Manuscript received August 17, 2007; revised November 15, 2007

Uniform Tag-based Rich Component Generation for Web Application Development

Quan Liang Chen
University of Tokushima
Course of Info Sci. & Intel. Syst.
Tokushima
JAPAN
quan@is.tokushima-u.ac.jp

Takao Shimomura
University of Tokushima
Dept. of Info Sci. & Intel. Syst.
Tokushima
JAPAN
simomura@is.tokushima-u.ac.jp

Nhor Sok Lang
University of Tokushima
Course of Info Sci. & Intel. Syst.
Tokushima
JAPAN
soklang@is.tokushima-u.ac.jp

Kenji Ikeda
University of Tokushima
Dept. of Info Sci. & Intel. Syst.
Tokushima
JAPAN
ikeda@is.tokushima-u.ac.jp

Abstract: A variety of rich client technologies such as Flash, Flex, OpenLaszlo, JavaScript, AJAX, and Applet have been employed to develop Web applications. They can display flexible and powerful graphical user interface in Web pages and excel the original functions of Web browsers that display HTML documents. This paper presents a rich-component definition method that enables programmers to easily write Web pages that contain rich components, regardless of the types of rich clients. This separates Web programming from Web page design, and we can share Web application development with other developers.

Key–Words: Flex, javascript, rich component, uniform tag, web applications

1 Introduction

A variety of rich client technologies such as Flash [1], Flex [2], OpenLaszlo [3], JavaScript [4], AJAX [5], and Applet [6] have been employed to develop Web applications. They can display flexible and powerful graphical user interface in Web pages and excel the original functions of Web browsers that display HTML documents. Some of these rich clients need to incorporate their plug-in software into Web browsers. Using the plug-in software, they display particular files of their own formats, or execute particular programs.

To develop Web applications using these rich clients, we need to define rich components displayed in Web pages using not HTML but some other languages, such as JavaScript code and its invocation, or XML-based code of their particular formats. Therefore, the developers of Web applications have to be accustomed with each of these languages. The aim of this paper is to present a novel mechanism that capsules these pieces of code.

Component-based development makes it much more efficient to develop Web applications [7], [8], [9]. If we employ useful components, we can develop Web applications of higher quality, more easily and more efficiently [10], [11]. Packaging techniques that provide easy interface to use components are also important [12].

This paper presents a rich-component definition method that enables programmers to easily write Web pages that contain rich components, regardless of the types of rich clients [13]. This separates Web programming from Web page design, and we can share Web application development with other developers [14]. The paper first proposes uniform rich component tags that can be written in the same way for any rich client and that generate various kinds of rich components. It then describes the method that implements these rich component tags. To make it possible to write tag handlers [15] that process rich component tags as little code as possible for each rich client, it applies the template method [16] to constructing these tag handlers.

The remainder of this paper is organized as follows: Section 2 shows an example of the rich component tags this paper presents, and explains the requirements for implementation of these tags. Section

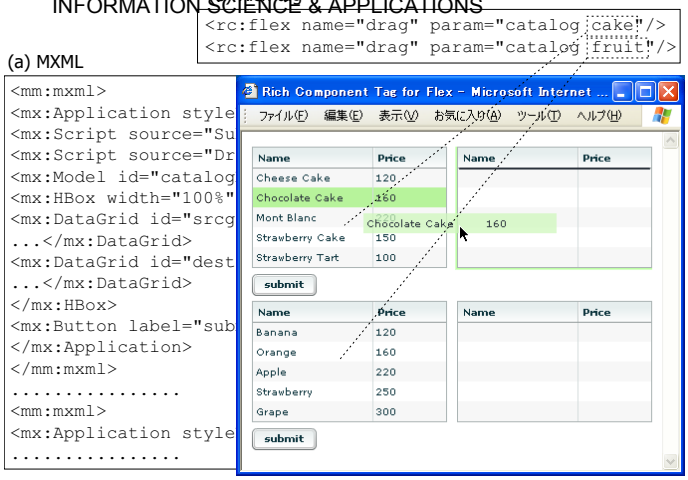

Figure 1: Example of drag-and-drop GUI with Flex

3 presents how to implement these rich component tags in a uniform way. Section 4 illustrates the processes of the tag handlers that create rich components such as Flex, Applet, and JavaScript. Section 5 discusses the effectiveness of this method and its limitation. Finally, Section 6 summarizes the paper.

2 Requirements for Defining Rich Components

This section shows an example of the rich component tags this paper presents, and explains the requirements for implementation of these tags.

2.1 Example of rich component definition

As an example of rich clients, Fig. 1 shows a Web page that is displayed using Flex. The upper row in the Web page shows a list of cake, and the lower row shows a list of fruit. In both lists, we can select and drag an item displayed in the left column and drop it in the right column. By drag-and-drop operations, we can change the order of items, return items from right to left, and select multiple items at a time and move them. In ordinary HTML functions, it is difficult to use such flexible and powerful graphical user interface on Web pages.

In Flex, we write such a Web page using a MXML language. As shown in Fig. 1(a), we need to write a lot of code, and we also have to get accustomed with this MXML language. The method this paper proposes enables programmers to easily write rich components using a <rc:flex> tag (See Fig. 1(b)).

2.2 Requirements

To make it possible to define rich components, regardless of the types of rich clients, we take into account the following requirements:

1. We identify rich clients with their rich component's tag name, and for any type of rich client, we can specify the attribute values of the rich component tag in a uniform way.

2. We do not need to write a tag handler that generates a rich component from scratch. For any type of rich component, we can write its tag handler in the same interface.

3. If a new rich client becomes available, we can easily add a tag that defines its rich components.

3 Rich Component Generation

This section presents how to implement these rich component tags in a uniform way.

3.1 Uniform tag handlers

We here simply explain uniform tags and how their tag handlers work. As shown in Fig. 2, when a Web browser sends a request to a JSP page on the Web server, the Servlet engine on the server preprocesses the JSP page, generates a Servlet program from it, and invokes the Servlet program. The Servlet receives the request, processes it, and then returns the response to the Web browser.

In this JSP page, we can write any tag programmers provide in addition to standard HTML tags such as <form>, <img>, and <a>. For example, when a Web browser sends a request to a JSP page in which <tag> is written, the Servlet engine preprocesses the JSP page, generates a Servlet, and then invokes the Servlet. This Servlet invokes a tag handler that processes this <tag>. The tag handler analyzes the values of attributes of <tag>, and then returns an appropriate response to the Web browser.

This paper proposes the tags that define rich components, and explains how to implement tag handlers that process these rich component tags. Figure 3 shows the outline of this process. Let's consider a case in which a rich component tag <rc> is written in a JSP page. When this JSP page receives a request, the tag handler that corresponds to the rich component tag is invoked. This tag hander generates a JSP file in which some tags for the corresponding rich client are written. The tag hander then transfers the request to this dynamically generated JSP file, and includes the response this JSP page returns.

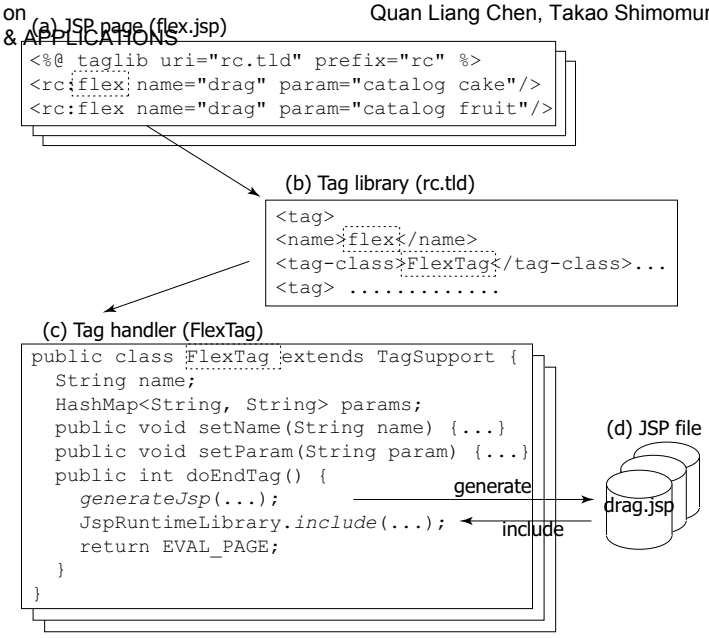

Figure 4: Generation of rich components using tags

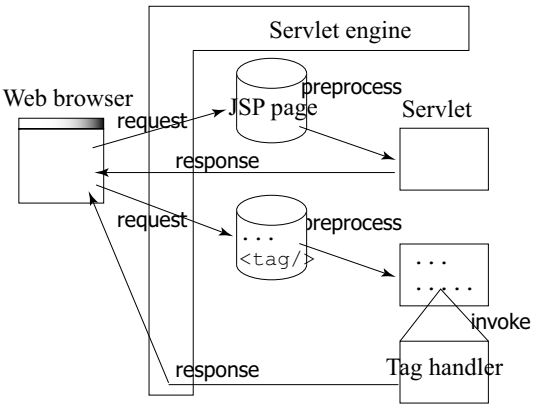

Figure 2: Invocation of tag handlers

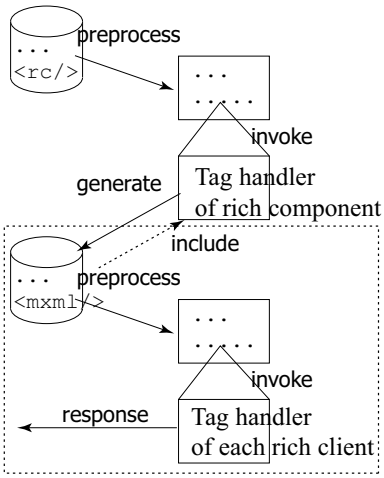

Figure 3: Generation and inclusion of JSP pages

When the dynamically generated JSP file is included, this JSP file is preprocessed by the Servlet engine, and the tags in this JSP file the rich client provides are processed by the tag handlers of the rich client. For example, if the generated JSP file contains Flex's tags written in MXML language, the tag handler of Flex will generate a Flash movie to return to the Web browser as its response.

3.2 Rich Component Generation Processes

There are some rich clients in which, to define their rich components, we have only to write a combination of HTML and JavaScript code, and others in which, we need to write XML-based code of their particular formats such as MXML [2], Code generation tags [17], and LZX [3].

If we need to write only HTML code to define rich components to be displayed in a Web page, the tag handler that processes the rich component tag has only to transform the tag into HTML code as in PageGen [18]. However, in the case of JSP pages in which particular XML-based code is written as in Flex, this simple method does not work because the JSP page must be preprocessed by the tag handler the corresponding rich client provides.

In the method this paper proposes, program-

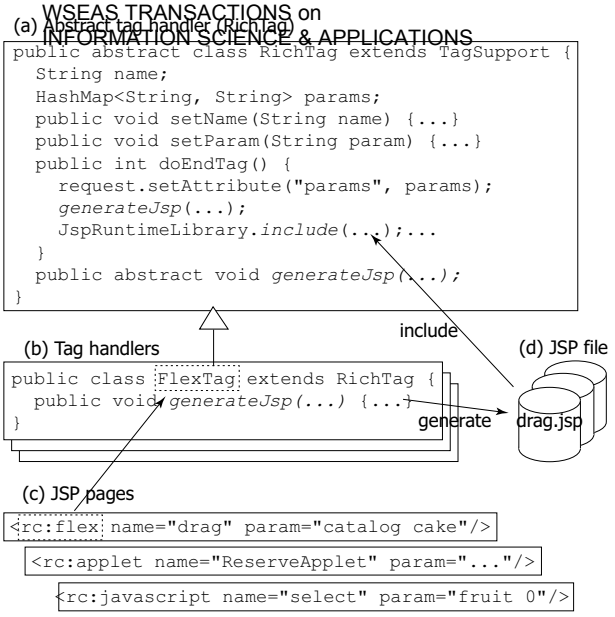

Figure 5: Implementation of tag handlers for rich components

mers specify the type of a rich client using the tag name of a <rc:richComponent> tag, and all types of rich clients can commonly use the values of tag attributes (Requirement 1). The tag handler of the <rc:richComponent> tag generates a JSP file that is written in a particular XML-based language of the corresponding rich client. The tag handler then includes this generated JSP file to let the rich client's tag handler preprocess it. In the case of generating a combination of HTML and JavaScript code, the tag handler skips the processes of generating and including a JSP file. If the JSP file written in a particular XML-based language is already prepared, the tag handler skips the process of generating a JSP file and only performs the process of including a JSP file.

Let's consider a case in which we use a <rc:flex> tag in a JSP page to create a Flex rich component. As shown in Fig. 4 (a), the name of the tag handler of a <rc:flex> tag is specified in a tag library descriptor file rc.tld. This file rc.tld specifies that the name of the tag handler class for the tag whose name is flex is FlexTag. Therefore, to process the <rc:flex> tag, FlexTag tag handler is invoked. As shown in Fig. 4 (c), Tag handler FlexTag first reads the values of attributes name and param using setName() and setParam() methods, and then using doEndTag() method, generates a JSP file, and includes the generated JSP file (discussed in Section 4.1 in detail).

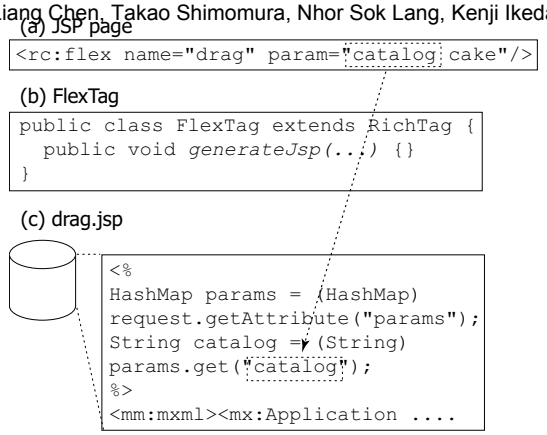

Figure 6: Generation of rich components for Flex

3.3 Introduction of abstract tag handler

The tag handlers that generate rich components need to perform a series of processes such as reading the values of attributes name and param, generating a JSP file, and including the generated JSP file. The process of generating a JSP file is dependent on the type of rich component. On the other hand, the other processes are common among all tag handlers of rich components. Therefore, instead of writing a tag handler for each type of rich clients separately, we introduce an abstract tag handler so that we can write tag handlers in the same interface and as little code as possible (Requirement 2).

As shown in Fig. 5, the abstract RichTag class reads the values of attributes name and param. The value of attribute param consists of a sequence of a pair of name and value. Class RichTag stores those values in variable params of type HashMap with name as a key. The doEndTag() method invokes an abstract method generateJsp() to let the tag handler of each rich component generate its own JSP file. In addition, it stores the value of variable params in the request object so that the generated JSP file can also refer to the values of attribute param of the rich component tag when it is included.

The tag handler of each rich component has only to implement generateJsp() method because it inherits the abstract RichTag class. This makes it easy to add a rich component for a new rich client when it becomes available (Requirement 3). For example, for other rich components such as Applet and JavaScript, we have only to define the classes of their tag handlers and write their generateJsp() methods. It is because the syntax of these rich component tags is the same although the semantics is different.

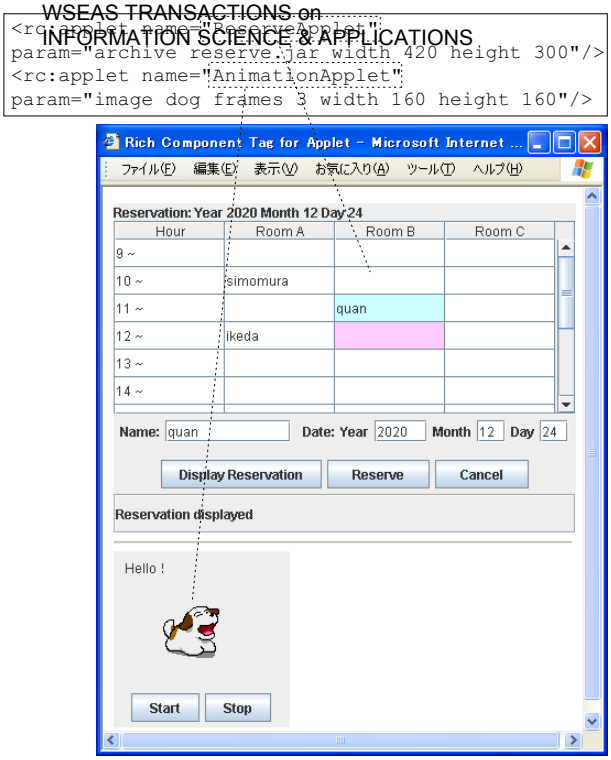

Figure 7: Example of meeting-room reservation and animation with JApplet

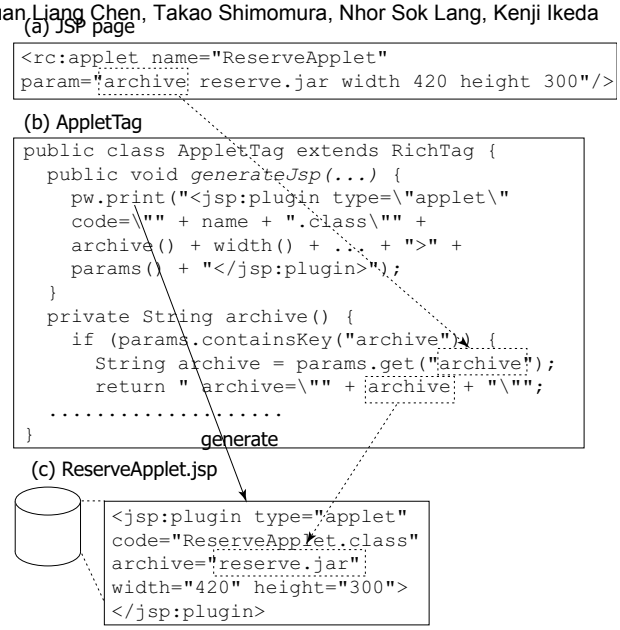

Figure 8: Generation of rich components for JApplet

4 Rich Component Tag Handlers

This section illustrates the processes of the tag handlers that create rich components such as Flex, Applet, and JavaScript.

4.1 Generation of rich components for Flex

We can generate the Flex rich component shown in Fig. 1 by writing FlexTag class that inherits abstract RichTag class. Using generateJsp() method, we may dynamically generate JSP page drag.jsp written in MXML. However, to make it easier to maintain drag.jsp, it is better to prepare this JSP page separately as a file.

Abstract class RichTag stores the value of attribute param of the <rc:flex> tag in the request object in advance. Therefore, as shown in Fig. 6(c), JSP page drag.jsp can receive this value as a HashMap object. This makes JSP page drag.jsp flexible, and makes it possible for multiple <rc:flex name="drag"/> tags to use the same JSP page drag.jsp to include.

4.2 Generation of rich components for Applet

Figure 7 shows an example of definition of Applet rich components. The upper row of the Web page shows a Web-based reservation system for meeting rooms. When we specify a date (year, month, day) and click on the "Display Reservation" button, the reservation status for that day is shown. The top row shows the names of meeting rooms, room A, room B, and room C. The left column shows several periods of time. For example, room B is reserved by Quan from 11:00. To make a reservation, we choose a vacant cell and then click on the "Reserve" button. To cancel a reservation, we select a reserved cell and then click on the "Reserve" button. The "Cancel" button cancels the current selections. The lower row shows the animation of a dog.

Although we can easily define applets using <applet> tags, we can also define them using <rc:applet> tags in the same way as we define other types of rich components. We can specify various kinds of values in an <applet> tag all together using attribute param of the <rc:applet> tag.

As shown in Fig. 8, the <rc:applet> tag dynamically generates a JSP file (ReserveApplet.jsp) whose name is the same as the applet, and then includes this JSP file.

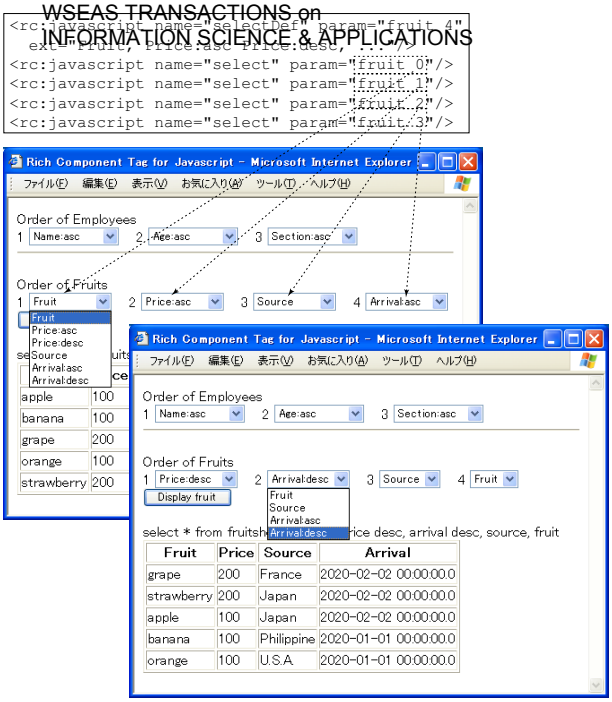

Figure 9: Example of hierarchical pull-down menus with JavaScript

4.3 Generation of rich components for JavaScript

Figure 9 shows an example of definition of JavaScript rich components that display hierarchical pull-down menus. The upper row shows three hierarchical pull-down menus for employees, and the lower row shows four hierarchical pull-down menus for fruit. Below the pull-down menus for fruit, there is a table that has four fields, fruit, price, source, and arrival. These pull-down menus are related to each other.

In the hierarchical pull-down menus for fruit, for example, four menus are displayed in a line, and they specify the order in which fruits are arranged in the table below. The menu consists of six menu items such as “Fruit”, “Price:asc”, “Price:desc”, “Source”, “Arrival:asc”, and “Arrival:desc”. In the first pull-down menu, if we choose “Price:desc” that arranges the fruits in descending order of price, the second pull-down menu will show only four of six menu items that consists of “Fruit”, “Source”, “Arrival:asc”, and “Arrival:desc”, because the order of price has already been specified. If we then specify “Arrival:desc” in the second pull-down menu, the third pull-down menu will show only two menu items “Fruit” and “Source”.

Using a <rc:javascript name=“selectDef”> tag, we specify the number of pull-down menus. We also specify the name of each menu item, and the group

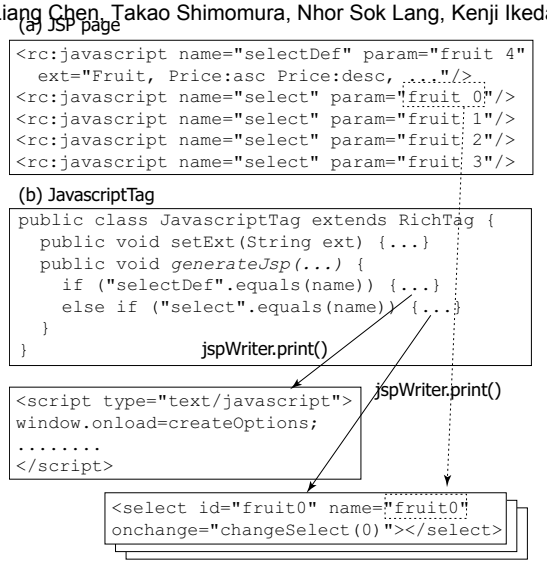

Figure 10: Generation of rich components for JavaScript

it belongs to using attribute ext of the tag as additional information. A <rc:javascript name=“select”> tag defines each pull-down menu displayed in a Web page.

Hierarchical pull-down menus have only to generate a combination of HTML and JavaScript code. Therefore, the generateJsp() method of the tag handler generates HTML and JavaScript code instead of generating a JSP file. In this case, the tag handler does not perform its including process. As shown in Fig. 10, the <rc:javascript name=“selectDef”> tag generates JavaScript code to operate hierarchical pull-down menus, and the <rc:javascript name=“select”> tag generates HTML code to display those hierarchical pull-down menus.

5 Observation

Some rich clients use a combination of HTML and JavaScript code to define their rich components, and other rich clients use particular XML-based languages such as MXML and LZX. In the method this paper has presented, various kinds of rich components can be defined in a uniform interface by using <rc:richComponent> tags. This makes it possible to separate the programming of Web applications from the design of Web pages.

Table 1 shows the comparison between the conventional development with JSP and the rich component method with <rc> tags. The table compares the lines of code that are required by both methods when we create and use rich components provided by rich

Table 1: Lines of code required to define rich components

		Conventional development with JSP		**Rich component method with <rc>**	
		Web programmer	Common modules	Web programmer	Rich component programmer
Flex	dragging items	151 (MXML 48, ActionScript 74, XML 29)	0 —	30 (<rc> 1, XML 29)	129 (JSP 7, MXML 48, ActionScript 74) (FlexTag handler 4)
	text input	37 (MXML 37)	0 —	1 (<rc> 1)	40 (JSP 9, MXML 31)
Applet	reservation	3 (JSP 3)	279 (Java 279)	2 (<rc> 2)	279 (Java 279) (AppletTag handler 73)
	animation	7 (JSP 7)	62 (Java 62)	2 (<rc> 2)	62 (Java 62)
JavaScript	hierarchical pull-down menus	49 (JSP 49)	471 (Java 312, JavaScript 159)	6 (<rc> 2, <rc> 4)	485 (JavaScriptTag handler 326, JavaScript 159)
	HTML view editor	102 (JSP 23, MenuFile 79)	419 (Java 136, JavaScript 283)	82 (<rc> 3, MenuFile 79)	431 (JavaScriptTag handler 148, JavaScript 283)

clients such as Flex, Applet, and JavaScript. The effects of reducing the number of lines of code with rich component tags are remarkable when we use rich components for such rich clients as Flex in which a XML-based language is used to define their rich components. Once rich component programmers prepare Flex rich components, Web programmers can define these rich components using <rc> tags any number of times. For Applet rich components, although the effects of reducing code with rich component tags are small, we think that the effects of enabling programmers to define rich components using <rc> tags in a uniform way are not small. For JavaScript rich components, we can define rich components using <rc> tags concisely. Even in the conventional development, Web programmers might be able to define rich components concisely if the most parts of JSP code are incorporated into the common modules. In this case, however, we think that this method that uses <rc> tags is better.

The <rc> tags have common attributes name and param. The value of attribute param consists of a sequence of a pair of (name, value), with which we specified various kinds of values such as (catalog, cake), (archive, reserve.jar), and (width, 420) as shown in Figures 1 and 7. However, there are some values that are difficult to define using such a pair of values. For example, for the hierarchical pull-down menus composed as JavaScript rich components, when we specified the menu items, their order, and the groups they belong to, we had to introduce another attribute ext (See Fig. 9).

This method has introduced an abstract class RichTag to apply the template method to constructing the tag handlers of rich components. This makes it possible to easily add new rich components to Web pages when the corresponding rich client becomes available. The method enables programmers to easily develop rich component tags for various kinds of rich clients to prepare useful rich components, and makes the process of developing Web applications efficient.

6 Conclusion

This paper has proposed rich component tags that enable programmers to easily write Web pages that are displayed by various kinds of rich clients, and described their implementation. This method makes

it possible to define various kinds of rich components in a uniform interface. The tag handlers that process rich components dynamically generate JSP pages and include them. Therefore, the recursive process of <rc:richComponent> tags is possible in which generated JSP pages are also defined using <rc:richComponent> tags. We are going to investigate various kinds of applications for rich component tags.

References:

[1] *Adobe Systems Inc.* http://www.adobe.com/products/flash/, 2006.

[2] *Adobe Systems Inc.* http://www.adobe.com/products/flex/, 2006.

[3] *Laszlo Systems, Inc.* http://www.openlaszlo.org/, 2006.

[4] Michael Brooks. *Essentials for Design Javascript Comprehensive*. Prentice Hall, 7 2006.

[5] Edmond Woychowsky. *Ajax : Creating Web Pages with Asynchronous Javascript and Xml*. Prentice Hall, 8 2006.

[6] *Sun Microsystems, Inc. : Java 2 Platform Standard Ed. 5.0*. http://java.sun.com/j2se/1.5.0/docs/api/, 2004.

[7] Seung C. Lee and Ashraf I. Shirani. A component based methodology for web application development. *Journal of Systems and Software*, Vol. 71, No. 1-2, pp. 177–187, 4 2004.

[8] Takao Shimomura, Kenji Ikeda, Quan Liang Chen, Nhor Sok Lang, and Takahashi Muneo. Visual programming for web applications that use html frame facilities. In *Proc. of WSEAS International Conference on Computer Engineering and Applications*, pp. 384–389, 1 2007.

[9] Takao Shimomura, Kenji Ikeda, Quan Liang Chen, Nhor Sok Lang, and Muneo Takahashi. Visual programming of hierarchical frames for web applications. *WSEAS Transactions on Information Science & Applications*, Vol. 4, No. 5, pp. 968–975, 2007.

[10] K.M. Khan and J. Han. Composing security-aware software. *IEEE Software*, Vol. 19, No. 1, pp. 34–41, 2002.

[11] A. Repenning, A. Ioannidou, M. Payton, W. Ye, and J. Roschelle. Using components for rapid distributed software development. *IEEE Software*, Vol. 18, No. 2, pp. 38–45, 2001.

[12] J. Hopkins. Component primer. *Communications of the ACM*, Vol. 43, No. 10, pp. 27–30, 4 2000.

[13] Takao Shimomura, Kenji Ikeda, Quan Liang Chen, Nhor Sok Lang, and Muneo Takahashi. Rich component generation for web applications using custom tags. In *Proc. of WSEAS International Conference on Computer Engineering and Applications*, pp. 390–395, 1 2007.

[14] A. Leff and J.T. Rayfield. Web-application development using the model/view/controller design pattern. In *Fifth International Enterprise Distributed Object Computing Conference*, pp. 118–127, 9 2001.

[15] *Sun Microsystems, Inc. : JavaServer Pages Technology*. http://java.sun.com/products/jsp, 2006.

[16] Steven John Metsker and William C. Wake. *Design Patterns in Java*. Addison-Wesley, 4 2006.

[17] *Iron Speed, Inc. : Iron Speed Designer*. http://www.ironspeed.com/, 2006.

[18] Nasir Al-Darwish. Pagegen: an effective scheme for dynamic generation of web pages. *Information and Software Technology*, Vol. 45, No. 10, pp. 651–662, 7 2003.