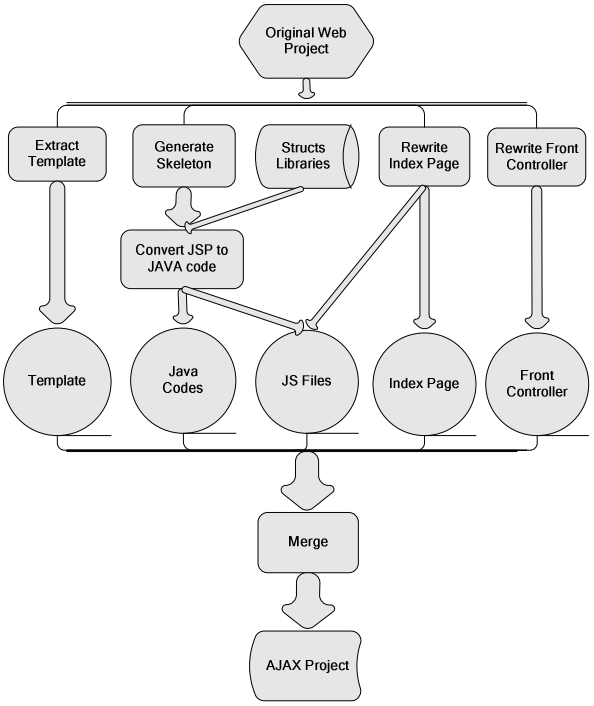

Figure 1. Common Process

4. The Details of Workflow

4.1 Extract Template

Generally, there is a share part in most pages of a web site (header, menu, footer and so on), which does not need to be transferred between server / client. We can extract this part and make it into a template (in fact, lots of IDEs and frameworks support this function). Part A and B, in figure 2 is what we should extract out.

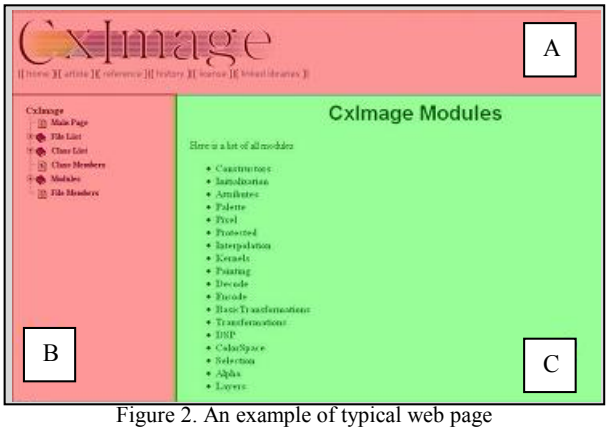

Figure 2. An example of typical web page

4.2 Generate Skeleton

Generating skeleton means generating java classes from corresponding JSP files automatically. We will use these java classes to supplant the old JSP files to respond to client requests. Figure 3 shows the original JSP Files of the traditional web site. We use the tool ACT to generate skeleton and figure 4 shows the generated java classes.

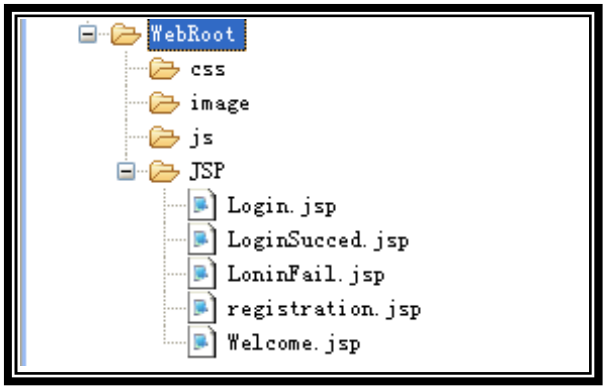

Figure 3. Original JSP Files

Figure 4. Result java classes of Generating Skeleton

4.3 Rewrite Index Page

We have extracted the template in 4.1, now we use it to generate the index page which will not be updated for every request. We only need to update part C in figure 2. To do that, we use some utility JavaScripts to control the update process.

When user clicks some link, we use the utility JavaScript to send XMLRequest to the server and get the response. Then we use the response string to replace the main content part (part C in figure 2), for example, in callback function, we can write, contentDiv.innerHTML = responseString; . If the server response is not the correct content but some error, which means something bad happened, and then we should use the traditional way to refresh this page. That's a backup solution to ensure the web site always works.

Some other utility files, i.e. JavaScripts, CSS, or more convenient UIs which customer wants, could also be selectively added to our index page.

4.4 Rewrite Front Controller

The traditional web sites use HTTPRequest and HTTPResponse to refresh the whole page. In order to get a new AJAX project from the traditional one, we must write a new front controller to in place of the old one.

We introduce an AJAX framework, Direct Web Remoting (DWR) [3], which we will use to process the XMLRequest. DWR is an excellent framework used widely in AJAX projects, to control the XMLRequest between server and client.

AJAX request URLs are different from the traditional ones so we should make a mapping between them. Then the new front controller can find out which action the AJAX request wants. The front controller will catch all the AJAX URL requests. After receiving them, the front controller will do following.

1. Find out the action class by reading the old structs configuration file.
2. Extract the form bean and pass it to the function "execute()", and meanwhile invoke this function.
3. Get the return value of the 'forward' type.
4. Based on the return value in step 3, find the corresponding JSP file, and then find the corresponding java class in the skeleton generated in 4.2, for instance, LoginSucceed (forward) -> /JSP/LoginSucceed.jsp -> com.tongji.example.jsps.LoginSucceed.java.
5. Call the getContentString() function of the target class (result class of step 4) to get the response string and send it to the client.

If some error happened during above processes, we will return the error to the client.
Limit to the space, some other detailed works, which are very easy, have not been discussed in this step.

4.5 Convert JSP to JAVA code

This step is the most important and difficult part. Figure 5 shows the workflow of how to convert the JSP file to JAVA codes. The steps described below are all processed by the ACT tool automatically.

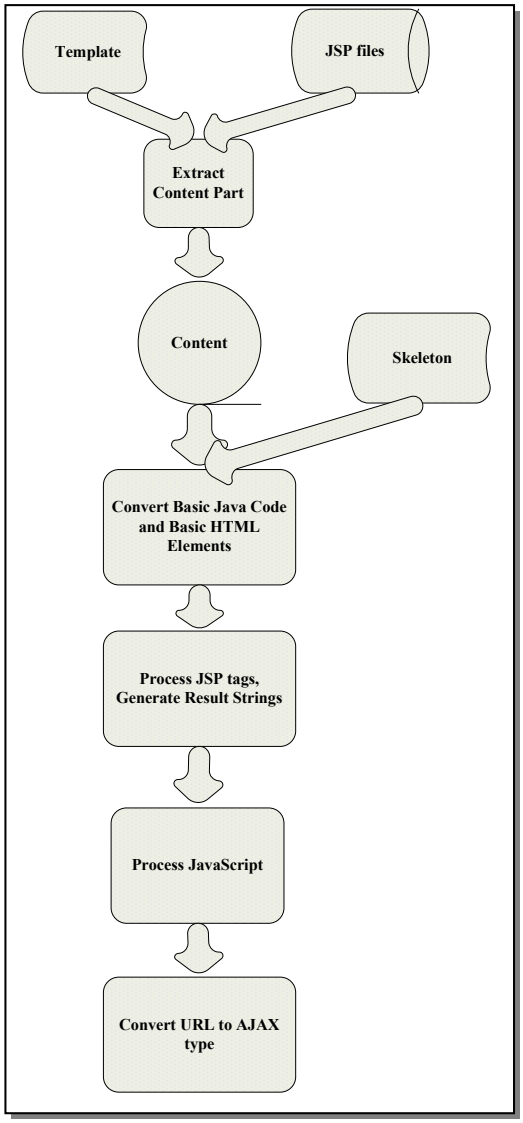

Figure 5. Workflow for Converting JSP to JAVA code

4.5.1 Extract Content Part

For each JSP file, we extract the content part (part C in figure 2) but ignore the shared part (part A and Part B in figure 2).

4.5.2 Convert basic Java Code and Basic HTML Elements

Figure 6 shows the original JSP code from the traditional web project and figure 7 shows the result of converting basic HTML elements to java code.

We can see that, the converting result is a function, whose result string is simple HTML code. The client can use it to substitute the old JSP code to show the same information to customer. That's just what we want.

```
<%
User user = (User)request.getSession().getAttribute
("CurrentUser");
%>
<div>
        <H2>Welcome <%=user.getName()%>!</H2>
</div>
```

Figure 6. Original JSP code

```
String getContentString(HttpRequest request, HttpResponse
response){
        OutputStream out = new OutputStream();
        User user =
        (User)request.getSession().getAttribute("CurrentUs
        er");
        out.write("<div>");
        out.write("<H2>Welcome ");
        out.write(user.getName().toString());
        out.write("!</H2>"+
        "</div>");

        return out.getString();
}
```

Figure 7. Converting Result (Java code)

4.5.3 Process JSP tags

Until now we have converted most JSP code. There are still other customer JSP tags, now what we should do is to process these JSP tags. First find the classes corresponding to these JSP tags in the config file and then invoke the functions of the classes, just like what the java web server did. If this way doesn't work, we can parse the tags by ourselves, however it means only the structs tags will be processed.

4.5.4 Process JavaScript

There may be some JavaScript codes embedded in the HTML, which are not included in the response string and will not run in the client web browser. To achieve this, we extract these JavaScript codes and send them to client in another function. It's very similar with the HTML part process.

4.5.5 Convert URL to AJAX type

Now the server can return the content string to only refresh the content part. The request sent to server is

still the traditional type, so we should convert it to AJAX type. We should change the 'src' property of all link elements. If the link is clicked, we call a JS function and pass the old URL to it. This function will first determine whether the URL is an internal link; if it is, we convert it to an XMLRequest and send it to the server; if not, we process it in the traditional way.
We can also change the form element to AJAX type request but there may be some difficulties. If we cannot do this, we can follow the traditional way. Fortunately, there are not many form requests in a web site, so this is not a big problem in our solution.

4.6 Merge the result

We have finished all the detailed works. Now we merge all of them together into a whole solution and do some adjustment.

5. Effort Estimation

Assume we have a traditional web project in hand. We use the ACT tool to reengineer it with AJAX. Since all the processes in this paper are done automatically by the tool, thus we can get the new AJAX front controller, index page, utility JavaScript files and java codes by the converting process of ACT. What we should do manually is some tiny adjustment to make the new project run. Based on this, compared to reengineer a traditional web site with AJAX of medium size, our method will save about 90% workload. Maybe we can get the new web site in only one day.

We choose "Shenzhen Urban Transportation Information System (UTIS)", which is a publicly available dynamic web application, as our migration case study. Table 1 shows the overview of this project.

	count
JSP pages	96
Java classes	261
code lines	15,169

Table 1 Shenzhen Urban Transportation Information System overview

If conducting a manually engineering to ajaxify this web application, about 70% of the source codes need to be porting to AJAX type. The average build efforts of our develop group is around 500 LOC/ PM, so the workload of ajaxifying STSP is about 12 person-months. ((15169*70%)/500 ≈ 12 person-months). Table 2 shows the concret data about the effort cost of automatic porting and manually porting. (the 0.25 mouth of automatic porting includes UI adjustment time)

In the respects of ajaxifying rate, customer experience improvement and performance improvement, our automatic porting method is not that good as manually porting method, about 5%-10% lower. In the respect of effort cost, our method is much better which best balances the deficiencies in other respects. As a whole, our automatic porting brings more benefits for the organizations or companies.

	Manually porting	our automatic porting
Effort cost of porting	12 person-months	0.25 person-months
Ajaxifying rate	about 80%	70%-75%
Customer experience improvement	80%	70%
Performance improvement	58%	46%

Table 2 Porting overview

6. Conclusion and future work

In this paper, we propose a process to reengineer a traditional web project with AJAX. The most important contribution is to make the converting by an automatic tool ACT very swiftly. It is especially useful for the project manager. By using this tool, compared to reengineer a traditional web site with AJAX of medium size, our method will save about 90% workload. There may be a few pages still in the traditional way, but the user experience can be improved a lot. If the managers are not satisfied with the converted result, what they lost is only one working day.

Now the ACT tool can only do some basic jobs, such as converting basic HTML elements and java code, making simple index page and simple front controller. In the future, we will continue working on this tool to make it process more complex JSP files (i.e. some JSP files including customer JSP tags, some other complex JavaScripts).

Though we don't need to refresh the whole page, we still need refresh the whole content part. In the future, we will only refresh the different area in the content part between the two fore-and-aft pages.

References

[1] Michele Angelaccio and Berta Buttarazzi. A Performance Evaluation of Asynchronous WebInterfaces for Collaborative Web Services, WOMP 2006 Workshop, 2006, Italy.

[2] Keith Smith. Simplifying Ajax-Style Web Development, 2006 IEEE. Reprinted from Computer Magazine, Volume 39, Issue 5, May 2006, pp. 98-101.
[3] Direct Web Remoting: www.dwr.com.
[4] R. Fielding. Architectural styles and the design of network-based software architecture. PhD thesis, UC, Irvine, Information and Computer Science, 2000.
[5] A. Mesbah and A. van Deursen. An architectural style for Ajax. In WICSA' 07: 6th Working IEEE/IFIP Conference on Software Architecture. IEEE Computer Society, 2007.
[6] A. Mesbah and A. van Deursen. Migrating multi-page web applications to single-page Ajax interfaces. In CSMR' 07: 11th European Conference on Software Maintenance and Reengineering. IEEE Computer Society, 2007.
[7] F. Ricca. Analysis, Testing and Re-structuring of Web Applications. PhD thesis, University of Genova (DISI), 2003.
[8] G. A. Di Lucca, M. Di Penta, and A. R. Fasolino. An approach to identify duplicated web pages. In COMPSAC' 02: 26th International Computer Software and Applications Conference, pages 481-486. IEEE Computer Society, 2002.
[9] J. Garrett. Ajax: A new approach to web applications. Adaptive path, 2005. http://www.adaptivepath.com/publications/essays/archives/000385.php.
[10] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora. Understanding cloned patterns in web applications. In IWPC' 05: 13th International Workshop on Program Comprehension, page 333-336. IEEE Computer Society, 2005.