Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2009, Article ID 879725, 16 pages
doi:10.1155/2009/879725

Research Article

Automatic Generation of Web Applications from Visual High-Level Functional Web Components

Quan Liang Chen and Takao Shimomura

Department of Information Science and Intelligent Systems, University of Tokushima, 2-1 Minamijosanjima, Tokushima 770-8506, Japan

Correspondence should be addressed to Takao Shimomura, simomura@is.tokushima-u.ac.jp

Received 10 June 2008; Revised 29 September 2008; Accepted 25 November 2008

Recommended by Andrea De Lucia

This paper presents high-level functional Web components such as frames, framesets, and pivot tables, which conventional development environments for Web applications have not yet supported. Frameset Web components provide several editing facilities such as adding, deleting, changing, and nesting of framesets to make it easier to develop Web applications that use frame facilities. Pivot table Web components sum up various kinds of data in two dimensions. They reduce the amount of code to be written by developers greatly. The paper also describes the system that implements these high-level functional components as visual Web components. This system assists designers in the development of Web applications based on the page-transition framework that models a Web application as a set of Web page transitions, and by using visual Web components, makes it easier to write processes to be executed when a Web page transfers to another.

1. Introduction

To develop a Web application, we need to write a lot of codes to perform processes such as display of Web pages, receipt of requests, execution of actions, session operations, database accesses, and business logic. On the other hand, before we develop a Web application, we have some images of the application (i.e., what it will be like or what it will look like) in our mind. The objective of our research is to make it possible to develop Web applications with less code by making full use of such image. Conventional development environments for Web applications only provide fundamental Web components such as text fields, buttons, checkboxes, anchors, and tables. They have not yet supported high-level functional Web components. This paper presents the automatic generation of Web applications that makes use of customizable visual high-level functional Web components such as frames, framesets, and pivot tables, which are close to our image and make it possible to develop Web applications with less code.

Web applications that need to show a lot of information on one Web page at a time often use inline frames, exchange displayed elements using their tabs, or enable users to see a hidden part of the page using the scroll bars of a Web browser. On the other hand, if Web applications use an HTML frame facility, they can display a lot of information on one Web page at a time and show it to users [1]. The users can easily grasp the outline of the provided information, and at the same time, they can see the contents of the frame of interest in detail by extending the frame to the whole page if they need to. Therefore, some Web applications that need to display a lot of information at a time such as computer-assisted instruction systems, Web-based chat systems, and the Help windows of various kinds of Web applications often use the HTML frame facility [2, 3].

The pivot tables sum up various kinds of data in two dimensions. They reduce the amount of code to be written by developers greatly. The paper describes how to implement high-level functional components as visual Web components and it also presents the system, which is an example of their implementation. This system assists designers in the development of Web applications based on the page-transition framework. This framework models a Web application as a set of Web page transitions, and by using

visual Web components, makes it easier to write processes to be executed when a Web page transfers to another Web page.

2. Image-Oriented Design

2.1. Design Example of a Web Application. In the system, we design a Web application by copying our image of each Web page into a Web page window. Like home page building tools [4], we choose Web components such as hyperlinks, HTML tables, text fields, text areas, submit buttons, framesets, and pivot tables from menus (or buttons), and paste them into a Web page window that corresponds to each Web page. In this section, we consider a simple Web application *favoriteCake* that obtains the information of favorite cakes by means of questionnaires. This application consists of three Web pages, *entryCake, chooseCake*, and *loveCake*.

(1) Customers first enter their names in the *entryCake* page.

(2) They choose their favorite cake from a menu in the *chooseCake* page.

(3) The *loveCake* page shows the cumulative result and the history of their answers.

Figure 1(b) illustrates the design of this application, which consists of *entryCake* page, *chooseCake* page, *loveCake* page, *cake* DB table, and *favoriteCake* DB table. Figure 1(a) shows an example of its execution.

We store the names and the images of a variety of cakes in a database table (hereafter, described as a DB table) *cake*, which is shown in the top right corner of Figure 1(b). The DB table *favoriteCake*, which is shown in the bottom right corner of Figure 1(b), stores the history of customers' answers. It stores the customer's name in the *name* field, and the customer's favorite cake in the *cake* field. For each kind of cake, the *loveCake* page displays the number of the customers who like it by using a pivot table (described in detail in Section 4).

We drag and drop each field of these DB tables into the Web pages. By this, the system inserts field references such as *cake.name, cake.image, favoriteCake.name*, and *favoriteCake.cake* at the dropped positions. At the execution of the application, these field references display the values of the corresponding fields in various forms such as text, images, buttons, and checkboxes. In addition to DB tables that are stored persistently, the system introduces the Program tables that are only used during the execution of the program. Developers can use these Program tables as visual components [5]. By dragging some fields of a Program table into a Web page, their values can be displayed when the application is executed. The Program table can be joined to other Program tables and DB tables. For example, a Program table can be used to store the contents of a shopping cart in online shopping applications, which is only used in a session of the application.

2.2. MVC Architecture. In the system, we develop Web applications based on the Model-View-Controller (MVC) architecture [6] as shown in Figure 2. The MVC architecture isolates business logic from user interface, resulting in an application where it is easier to modify either the visual appearance of the application or the underlying business rules without affecting the other. The model represents the data of the application and the business rules used to manipulate the data; the view corresponds to elements of the user interface; and the controller manages details involving the communication of user actions to the model.

We first design DB tables and Program tables. If database tables have already been created by some DBMS tools, we have only to recall them in the DB table windows of the design workspace. Next, we drag some of the fields of these tables into Web pages to display. Finally, we write business processes in the Web source windows. The system automatically generates methods necessary for accessing the Program table data. In summary, we design the data access layer (model) of the application using DB tables and Program tables, design its presentation layer (view) using Web page windows, and write its business-logic layer (controller) using Web source windows.

In any phase of the development, developers can verify their applications by displaying Web page previews, Web page transitions, Web frame references, and various kinds of field references. As shown in Figure 1(b), the system displays references between components as arrows whose colors indicate the types of the references.

3. Frame and Frameset Components

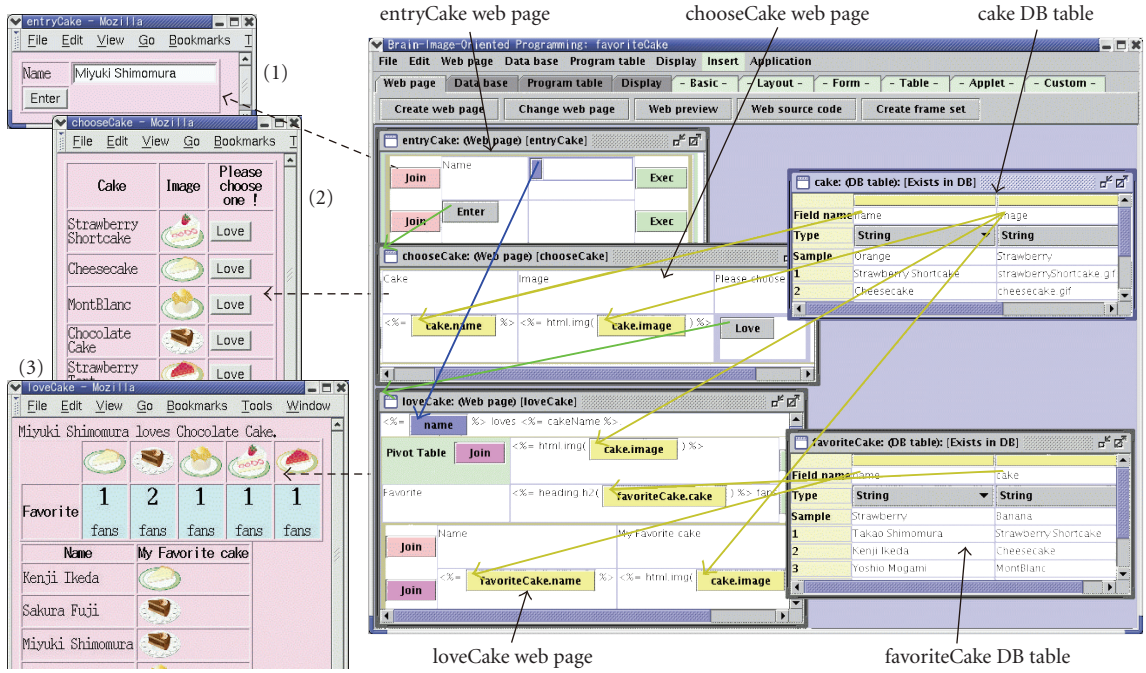
3.1. Requirements for Frames and Framesets. Web pages can contain framesets, and framesets contain frames. Moreover, the Web page that is displayed in a frame can also contain framesets. Because this relationship ranges among multiple pages, it is difficult to grasp with the conventional development environments. To make it easier to develop Web applications that use frame facilities, we take into account the following requirements.

(1) We can nest frames in a frameset more than once, and we can create a frame pointing to another Web page that contains framesets.

(2) We can easily understand the relationship between the frames and the Web pages that are displayed in the corresponding frames.

(3) We can easily change the hierarchical structure of framesets (i.e., which frameset should contain which frames and framesets), easily add and delete frames, and easily change the Web pages that will be displayed inside frames by using the mouse dragging (see Figure 3(c)).

3.2. Introduction of Frameset Page Design Windows. We introduce *Frameset page design windows* as root Web pages that represent framesets, and link each frame included in those windows to an ordinary *Web page design window*. For example, in Figure 3(a), *Frameset page design window* Y has two frames, and its upper frame is linked to *Web page design window* C, and its lower frame is linked to *Web page design*



(a) Execution example (b) Desgin with BioPro

Figure 1: Questionnaire program *favoriteCake*.

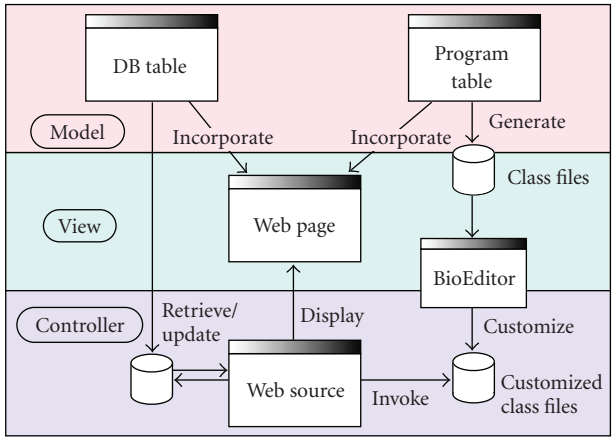


Figure 2: MVC architecture.

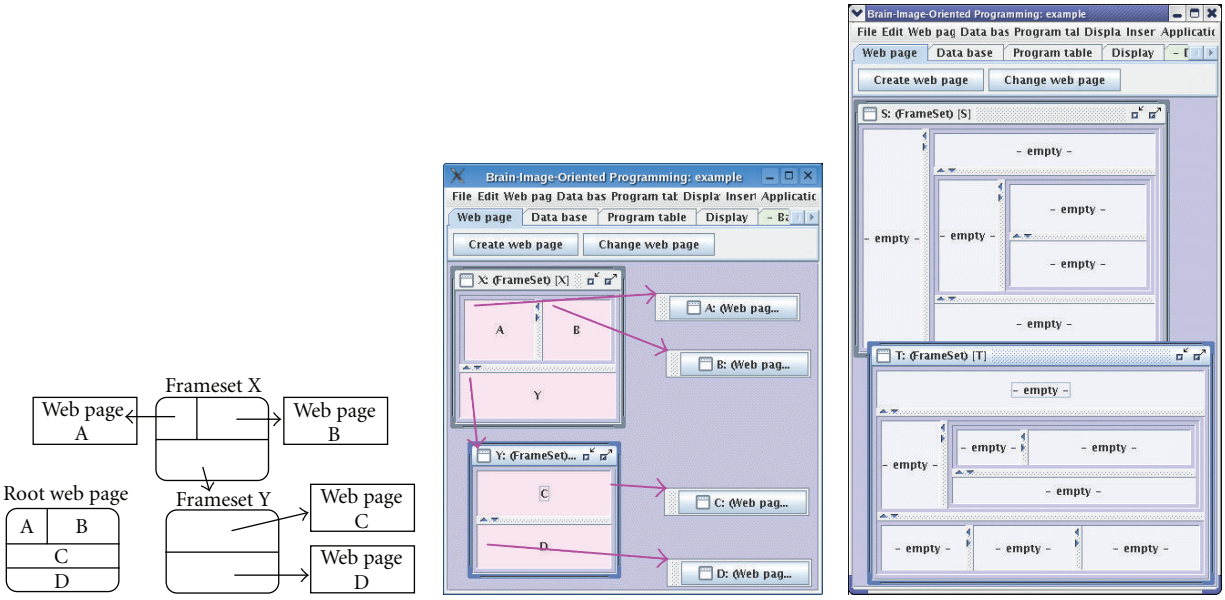
window D. We make it possible to link each frame included in a *Frameset page design window* to not only an ordinary *Web page design window,* but also another *Frameset page design window*. This enables frames to be nested in a frameset more than once.

Figure 3(b) shows two *Frameset page design windows* and four *Web page design windows* created by a programmer using the BioPro system, as illustrated in Figure 3(a). Each frame of the *Frameset page design window* is linked with a line to the *Web page design window* that is displayed in it.

As shown in Figure 3(a), the *Frameset page design window* X has two frames, where its upper frame is divided into two other frames, left and right frames, each of which is linked to *Web page design windows* A and B, respectively. Its lower frame is linked to another *Frameset page design window* Y. When we execute the Web application that contains these pages, the *Frameset page design window* X is displayed as a root Web page that includes four Web pages A, B, C, and D as shown in Figure 3(a). In a *Frameset page design window*, we can easily divide, delete, and exchange frames by clicking or drag-and-dropping the mouse. Figure 3(c) shows an example of nested frames in *Frameset page design windows*.

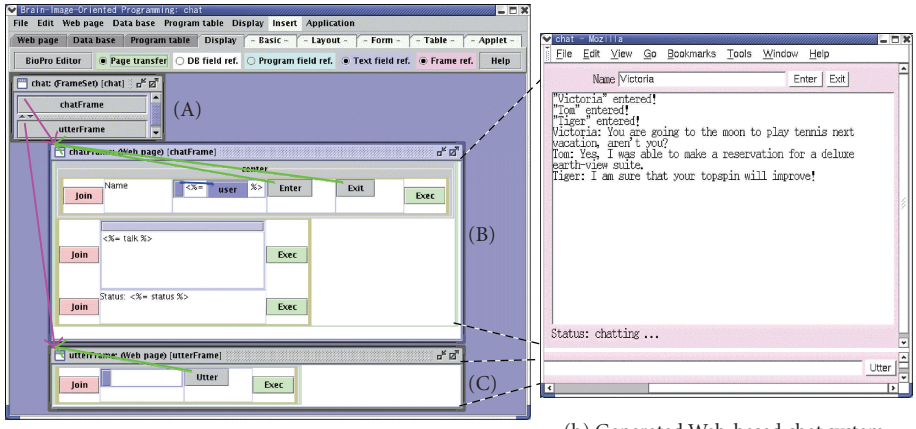
3.3. Visual Programming for Frames and Framesets. As an example of a Web application that uses frames, let us consider a simple Web-based chat system. As shown in Figure 4(b), this Web page consists of two frames. The upper frame has a text field to enter a name, and buttons to enter and exit a chat room. Below them, it has an area to display the contents of chatting and its status. The lower frame has a text field to enter a chat message, and a button to send the message. To update the contents of chatting that vary any minute, the upper frame keeps being refreshed at a certain period of time. Because this Web page is divided into these two frames, the lower frame is not affected by refreshing the upper frame even when a user is entering a message.

Figure 4(a) shows an example of visual programming of this Web-based chat system. The window (A) in the top left corner of Figure 4(a) is a *Frameset page*, which corresponds to the *Frameset page* Y in Figure 3(a). The upper frame is linked to chatFrame *Web page* to display chatFrame



(a) Example of page design for framesets (b) Frameset page design windows for (a) (c) Example of nested frames in framesets

FIGURE 3: *Frameset page design windows* created by the BioPro system.



(a) Visual programming using frameset pages (b) Generated Web-based chat system

FIGURE 4: Visual programming using *frameset pages*.

Web page. The lower frame is linked to utterFrame *Web page* to display utterFrame Web page. Windows (B) and (C) of Figure 4(a) show the *Web pages* of chatFrame and utterFrame, respectively.

4. Pivot Table Components

4.1. Visual Programming for Pivot Tables. This section introduces a pivot table that sums up various kinds of data in two dimensions and describes how we use this table as a visual Web component. The pivot table consists of three cells, row, column, and data. Into each cell of the pivot table, a field of either a DB table or a Program table is dragged. At the execution time of the application, the row and column cells of the pivot table are dynamically expanded to display the values of the corresponding field of the DB table or the Program table. The data cells of the expanded pivot table display the corresponding field values for each row and each column in a specified form.

Figure 5 illustrates how the pivot table is expanded at the execution time of the application. The way of expanding the pivot table changes depending on whether or not a field of a DB table (or a Program table) is assigned to each cell of the pivot table. Let $\{d_{ij}\}$ be a set of field values in the data cell that corresponds to each row and column. The value of the set $\{d_{ij}\}$ is displayed in the data cell in a variety of forms such as *Standard* (the sum of the element values), *Counter* (the number of the elements, that is, $\#\{d_{ij}\}$), and *Image* (the image the first element's value refers to). In addition,

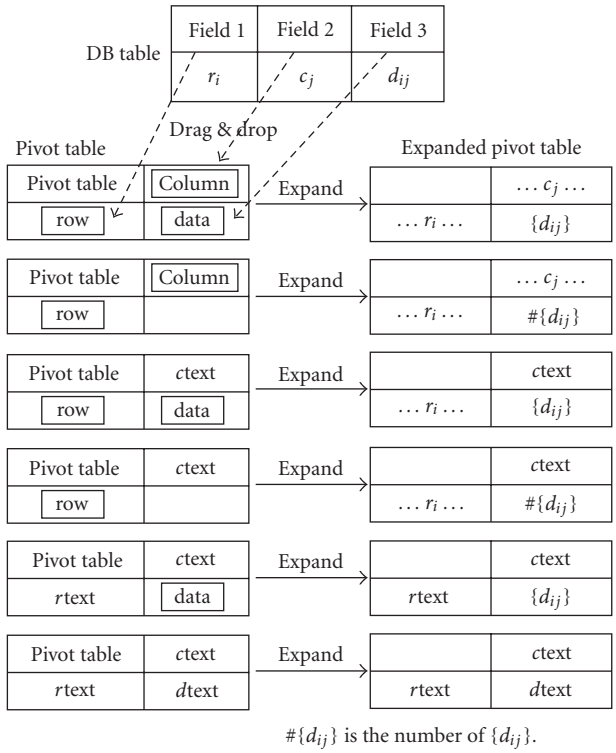


Figure 5: Pivot table expansion.

developers can customize a way of displaying the data cell by creating a class and its methods that define how the data cell should be displayed. For example, as shown in Figure 6(a), we can display the name and the picture of the person whose name is the value of d_{ij}. If we further customize it to specify the data cell's format, background color, and component to be displayed as shown in Figure 6(b), checkboxes used in the meeting room reservation system will be changed to buttons as shown in Figure 6(c). Customization of visual functional Web components will be described in more detail in Section 5.

4.2. A Reservation System for Meeting Rooms. Figure 7 shows an example of design for the meeting room reservation system that uses a pivot table, and Figure 6 shows an example of execution of this application. This meeting room reservation system works as follows.

(1) Customers enter a date for making a reservation, and their names and passwords.

(2) The system displays reservations for that date using the pictures of the people who have reservations.

(3) The customers check a vacant room off to make a reservation or check their own pictures to cancel the reservation.

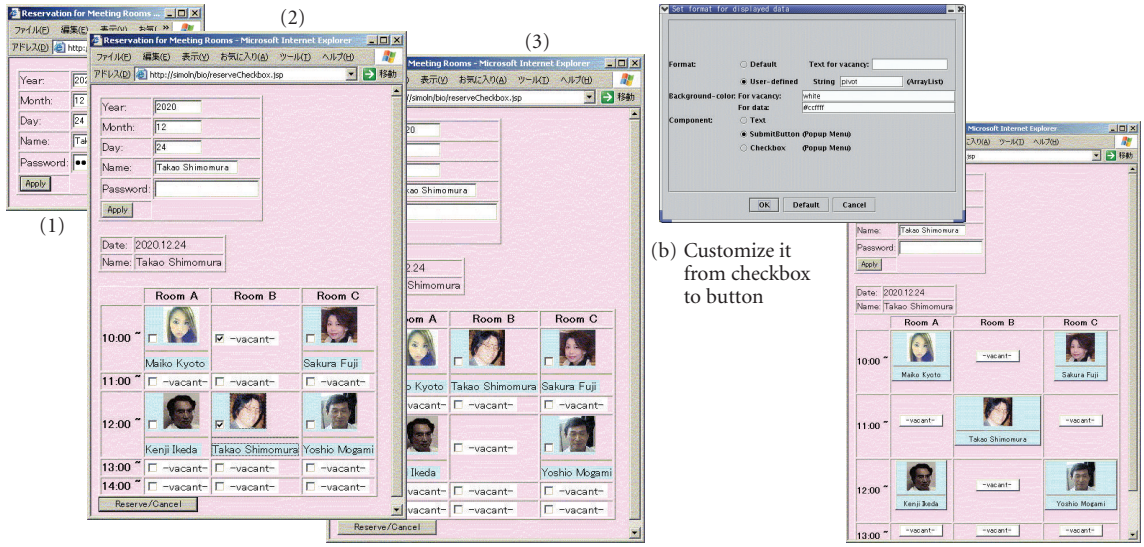
We record reservations for meeting rooms in a database. The DB table *meeting* (in the top right corner of Figure 7) records meeting rooms (*room*), periods of time (*hour*), and people who have reservations (*name*). This DB table *meeting* is a virtual table only used for designing the Web page, and the real database table is dynamically determined at the execution time of the application. The pivot table is arranged in the bottom left corner of Figure 7. We drag and drop the *room* field of the DB table *meeting* into the column cell of the pivot table to insert a field reference *meeting.room*. Similarly, we insert a field reference *meeting.hour* to the row cell of the pivot table, and a field reference *meeting.name* to the data cell. As shown in Figure 6(a), when this Web application is executed, this pivot table will be dynamically expanded so that each row will display a period of time; each column will display the name of a meeting room; and each data cell will display the name and the picture of the person who has a reservation for the corresponding row and column.

In this example, we have displayed the data cells as checkboxes. Instead, we can also display them in a variety of forms such as text and buttons. If we display the data cells as submit buttons, we will not need the *Reserve/Cancel* button, and customers can make a reservation immediately by clicking on a vacant button, and can cancel the reservation immediately by clicking on their own pictures.

5. Customization of Visual Functional Web Components

5.1. Definition of Web Components. In the proposed system, we can create a new Web component and add it to the system. When we design a Web application, we can use these created Web components as visual components in the same way as we use other components such as pivot tables, DB tables, and Program tables. To create a new component, we define a class that extends WebComp class. We have only to define several methods to override those defined in the super WebComp class. Table 1 shows some methods the super WebComp class provides. We define a component name, write code to create a visual component (a JComponent object in Java), and specify how to change the component's properties. For the verification of relationships between components, we write the code that obtains the relationships of the new component with other components. For code generation, we define HTML/JSP code to display this component in a Web page, and so on.

For example, we here create a new Web component "Autograph". When we choose this component, a dialog will open to enter an autograph. Then, this component will be inserted into a Web page design window. When this Web application is executed, the entered autograph will be shown in italic. Figure 8 shows an Autograph class for this component. This class extends a WebCompAdapter class, which extends the WebComp class a part of whose methods have been shown in Table 1. The Autograph class defines getMenuName( ) method to specify its component name "Autograph", which will be displayed as the name of a menu item that corresponds to this component. CreateWebComp( ) method creates a JLabel object to display this visual component in a Web page design window. ChangeProperties( ) method invokes specifyProperties( ) method,



(a) Excecution of meeting-room reservation system with checkboxes

(c) The same system with buttons

Figure 6: Execution of meeting room reservation system.

Table 1: Definition of customizable visual Web components.

	Methods	Description
Creation	String getMenuName( )	Define a component name
	JComponent createWebComp( )	Create a visual component
	void changeProperties(JComponent comp)	Display a dialog to change the properties of a component comp
Verification	void addRefers(ArrayList refers)	Check and add a relationship of this component with other components
Code generation	void addParameterNames(HashMap pageToParams)	Add the names of parameters this page sends
	void addInitializeParameterCode(HashMap pageToCode)	Define code that will be executed to analyze received parameters if necessary
	void addJSP(JSP jsp)	Define HTML/JSP code to display a component in a Web page

which displays a dialog to enter this component's properties. AddJSP( ) method defines how to display this component in a Web page when this Web application is executed. This addJSP( ) method defines a <h2 style="font-style:italic"> tag to display the specified autograph in italic.

5.2. Customization of Pivot Tables. A pivot table is one of visual functional Web components. High-level functional Web components are easy to use, and they can easily produce even complicated display of Web pages. However, the higher level they are on, the less flexible they will be. Therefore, it is important to provide a mechanism to customize those components. To customize the display of data cells in a pivot table, we can specify the name of a method for customization as shown in Figure 6(b). Figure 9 illustrates an example of such a method, pivot( ). This method makes it possible to display a person's name and picture in the data cells of the pivot table as shown in Figures 6(a) and 6(c). Pivot( ) method has a parameter dataList of type ArrayList, which contains a list of data corresponding to this data cell. In this meeting room reservation system, it contains only one value, which is the name of the person who reserves a meeting room that corresponds to this data cell. This pivot( ) method retrieves the person's picture from DB table "person", and returns HTML code that consists of an <img> tag, a <hr> tag, and the person's name.

6. Implementation of Visual Functional Web Components

6.1. Implementation of Frameset Hierarchy. To develop *Frameset page design windows*, we define FrameSet class and Frame class. As shown in Figure 10, both of FrameSet class and Frame class extend abstract class FrameOrSet. In

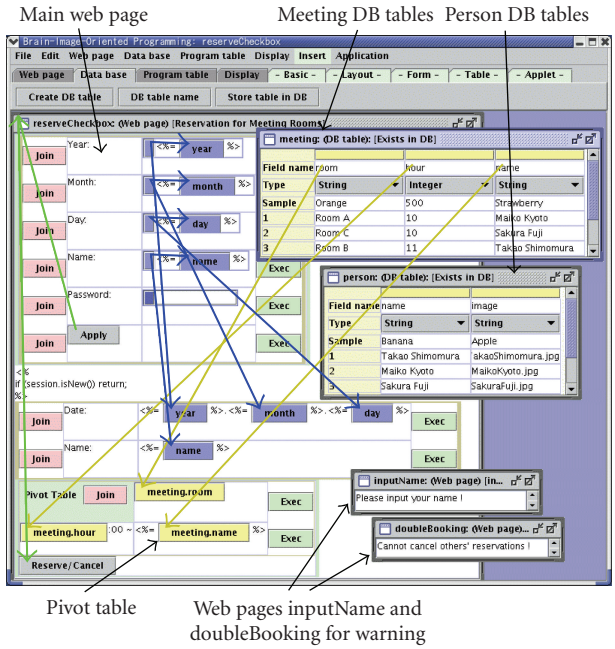


Figure 7: Design of meeting room reservation system.

```
public class Autograph extends WebCompAdapter {
 public String getMenuName( ) { return "Autograph"; }
 public String autograph = "Quan Liang Chen";
 public JComponent createWebComp(WebPage
 containerWebPage, CellTextPane containerTextPane) {
 JLabel label = new JLabel( );
 label.setBackground(new Color(200, 255, 200));
 label.setBorder(new LineBorder(Color.green));
 label.setFont(new Font("SansSerif", Font.ITALIC, 10));
 if (!specifyProperties(label)) return null;
 return label;
 }
 public void changeProperties(JComponent comp,
 int x, int y) {
 specifyProperties(comp);
 }
 public void addJSP(JSP jsp) throws Exception {
 String html = "<h2 style=\"font-style:italic\">" +
  autograph + "</h2>; \n";
 jsp.code += html;
 }
 private boolean specifyProperties(JComponent comp) {
 String title = "Specify autograph";
 String[ ]names = new String[ ]{ "Autograph", };
 String[ ]values = { autograph, };
 String[ ]defaultValues = { "Who am I?", };
 values = new InputTextDialog(title, names, values,
  defaultValues).getInputValues( );
 if (values == null) return false;
 autograph = values[0];
 ((JLabel) comp).setText(autograph);
 return true;
 }
}
```

Figure 8: Definition of Web component "Autograph".

```
private static final String dbTableName = "person";
private static final String nameFieldName = "name";
private static final String imageFieldName = "image";
private static final String vacant = "-vacant-";
public String pivot(ArrayList dataList) throws Exception {
 String html = null;
 int size = dataList.size( );
 if (size == 0) {
  html = vacant;
 } else if (size >= 1) {
  String name = (String) dataList.get(0);
  java.sql.Statement stm = con.createStatement( );
  ResultSet result = stm.executeQuery("select" +
   imageFieldName + "from" + dbTableName +
   "where" + nameFieldName + "=" + "'" + name + "'");
  if (result.next( )) {
   String image = result.getString(1);
   html = "<img src=\'" + Std.resourceDirName( ) +
    image + "\" width=\'50\' >;<;hr>" + name;
  } else {
   html = name;
  }
 }
 return html;
}
```

Figure 9: Definition of a method pivot( ) to customize the data cell display of a pivot table.

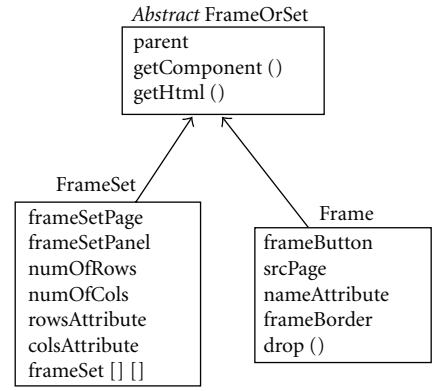


Figure 10: FrameSet and Frame that extend abstract FrameOrSet.

FrameOrSet class, variable parent refers to its parent frameset or frame. GetComponent( ) method returns JComponent that displays its frameset or frame. A frameset is displayed as JPanel that contains a JSplitPane, and a frame is displayed as a button in a *Frameset page design window*. GetHtml( ) method returns the HTML code that represents its frameset or frame. In FrameSet class, frameset[ ][ ]array points to its child frameset or frame. In Frame class, variable srcPage points to a *Web page* or a *Frameset page*. Drop( ) method enables a user to drag and drop a frame to another frame to exchange them.

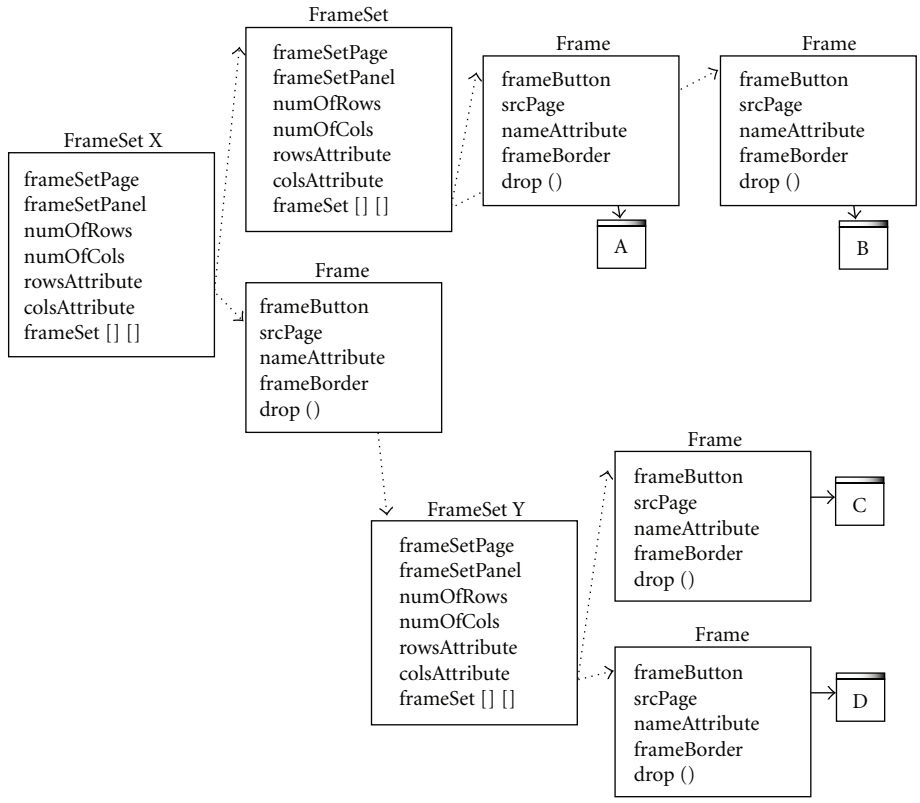


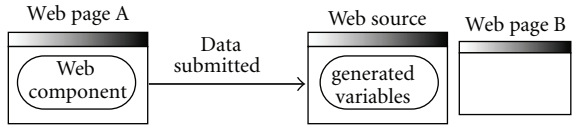
Figure 11: Frameset hierarchy.

Figure 11 illustrates the frameset hierarchy of the two *Frameset pages* shown in Figure 3(b), where Frame and FrameSet instances also have a pointer that refers to their parent frame or frameset because they extend the abstract FrameOrSet class shown in Figure 10.

6.2. Page-Transition Framework. Each Web page is designed using visual Web components. The reference relationships of Web components inside the same Web pages or between Web pages are created by the drag and drop operations of the components, and represented by the arrows whose colors show the types of the references. When a request is submitted from a Web page S, as a result, if a Web page T is displayed, we call page S a source page, and page T a target page. When a source page transfers to a target page, some of the Web components in the source page automatically submit some data with the request. The target page automatically analyzes those submitted data, generates some variables, and stores the analyzed results in the generated variables. In this target page, by using those generated variables, developers can easily write actions to be executed when the target page comes from each one of its source pages.

As indicated in the Struts framework [7], the Web application controlled by JavaServlet containers is composed of a sequence of display of a JSP page, receipt of a request, execution of an action, and forwarding the request to the next JSP page. The system provides the page-transition framework for Web applications, where form data can be automatically submitted and analyzed, and actions to be executed in a target page can be written for each one of the source pages of the target page. Figure 12 illustrates how form data are automatically submitted and analyzed. In the system, Web pages are composed of a variety of Web components such as text fields, field references from program and DB tables, submit buttons, HTML tables, and pivot tables. When a request is submitted from a source page A to a target page B, some of Web components included in source page A automatically submit their data. Target page B automatically receives these submitted data, analyzes the contents of the data, transforms them into appropriate values, and generates some variables to store those values. For example, a text field Web component whose name is "name" submits the text that is input in this field, and the target page receives this submitted text, generates a variable whose name is "name", and stores the text in this variable.

For example, in the Web-based chat system shown in Figure 4, a text field Web component whose name is "user" submits the text that is input in this field, and the target page receives this submitted text, generates a variable whose name is "user", and stores the text in this variable. In the *Web source window* of chatFrame *Web page*, we can refer to variables user, enter, exit, and textarea as predefined variables, which are generated from the visual design of the Web pages by the



Web component	Data submitted	Generated variables
Text field	name	String **name ;**
Hidden field	cakeName	String **cakeName ;**
Pivot table (button type)	meetingHour meetingRoom	int **meetingHour ;** String **meetingRoom ;**
Pivot table (checkbox type)	meetingName	int[ ] **meetingHour ;** String[ ] **meetingRoom ;** boolean **meetingName ;**
Program table	-	Cart **cart ;**
DB table	-	String **meeting ;**

Figure 12: Automatic generation of variables.

	Web page A (contains Pivot table) ---->	Web page B (comes from A)
Submit Button	<input type="hidden" name="**meetingHour**" value="<%= rowValue %>"> <input type="hidden" name="**meetingRoom**" value="<%= columnValue %>">	**meetingHour** = Integer.parseInt(request.getParameter ("meetingHour")); **meetingRoom** = request.getParameter ("meetingRoom");
Checkbox	<input type="checkbox" name="**meetingName**" value="<%= cellNo %>">	**meetingHour**[i] = rows[cellNo / columns.length]; **meetingRoom**[i] = columns[cellNo % columns.length];

Figure 13: Automatic interpretation of submitted data for pivot tables.

system. For example, variable user has the value of the name text field. We do not need to care about the inconsistency between the parameter names of a sender and a receiver. Using these predefined variables, we write necessary actions for the processes of entering and exiting the room. In the *Web source window* of utterFrame *Web page*, we write the process for adding the submitted message by referring to the contents of the message as a predefined variable.

The pivot table Web component that sums up data in two dimensions (shown in Figure 7) submits the data that identify which data cells are selected or clicked. The target page automatically receives these submitted data and analyzes the contents of the data. When the data cells of the pivot table are displayed as buttons, as shown in Figure 13, the pivot table Web component assigns the row value (*hour*) that corresponds to the selected cell to parameter *meetingHour*, and the column value (*room*) that corresponds to the selected cell to parameter *meetingRoom*, and sends these parameters. The target page receives these parameters and automatically generates variables "int meetingHour" and "String meetingRoom " to store them. When the data cells of the pivot table are displayed as checkboxes, the pivot table Web component assigns the selected cell number (*col* + *columns.length* ∗ *row*) to parameter *meetingName*, and sends this parameter. The target page receives a sequence of the selected cell numbers, obtains the number of the selected cells, and automatically generates two variables "int[ ] meetingHour" and "String[ ] meetingRoom". Then, from the selected cell numbers, it obtains the row numbers and the column numbers of the selected cells, and stores their row and column field values in *meetingHour[i]* and *meetingRoom[i]*, respectively. When no fields are assigned to the row and column cells of the pivot table, the pivot-table Web component generates variable "boolean meetingName" that indicates whether or not the data cell is selected. This mechanism enables the target page to know the cells of the pivot table selected in its source page.

The automatically generated variables consist of not only the variables that are generated by analyzing received requests. In addition to these variables, the target page automatically generates the variables that refer to Program tables and DB tables when the target page refers to some fields of those tables. When the target page refers to a field of a Program table *Cart*, it generates the variable *cart* that refers to the *Cart* object taken out of the session. When the target page refers to a field of a DB table *meeting*, it generates the variable *meeting* whose initial value is also "meeting". When the database table to be dealt with is dynamically changed at the execution time of the application (see Figure 6(a)), we have only to assign the real database table's name to this variable *meeting*. The field references of the DB table display their field values according to the database table variable *meeting* points to.

6.3. Actions Defined in the Web-Based Chat System. In the *Web source window* that corresponds to chatFrame *Web page*, we can here define actions to be executed when control transfers to this page. The "Predefined vars:" column of the *Web source window* shows some variables that contain submitted data and these are automatically generated by the system. Figure 14 shows the *Web source window* that corresponds to utterFrame *Web page*. Although a typical page transition (a default transition) is specified as an arc from one *Web page* to another *Web page*, the other transitions (as when a failure occurs) can be specified in this *Web source window* as an action. The actions defined here will be synthesized with designed Web components to generate the program code of the Web application (see Figure 17).

Figure 15 illustrates how predefined variables are generated for chatFrame and utterFrame *Web source windows*. When we click on "enter" or "exit" button in chatFrame Web page, a request is submitted to chatFrame Web page itself. Therefore, variables enter, exit, textarea, user are generated as predefined variables in chatFrame *Web source window*. When we click on "utter" button in utterFrame Web page, a request is submitted to utterFrame Web page itself. Therefore, variable utter for the text field, whose name attribute is "utter", is generated as a predefined variable in utterFrame *Web source window* as shown in "Predefined vars:" column

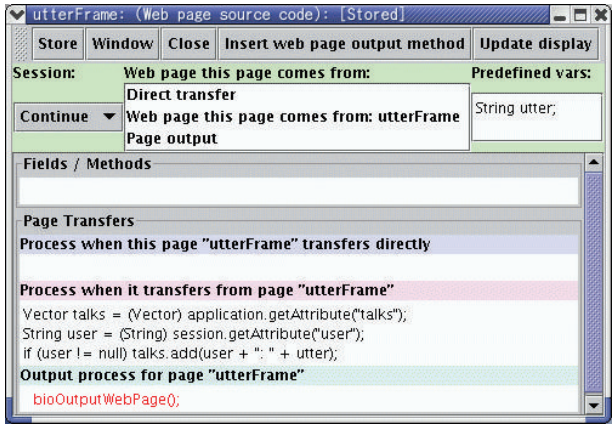


Figure 14: *Web source window* for utterFrame page.

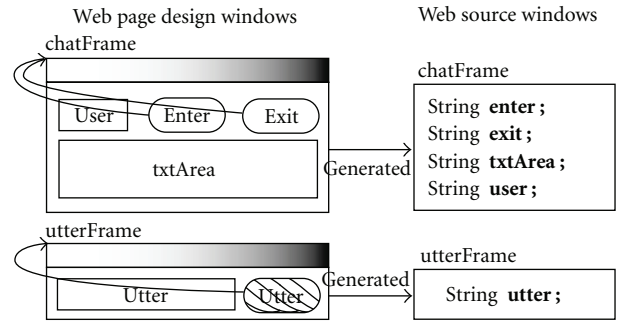


Figure 15: Predefined variables for chatFrame and utterFrame *Web source windows*.

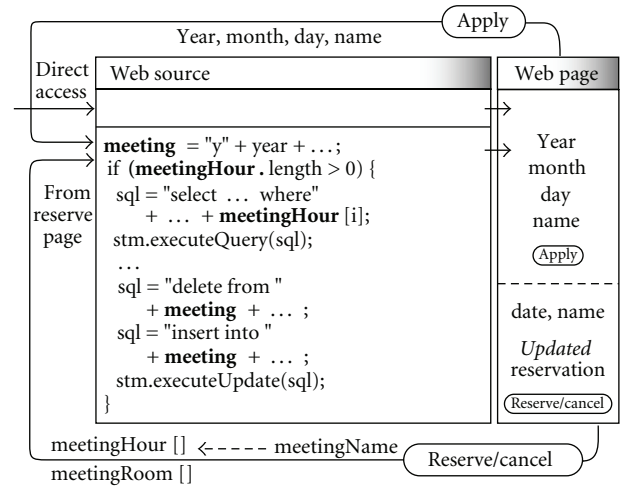


Figure 16: Actions for the *reserveCheckbox* page.

of Figure 14. Although utterFrame Web page contains "utter" button, no variable for that button is generated because this button's name attribute has not been specified in this Web application.

6.4. Actions Defined in the Reservation System. As shown in Figure 7, the reservation system for meeting rooms consists of three Web pages, the *reserveCheckbox* page that displays the reservation, the *inputName* page that prompts customers to enter their names, and the *doubleBooking* page that refuses their requests. Figure 16 illustrates an example of action code written in the Web source window that corresponds to the *reserveCheckbox* page. The *reserveCheckbox* page is accessed directly and it also comes from the *reserveCheckbox* page itself.

(1) When this page is accessed directly, we do nothing, that is, no action needs to be written. In this case, the text fields for entering the date and the customer's name and password are displayed.

(2) When the *reserveCheckbox* page comes from itself, we have the following two cases:

(2-1) the customer enters the date and his or her name and then clicks on the *Apply* button;

(2-2) the customer selects some of the checkboxes displayed and then clicks on the *Reserve/Cancel* button.

In the case of (2-1), the date and the customer's name are sent, and variables *year, month, day*, and *name* are automatically generated. Variable *meeting* that points to the DB table, which is used in the pivot table, is also generated. We construct a real DB table name from variables *year, month*, and *day*, and assign the real DB table name to variable *meeting*. To display the reservation for that date, the system dynamically expands the pivot table by retrieving the DB table this variable *meeting* points to.

In the case of (2-2), parameter *meetingName* that indicates which checkboxes are selected is sent, and variables *meetingHour[ ]* and *meetingRoom[ ]* are automatically generated that store the field values corresponding to the selected cells. In this case, the condition that *meetingHour.length* is greater than zero becomes true. We write some code to update the database. First, when the customer's name has not yet been entered, we display the *inputName* page as shown in Figure 7. Next, we retrieve the database using the values of variables *meetingHour[i]* and *meetingRoom[i]*. When the customer tries to cancel others' reservations, we display the *doubleBooking* page. Finally, we execute *delete* SQL statements for cancellations and *insert* SQL statements for reservations to update the database.

From a *Web page*, code for displaying the corresponding Web page will be automatically generated. If it refers to some fields of a *DB table*, code for retrieving records from the corresponding DB table will be automatically generated. Submitted data can be automatically received and analyzed, and then the variables that contain received data will be automatically generated. The other logic such as updating a DB table needs to be written manually as an action in a *Web source window*.

6.5. System Configuration. The system exists on a client machine, and a Web server/Servlet container and a database server exist on server machines. The system generates JSP pages and Java class files from designed Web pages, Program

tables, DB tables, and Web page source files. It then uploads them to the Web server. To run a Web application, the system invokes a Web browser so that it will display the first JSP page, which is either automatically determined or chosen by a user. To display the preview of a Web page, it generates a JSP page for preview, and uploads the JSP page to the Web server. Then, the system itself accesses the JSP page, and shows its output result in a Web preview window.

The BioPro system is not based on any of the existing frameworks. It proposes a visual programming framework, where a generated application only uses Servlets and JSPs. To connect to a PostgreSQL database server, the generated application uses JDBC. The system generates Web application program code from the visual design of a Web application. As shown in Figure 2, we first visually design the contents of each of Web pages in their *Web page design windows*. When we use database tables, we first visually design those database tables in *DB table design windows*. Instead, we may specify the names of existing database tables to display them in *DB table design windows*. We drag and drop the fields of a database table from a *DB table design window* to a *Web page design window* so that these fields will be displayed in the corresponding Web page. When we use a table in the program that exists only during the execution of the program, we visually design the contents of this table in a *Program table design window*. For example, we design a table of an online shop cart in this *Program table design window*. The system automatically generates JavaBeans code from these *Program tables*. We next write the actions that are executed when control transfers to a Web page in the *Web source window* that corresponds to the Web page. Control may transfer to one Web page from multiple Web pages. In the *Web source window* of a *Web page*, for each Web page control transfers from, we can write a necessary action, which is executed when control transfers from the Web page to this Web page. From these resources, the BioPro system automatically generates Servlets, JSP pages [8], and Java classes that compose a Web application in a client side, and uploads them to the Web server together with other resources such as image files, and customized Java classes. To start the Web application, the system then runs a Web browser to make it send a request to the entry Web page of the Web application.

The system generates code for connecting a database server, retrieving records, receiving submitted data, and displaying Web components designed in a *Web page*. These processes will be run as threads. After these threads complete, it will start a Web browser to access the entry Web page of the Web application. The entry Web page is automatically determined as a Web page that does not have its preceding page.

Figure 17 shows a method to automatically generate the program code of the Web-based chat system from the visual design created in Section 3.3 "Visual programming for frames and framesets". The part of the program code printed in italics in Figure 17 represents the code that was automatically generated by the BioPro system. We first designed a Frameset page chat in the *Frameset page design window*, and designed two Web pages that are displayed

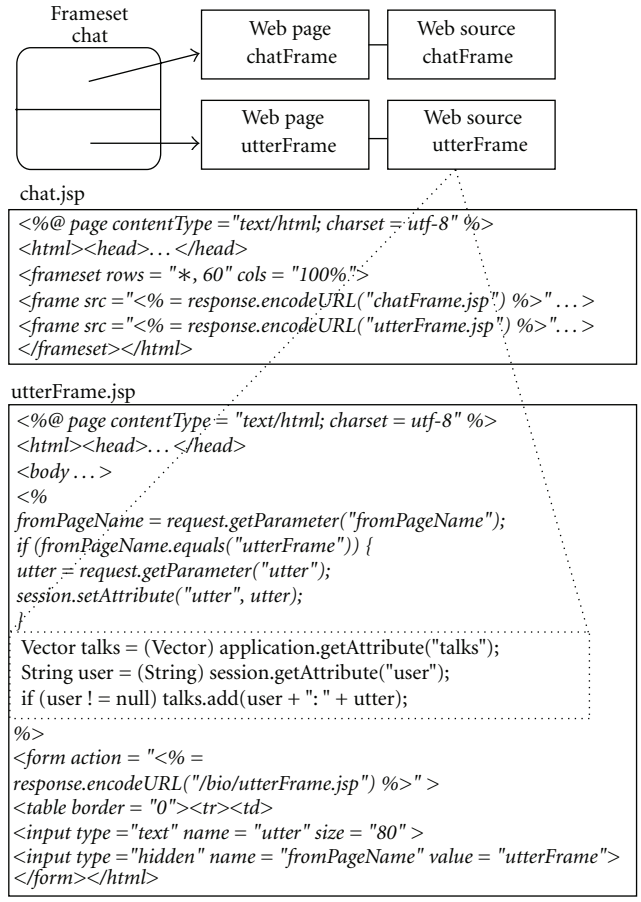


Figure 17: Automatic generation of Web application program code.

inside its frames in the *Web page design windows*. In the *Web source windows* of the corresponding Web pages, we then wrote the necessary actions that would be executed when control transfers to each of the Web pages by referring to predefined variables (e.g., "utter" as shown in Figure 14) that were automatically generated by the BioPro system.

The BioPro system generates the program code that composes the Web application from these pieces of design information. It generates a JSP page "chat.jsp" from the *Frameset page* chat, and as shown in Figure 17, it generates a JSP page "utterFrame.jsp" by synthesizing the utterFrame *Web page* and its *Web source*.

7. Observations on the Proposed System

7.1. Applications of Functional Web Components. As shown in Figure 4, the structure of the Web application is displayed visually, and this makes it easier to understand the program and efficient to modify and debug the program. When we design framesets and the frames that the framesets contain, we can easily create them, and divide, delete, and exchange frames by clicking or drag-and-dropping the mouse. In addition, we can create another frameset in a different *Frameset page design window*, and make the parent frame refer to the created child frameset. This simplifies the design

Table 2: Comparison with other tools.

Facilities	IDE	Visual IDE	BioPro
Code-based HTML components	○	○	○
Visual HTML components	—	○	○
Customizable visual high-level Web components	—	—	○
Visual DB tables	—	○	○
Visual Program tables	—	—	○
Assist in writing actions	—	—	○
Form data generation	—	—	○
Test/Preview Web pages	○	○	○
Debugging support	○	○	—
Refactoring support	○	○	—

of framesets and the frames that the framesets contain. We can also easily see the hierarchy of the frames in the *Frameset page design windows*, and see the contents of the frames in the *Web page design windows* that are pointed at from the corresponding frames in the *Frameset page design windows*.

The pivot tables can sum up a various kinds of data in two dimensions. Developers can customize the form for displaying the summed up data that correspond to each row and column. The information of selected cells in the pivot table is automatically sent to a target page, and in the target page, developers can easily obtain this information through automatically generated variables. We think that the pivot tables can be applied to various Web applications. The application *favoriteCake* that has been shown in Figure 1 is also one example of the applications that use pivot tables. Using a pivot table, the *loveCake* page displays the number of the customers who like each kind of cake. This application only uses two of the three cells in the pivot table. The column cell is expanded to display the images of the cakes that are stored in the DB table *cake*. The data cell is expanded to display the number of the customers who like each kind of cake using the display form *Counter* (see Section 4.1 "Visual programming for pivot tables"). To do this, we click on the Join button of the pivot table and enter a condition "cake.name = favoriteCake.cake" to join two DB tables *cake* and *favoriteCake*. The system automatically executes the following SQL statement to expand the pivot table:

```
select cake.image, favoriteCake.cake from
cake, favoriteCake where cake.name = favorite-
Cake.cake.
```

For each kind of cake (*cake.image*), the number of the values the field *favoriteCake.cake* has is equivalent to the number of the customers who like that cake.

Web source windows show some automatically generated variables in their predefined variable column. These automatically generated variables that have the values of the query data submitted to the server enable developers to easily write actions in the fields/methods columns and the page transfers columns of the *Web source windows*. This avoids a mismatch problem between variable names that are written in the form tags of JSP pages and in the methods of Servlets.

7.2. Comparison of Facilities with Other Tools. Web application development tools are broadly classified into two groups, text-oriented and visual-oriented. BioPro is a kind of visual-oriented tool with high-level functional Web components. For example, NetBeans [9] is a text-oriented tool, and Sun Java Studio Creator [10] is a visual-oriented tool. When we choose a Web component from a palette, NetBeans will generate and display its corresponding HTML code in a source window while Sun Java Studio Creator will paste its corresponding visual component in a form window. Table 2 shows the comparison of the BioPro system with a conventional text-oriented IDE, NetBeans [9] and a conventional visual-oriented IDE, Sun Java Studio Creator [10] concerning the facilities that assist in the development of Web applications. IDEs are integrated development environments that include editors, compilers, debuggers, project management, various kinds of source code templates, refactoring facilities, and application servers. On the other hand, unlike commercial software tools, the proposed system BioPro is not a comprehensive Web development tool. It has been developed to evaluate visual programming for high-level functional Web components. These high-level functional Web components can reduce the amount of codes required to display the components in Web pages, and perform actions. We think that it will be much easier to develop Web applications if such customizable high-level functional Web components are available even in any type of software development environment.

7.3. Code Generation Efficiency of Functional Web Components. To evaluate the efficiency and the ease of the development with the system, we developed several typical Web applications using the system, and compared it with the development using an existing integrated development environment IDE [11] and Struts [7]. Each of four programmers developed several sample programs with IDE, BioPro, and Struts in this order. Those sample programs include Web applications (1) selectFruit (selection of fruits), (2) onlineShop (online shopping using a Program table), (3) reserveRoom (reservation for meeting rooms using one pivot and two DB tables), (4) favoriteCake (a questionnaire program about favorite cakes using a pivot table), and (5)

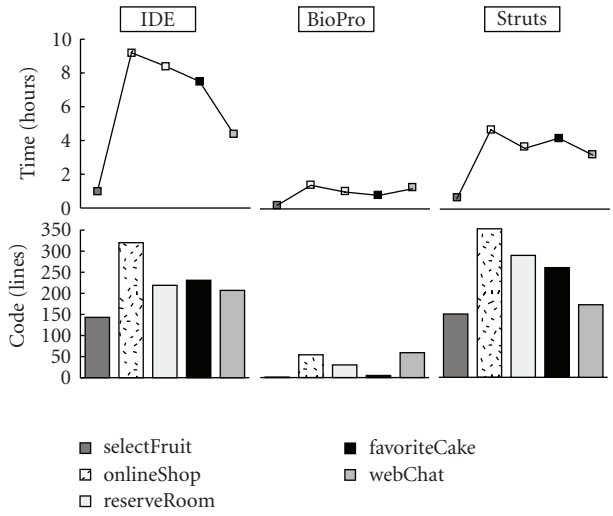


Figure 18: Time and lines of code required.

webChat (Web-based chatting using a frameset). Figure 18 shows the time and the lines of code required to develop these applications on average. The time indicates how many hours it took to make an application, test it, and make sure of its execution result. The lines of code indicate the total lines of JSP and Java code required to develop each application. Because of the learning bias, the time required might have been advantageous to the subsequent tools, BioPro and Struts. However, by using visual programming of functional Web components, we were able to greatly reduce the code that was required to develop Web applications.

The application "webChat" is a Web-based chatting program that uses a frame set. In the Web-based chat system which is a typical example of conventional development, it required 212 lines of code (10 for chat.jsp, 118 for chatFrame.jsp, and 84 for utterFrame.jsp). This method reduced it to 63 lines of code, which is about one-third. Talking about the efficiency, it only took about fifteen minutes to design the chat Frameset, chatFrame Web page, and utterFrame Web page, which are shown in Figure 4(a).

The application "reserveRoom" is a meeting-room reservation system that makes use of a pivot table, where the data cells are displayed as either of checkboxes and buttons. The application "favoriteCake" shown in Figure 1 is a questionnaire program of favorite cakes that also uses a pivot table.

The applications that used pivot tables reduced the amount of code greatly. Pivot table components automatically retrieve the database to expand the pivot tables, and submit the information of selected cells. Target pages can automatically receive and interpret this information and generate some variables to store it.

8. Related Work

This section gives the outlines and brief discussions of text-oriented IDEs, Web application development frameworks, visual development environments for client programs, visual-oriented IDEs, and model-based approaches.

8.1. Text-Oriented IDEs. As for world-widely used systems that assist in the development of Web applications, there are several IDEs such as Sun One Studio [11], IntelliJ IDEA [12], CodeWarrior [13], Eclipse [14], NetBeans [9]. IDE is an integrated development environment that includes editors, compilers, debuggers, project managements, various source code templates, and application servers. It assists in the development of software using object-oriented programming languages like Java. Besides these systems, Zope [15] is an application server with which users can easily develop Web applications using a Web browser that is connected to a Zope server. FAR [16] is an end-user visual language to assist in the development of the Web applications that use spreadsheets. DENIM [17] is a sketch-based visual language to assist in the early stages of Web site design using the graph representation that consists of Web pages as nodes and the dependencies between the Web pages as arcs. JWIG [18] provides a session model and a flexible mechanism for dynamic construction of XHTML documents. With JWIG, a Web application can be written as a single thread using an extension of Java. PageGen [19] provides a scheme for dynamic generation of Web pages.

These text-oriented IDEs have a variety of functions. However, they only provide fundamental Web components such as textfields, buttons, checkboxes, anchors, and tables, and Web applications are developed using text-based languages such as XHTML, JSP, JSP tag libraries, and Java. On the other hand, the proposed system provides visual high-level functional Web components, and this paper has also presented how to implement and customize these components in a flexible manner.

8.2. Web Application Development Frameworks. Several frameworks for efficiently developing Web applications have been proposed. Struts [7] provides a framework for building Web applications that consists of such components as views, controllers, and actions. Separately from business processes, users can easily write code for verifying form data and can specify target actions to which requests are forwarded. Tiles is a framework for creating Web pages that separates Web page layouts and their contents. It is used together with the Struts framework to create JSP pages to which requests are forwarded. Tiles makes it easier to change the look and feel of a Web site. JavaServer Faces [20] simplifies building user interfaces for Web applications. It wires client-generated events to server-side event handlers. Tapestry [21] is a framework for creating Web applications in Java, where a Web application is composed of a combination of a specification file in XML, an XHTML template and a Java class. The template defines the XHTML document that includes dynamic contents, and the page components written in Java define the representation of the dynamic contents.

These Web application development frameworks make it easier to develop Web applications because they standardize various processes such as the receipt of requests, the validation of form data, and Web page transfers, and these processes become independent from the others. The BioPro system does not use these frameworks. However, it is based on the MVC architecture, where data access layer (model), presentation layer (view), and business-logic layer (controller) can be independent from the other.

8.3. Visual Development Environments for Client Programs. In the development of client programs, a variety of graphical components have been used to create their graphical user interfaces. This has made the software development easier [11]. There are several researches on software development that makes use of graphics, which include rapid development of visual applications [22], the visualization of software requirements using multimedia [23–25], assistance for object-oriented programming using UML [26, 27], the development of language processors using the graphical representation of their behaviors [28, 29], automatic form generation by the combination of graphical components [30, 31], and visual software development environments [32, 33]. To assist in database accesses using graphics, visual retrieval of structured Web information [34], and the visualization of the contents of a database [35] have been researched.

In the BioPro system, we can design database tables in the same way as visual programming tools, and the fields of the tables can be pasted in appropriate places of Web pages by mouse dragging.

8.4. Visual-Oriented IDE's. As for the development of server-side programs using graphics, there are some Web design tools such as IronSpeed [36], Sun Java Studio Creator [10], Visual Studio.NET [37], Web Sphere Studio [38], and Dreamweaver [39]. With these tools, users can design the contents of Web pages using a variety of graphical components. By connecting the pages to databases, they can create the dynamic contents of the pages as well. These tools make it easy to design Web pages, and program code is generated from those designed Web pages.

However, to develop a complete Web application, users need to write code to define business processes and add it to the generated code. On the other hand, the system this paper presents provides some high-level functional Web components such as frames and framesets [40], pivot tables [41], and customizable visual Web components. The pivot tables sum up various kinds of data in two dimensions as in Microsoft Excel. The differences between Microsoft Excel and this system is as follows:

(1) this system generates some Web components such as radio buttons and checkboxes in the data cells of a pivot table to send a request to the server;

(2) this system generates some variables that record the values indicating which radio button or checkboxes are checked to automatically receive those values on the sever side.

8.5. Model-Based Approaches. The Object-Oriented Hypermedia Design Method (OOHDM) [42–46] is a model-based approach for building hypermedia applications. It comprises four different activities: conceptual design, navigational design, abstract interface design, and implementation. It models a Web application so that the navigation model can be separated from the conceptual model. UWAT+ [47, 48] makes it possible to design Web application transactions according to the user's perspective and to integrate the Web transaction design with the information and navigation design of the Web application. Web Modeling Language (WebML) [49] is a visual notation for specifying complex Web sites at the conceptual level. WebML enables the high-level description of a Web site under distinct orthogonal dimensions: its data content (structural model), the pages that compose it (composition model), the topology of links between pages (navigation model), the layout and graphic requirements for page rendering (presentation model), and the customization features for one-to-one content delivery (personalization model). Comprehension of Web applications is a complex task, since several concerns coexist in their implementation, among which the business logic, the navigation structure (as supported by hyperlinks and form submission), and persistent data storage. Conallen's stereotypes [50] are a set of UML stereotypes designed with Web applications in mind. They add information on such things as navigation structure, page generation, and form submission that UML diagrams do not normally contain explicitly. OPM/Web [51] introduces hierarchical state expressing and suppressing to model both structure and dynamics of Web applications. WAST [52] specifies a navigational structure of Web applications and detects the inconsistency of parameter names between JSP pages and actions during the test execution.

Model-based approaches mainly support the conceptual design of Web applications. The BioPro system can assist in their implementation based on the conceptual design. In addition, the inconsistency problem of parameter names as described above will hardly occur. When users need to write code, the BioPro system shows these parameter names as automatically generated variables in Web source windows. Even if they use a wrong parameter name, this error can be detected during the compilation time because that wrong variable is not declared.

9. Conclusion

This paper has presented a method that makes it possible to visually design and program Web applications that use frame facilities and pivot tables. Image-oriented design using such graphical Web components and action writing for each source page of a target page, which is based on automatic interpretation of submitted data, have important roles to develop Web applications. They have made the development of Web applications easy, especially in the design of the presentation layer and action writing of the business-logic layer.

Existing tools such as Homepage Builder [4], and Dreamweaver & Fireworks [39] provide a variety of GUI components and have sufficient facilities for editing Web pages. On the other hand, the proposed system has facilitated the development of Web applications by providing Web components such as *Frameset pages, Web pages*, pivot tables, *DB tables, Program tables*, and *Web source windows*, where each Web component can easily refer to the definitions of the other components. As the next step, we are going to investigate how to visually incorporate rich components such as Flex, Flash, JavaScript, and Applets in the design phase. In the future, we intend to develop an end-user programming environment based on the BioPro system, where typical business patterns will be shown by using a sequence of functional components, and users will be guided and taught what to do next to develop a Web application they have in mind.

References

[1] S. Pemberton, D. Austin, J. Axelsson, et al., "XHTML™ 1.0 the extensible hypertext markup language (Second Edition)," 2002, http://www.w3.org/TR/xhtml1.

[2] P. van Schaik and J. Ling, "The effects of frame layout and differential background contrast on visual search performance in Web pages," *Interacting with Computers*, vol. 13, no. 5, pp. 513–525, 2001.

[3] T. Comber and J. Maltby, "Layout complexity: does it measure usability?" in *Proceedings of the International Conference on Human-Computer Interaction (INTERACT '97)*, pp. 623–626, Sydney, Australia, July 1997.

[4] IBM, "WebSphere Studio Homepage Builder," 2007, http://www-306.ibm.com/software/awdtools/hpbuilder.

[5] T. Shimomura, "Visual design and programming for Web applications," *Journal of Visual Languages & Computing*, vol. 16, no. 3, pp. 213–230, 2005.

[6] A. Leff and J. Rayfield, "Web-application development using the model/view/controller design pattern," in *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing (EDOC '01)*, pp. 118–127, Seattle, Wash, USA, September 2001.

[7] J. Goodwill, *Mastering Jakarta Struts*, John Wiley & Sons, New York, NY, USA, 2002.

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

[9] NetBeans, 2008, http://www.netbeans.org.

[10] Sun Microsystems, Inc., Sun Java Studio Creator, 2004, http://wwws.sun.com/software/products/jscreator.

[11] R. Mogha and R. Bhargava, *Sun One Studio Programming*, John Wiley & Sons, New York, NY, USA, 2002.

[12] JetBrains: IntelliJ IDEA, 2007, http://www.jetbrains.com/idea.

[13] "CodeWarrior Development Tools," 2008, http://www.freescale.com/codewarrior.

[14] S. Shavor, J. D'Anjou, S. Fairbrother, D. Kehn, J. Kellerman, and P. McCarthy, *The Java™ Developer's Guide to Eclipse*, Addison-Wesley, Reading, Mass, USA, 2003.

[15] A. Latteier and M. Pelletier, *The Zope Book*, Macmillan Computer, New York, NY, USA, 2001.

[16] M. Burnett, S. K. Chekka, and R. Pandey, "FAR: an end-user language to support cottage e-services," in *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*, pp. 195–202, Stresa, Italy, September 2001.

[17] J. Lin, M. Thomsen, and J. A. Landay, "A visual language for sketching large and complex interactive designs," *CHI Letters*, vol. 4, no. 1, pp. 307–314, 2002.

[18] A. S. Christensen, A. Møller, and M. I. Schwartzbach, "Extending Java for high-level Web service construction," *ACM Transactions on Programming Languages and Systems*, vol. 25, no. 6, pp. 814–875, 2003.

[19] N. Al-Darwish, "PageGen: an effective scheme for dynamic generation of Web pages," *Information and Software Technology*, vol. 45, no. 10, pp. 651–662, 2003.

[20] Sun Microsystems, Inc., JavaServer Faces, 2003, http://java.sun.com/j2ee/javaserverfaces.

[21] Apache Software Foundation: Tapestry, 2003, http://jakarta.apache.org/tapestry.

[22] G. D. Penna, B. Intrigila, and S. Orefice, "An environment for the design and implementation of visual applications," *Journal of Visual Languages & Computing*, vol. 15, no. 6, pp. 439–461, 2004.

[23] D. C. Kung, "An executable visual formalism for object-oriented conceptual modeling," *Journal of Systems and Software*, vol. 31, no. 1, pp. 33–43, 1995.

[24] D.-J. Chen, W.-C. Chen, and K. M. Kavi, "Visual requirement representation," *Journal of Systems and Software*, vol. 61, no. 2, pp. 129–143, 2002.

[25] R. Castelló, R. Mili, and I. G. Tollis, "ViSta: a tool suite for the visualization of behavioral requirements," *Journal of Systems and Software*, vol. 62, no. 3, pp. 141–159, 2002.

[26] S. J. Mellor and M. J. Balcer, *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley, Reading, Mass, USA, 2002.

[27] C. Nentwich, W. Emmerich, A. Finkelstein, and A. Zisman, "BOX: browsing objects in XML," *Software: Practice and Experience*, vol. 30, no. 15, pp. 1661–1676, 2000.

[28] K. Zhang, D.-Q. Zhang, and J. Cao, "Design, construction, and application of a generic visual language generation environment," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 289–307, 2001.

[29] S. Glass, D. Ince, and E. Fergus, "Llun—a high-level debugger for generated parsers," *Software: Practice and Experience*, vol. 31, no. 10, pp. 983–1001, 2001.

[30] S. Stoecklin and C. Allen, "Creating a reusable GUI component," *Software: Practice and Experience*, vol. 32, no. 5, pp. 403–416, 2002.

[31] S. A. Mamrak and S. Pole, "Automatic form generation," *Software: Practice and Experience*, vol. 32, no. 11, pp. 1051–1063, 2002.

[32] K. L. Mills and H. Gomaa, "A knowledge-based method for inferring semantic concepts from visual models of system behavior," *ACM Transactions on Software Engineering and Methodology*, vol. 9, no. 3, pp. 306–337, 2000.

[33] A. F. Blackwell, "See what you need: helping end-users to build abstractions," *Journal of Visual Languages & Computing*, vol. 12, no. 5, pp. 475–499, 2001.

[34] W.-S. Li, J. Shim, and K. S. Candan, "WebDB: a system for querying semi-structured data on the Web," *Journal of Visual Languages & Computing*, vol. 13, no. 1, pp. 3–33, 2002.

[35] I. F. Cruz and P. S. Leveille, "As you like it: personalized database visualization using a visual language," *Journal of Visual Languages & Computing*, vol. 12, no. 5, pp. 525–549, 2001.

[36] Iron Speed, Inc., "Iron Speed Designer," 2004, http://www.ironspeed.com.

[37] D. D. Loveh, D. Maharry, B. Sempf, and D. Xie, *Effective Visual Studio .Net*, Springer, New York, NY, USA, 2002.

[38] I. Redbooks, *Ejb 2.0 Development with Websphere Studio Application Developer*, Vervante, Rolling Hls Ests, Calif, USA, 2003.

[39] Adobe Systems Incorporated, "Adobe Dreamweaver and Fireworks," 2007, http://www.adobe.com/products/dreamweaver.

[40] T. Shimomura, K. Ikeda, Q. L. Chen, N. S. Lang, and M. Takahashi, "Visual programming of hierarchical frames for Web applications," in *Proceedings of the International Conference on Computer Engineering and Applications (CEA '07)*, pp. 384–389, Gold Coast, Australia, January 2007.

[41] T. Shimomura, K. Ikeda, Q. L. Chen, N. S. Lang, and M. Takahashi, "Visual pivot-table components for Web application development," in *Proceedings of the 3rd IASTED International Conference on Advances in Computer Science and Technology (ACST '07)*, pp. 90–95, Phuket, Thailand, April 2007.

[42] G. Rossi and D. Schwabe, "Object-oriented design structures in Web application models," *Annals of Software Engineering*, vol. 13, no. 1–4, pp. 97–110, 2002.

[43] D. Schwabe, L. Esmeraldo, G. Rossi, and F. Lyardet, "Engineering Web applications for reuse," *IEEE Multimedia*, vol. 8, no. 1, pp. 20–31, 2001.

[44] D. Schwabe and G. Rossi, "From domain models to hypermedia applications: an object-oriented approach," in *Proceedings of the International Workshop on Methodologies for Designing and Developing Hypermedia Applications*, Edinburgh, UK, September 1994.

[45] D. Schwabe and G. Rossi, "Building hypermedia applications as navigational views of information models," in *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS '95)*, p. 231, Maui, Hawaii, USA, January 1995.

[46] D. Schwabe and G. Rossi, "The object oriented hypermedia design model," *Communications of the ACM*, vol. 38, no. 8, pp. 45–46, 1995.

[47] D. Distante, G. Rossi, G. Canfora, and S. Tilley, "A comprehensive design model for integrating business processes in Web applications," *International Journal of Web Engineering and Technology*, vol. 3, no. 1, pp. 43–72, 2007.

[48] D. Distante, G. Canfora, S. Tilley, and S. Huang, "Redesigning legacy applications for the Web with UWAT+: a case study," in *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*, pp. 482–491, Shanghai, China, May 2006.

[49] S. Ceri, P. Fraternali, and A. Bongio, "Web modeling language (WebML): a modeling language for designing Web sites," *Computer Networks*, vol. 33, no. 1–6, pp. 137–157, 2000.

[50] F. Ricca, M. D. Penta, M. Torchiano, P. Tonella, and M. Ceccato, "An empirical study on the usefulness of Conallen's stereotypes inWeb application comprehension," in *Proceedings of the 8th IEEE International Symposium on Web Site Evolution (WSE '06)*, pp. 58–68, Philadelphia, Pa, USA, September 2006.

[51] I. Reinhartz-Berger, D. Dori, and S. Katz, "OPM/Web—object-process methodology for developing Web applications," *Annals of Software Engineering*, vol. 13, no. 1–4, pp. 141–161, 2002.

[52] H. Tai, T. Nerome, M. Abe, and M. Hori, "A model-driven development support environment for Web applications," *Transactions of Information Processing Society of Japan*, vol. 44, no. 6, pp. 1498–1508, 2003.