Web Application Framework for Toolkit-based GUI Programming

Masanobu Umeda Osamu Takata Isao Nagasawa

Department of Creation Informatics
Graduate School of Computer Science and Systems Engineering
Kyushu Institute of Technology

This paper proposes a new framework for building interactive applications using the WWW. This framework introduces the Web Application Shell (WASH) and the GUI library for web applications. The WASH is a web application container which, instead of UIMS, provides several basic services to web applications. The GUI library plays the same role as that in stand-alone applications, and enables toolkit-based GUI programming of web applications on the server side. The WASH and the GUI library encapsulate complex page descriptions in HTML and scripting languages, the representation of a client state using cookies and the URL, and the management of state integrity between a client and a server. Unlike other approaches such as JSP and WebMacro, application programmers need not know how and what page descriptions are generated, and how session and client state are managed. This framework makes it possible to use abstraction and modularization techniques in the GUI programming of web applications in the same way as that of traditional stand-alone applications. This paper also describes the design and implementation of the WASH servlet and Servlet Window Toolkit (SWT) in Java. SWT is practically compatible with Abstract Window Toolkit (AWT) and Swing. Through their use, Java applications using AWT and Swing can be easily transformed into web applications with little modification.

1 Introduction

Applications based on World Wide Web (WWW) technology have the advantage of centralizing business resources such as hardware and software. However, such web applications, unlike traditional stand-alone applications, involve serious difficulty in their development and maintenance.

The HyperText Markup Language (HTML) [1], which is used for describing presentation data in the WWW, is capable of describing static web pages, but is not capable of describing pages which are dynamically changed according to the context of their interaction with a user. Therefore, page descriptions are usually generated dynamically on the server side using ordinary programming languages, such as Perl, PHP, and Java. Such a server-side program, however, often becomes complicated, and will often become problematic in its development and maintenance. This is because (1) a server-side program is often cluttered up with descriptions in HTML and some client-side scripting languages, (2) HTML descriptions embedded in a program lack the supports of authoring tools, such as a syntax checker, and (3) it is the responsibility of a server-side program to manage the identity and state of clients using cookies or URL [2] rewriting due to the stateless feature of the Hyper Text Transfer Protocol (HTTP) [3].

JavaServer Pages (JSP) [4] and Active Server Pages (ASP) define a simple rule set in order to describe both programs in Java or VisualBasic and HTML descriptions in a single source file. These approaches can improve the descriptiveness of one content independent from an other, but are not effective for mixed contents. Tag libraries and macro language approaches, such as JSTL [5] and WebMacro [6], have also been proposed. However, they are not essential solutions for resolving the mixture of languages having different syntax and semantics. Supports of the identity management and state management of web clients are beyond their scope.

This paper proposes a new framework for building interactive applications using the WWW. This framework introduces the Web Application Shell (WASH) and the GUI library for web applications. The WASH is a web application container which provides several basic services to web applications in place of the user interface management system (UIMS). The GUI library plays the same role as it does in traditional stand-alone applications, and enables the toolkit-based GUI programming of web applications. This framework makes it possible to use abstraction and modularization techniques in the GUI programming of web applications in the same way as that of traditional stand-alone applications.

This paper also describes the design and imple-

mentation of the WASH servlet and Servlet Window Toolkit (SWT) in Java. SWT is practically compatible with Abstract Window Toolkit (AWT) and Swing. By using them, Java applications using AWT and Swing can be easily transformed into web applications with little modification in terms of GUI.

In the following sections, a new model of web applications and the details of the WASH are described. In section 4, the design and implementation of the WASH servlet and SWT are described. In section 5, experiments regarding practical web application programming are described. Related works are discussed in section 6.

2 A New Model of Web Applications

Based on the multitier model [7], Figure 1 illustrates the proposed architecture of web applications. The WASH, WASH proxy, and GUI library, which are hatched in the figure, are newly introduced elements in this framework.

The Web Application Shell (WASH) is located between an application gateway, such as CGI and Java Servlet, and a web application. The WASH plays a central role in this framework, that is, the life cycle management of an application, event management, and the state management of a web client. The WASH proxy performs some parts of the state management and event management in a web client on behalf of the WASH. These are functionally the same as fundamental services provided to a standalone GUI application by UIMS. These functions are implemented mainly as a part of an individual web application in a traditional multitier model, and are not well distinguished from application-specific ones. The proposed framework clearly distinguishes the functions related to UIMS from others, and in contrast defines them as functions which should be provided by a container of a web application. This functional separation advances the abstraction of the execution environment of a web application in a way different from Java Servlet, JSP, and ASP.

On the other hand, the GUI library provides presentation and operation using GUI and application programming interface to a web application through abstract GUI components. The GUI library encapsulates HTML descriptions of GUI, which are scattered in an application program, into GUI components. This encapsulation separates GUI descriptions and business logic in an application program, and GUI programming based on more abstract concepts becomes available.

3 The Web Application Shell

3.1 An Overview of the Web Application Shell

Figure 2 illustrates an overview of the WASH and its relationship to the WASH proxy and the GUI library. The WASH is composed of four subsystems: a system controller, application manager, event manager, and display manager. Communication between a user and a web application starts from an invocation event which is sent as a HTTP request from a web client to the system controller. If the system controller receives an invocation event, it requests the application manager to load and invoke an appropriate application. An application, which is invoked by the application manager, instantiates the GUI components of the GUI library for the interaction with a user in the same way as does traditional standalone applications. The system controller then requests presentation data regarding GUI components through the display manager, and returns the result to a web client. The returned result includes the definition of the WASH proxy written in JavaScript. Subsequent communication between a web client and the system controller is performed via the WASH proxy.

A user action on a web client, such as a button click by a mouse, is initially intercepted by the WASH proxy, and is then re-composed as a new URL representing an action event. The WASH proxy then sends it as a HTTP request to the system controller via a HTTP server. An action event is sent to the event manager via the system controller, and is dispatched to an appropriate GUI component as a GUI event. A GUI component, which receives a GUI event, may change its internal state, and if necessary notify an application of the event's reception. The display manager requests new presentation data regarding GUI components, and returns the data to a web client.

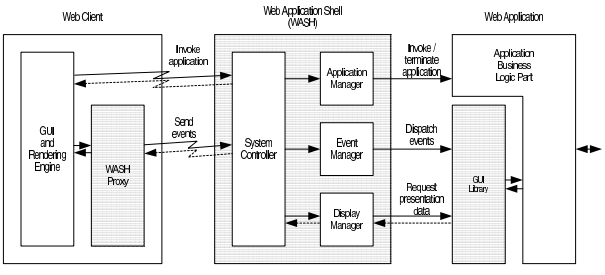

Figure 2: Overview of the WASH

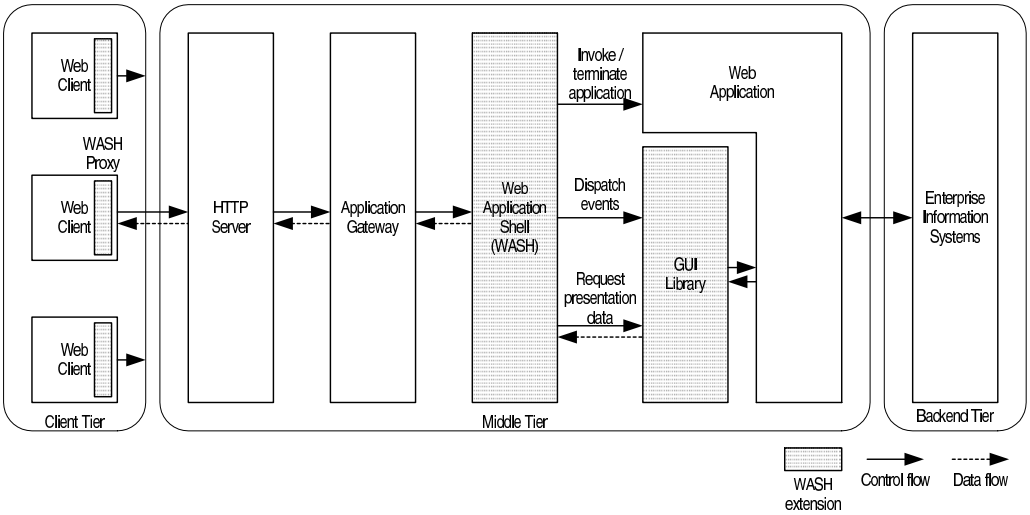

Figure 1: Architecture of web applications using the WASH

3.2 Events Handling

3.2.1 Representations of Events

A link represented by an anchor tag in a HTML description is a connection from one web resource to another. If a mouse or keyboard action activates a link, an associated resource is retrieved based on a protocol and a location specified by a URL. The WASH uses this simple communication mechanism as the basis for notification of GUI events from a web client to the WASH. Consequently, interactions that require quick feedback, such as drawing a rubber band, cannot be supported directly since the HTTP is inefficient for such purposes. This constraint over the protocol does not, however, deny interactive functions, such as a tool tip or a balloon help, using Dynamic HTML (DHTML) [8].

The WASH represents a GUI event raised on a web client by a URL which takes the following form:

`http://`*hostname*`/`*pathname*`/wash?`*parameters*

`wash` is the name of a WASH implementation. In the case of Java Servlet implementation, it is the name of a Java class. Therefore, the parameters part specifies all meanings of an event. WASH events are classified into three types: invocation event, action event, and pseudo event.

3.2.2 The Invocation Event

The invocation event invokes a web application on the server side. It is directly raised by a link activation on a web client. The parameters of the invocation event take the following form:

`application =` *ApplicationName*

ApplicationName specifies the name of an application to be invoked.

The reception of an invocation event by the system controller starts a new session, and prompts the controller to ask the application manager to invoke a named application. An invoked application is assigned an identifier, *ApplicationID*, to distinguish it from other applications running for the same web client at the same time.

3.2.3 The Action Event

The action event notifies a server-side application of a user action taken on a web client. It is directly raised by a link activation on a web client. Parameters of the action event take the following form:

`appID =` *ApplicationID* `&`
`targetID =` *ComponentID* `&`
`eventName =` *EventName* `&`
`eventData =` *EventData* `&`
StateValues

ApplicationID is an assigned identifier of an application. *ComponentID* specifies a GUI component to which an event is sent. *EventName* specifies the name of an event, and *EventData* is a parameter specific to an event. *StateValues* is described in section 3.4.

When the system controller receives an action event, it initially processes *StateValues*, and then sends the event to the event manager. The event manager dispatches a received event to a corresponding GUI component in an implementation-specific form.

3.2.4 The Pseudo Event

The pseudo event is indirectly raised by a response to an action event unlike other types of events. Parameters of the pseudo event take the following form:

appID = *ApplicationID* &
targetID = *ComponentID* &
eventName = *EventName*

The redraw event, which forces the display update of a web client, is defined as a pseudo event. Its details are described in the following section 3.3.

3.3 Display Update Management

A web page can be divided recursively into several panes called frames using `FRAMESET` and `FRAME` tags as shown in Figure 3. When a link in a frame is activated, either its own frame, its parent frame, or a whole page containing it can be updated. Generally, an action event raised in one frame may affect the presentation of other frames as well as this frame. In such a case, a whole page may have to be updated to reflect all changes. Updating a whole page may, however, cause unnecessary window flushing and lose the positions of frame separators. Therefore, it is important to minimize display update by updating frames selectively.

The redraw event, a kind of pseudo event, is defined in order to update a specific frame without updating a whole page. Redraw events are raised as soon as a HTML description of a frame, in which a link was activated, is loaded into a web client. Generation of this pseudo event is realized by the WASH proxy which is activated by a JavaScript program embedded in a HTML description. Figure 4 shows an example of a HTML description which raises a redraw event when the description is loaded, where `UpdateWindow(WIN, URL)` is a WASH proxy function to replace the contents of a window or a frame named `WIN` with a given `URL`. This HTML description is loaded when a radio button is clicked in the right-hand side of Figure 3. The button click causes resource changes in a component on the left-hand side. These changes are reflected to a web client by a redraw event for a frame on the left-hand side.

Unlike action events, redraw events are sent only to the display manager in order to request new presentation data regarding GUI components.

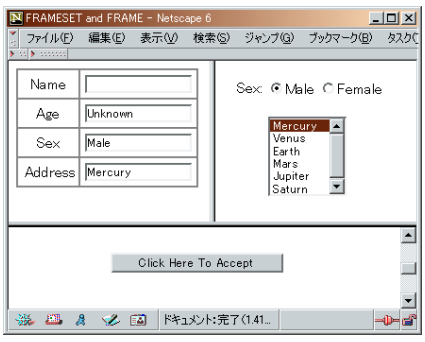

Figure 3: A web page split into panes using frames

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 ...">
<html>
<head>
 <script language="JavaScript" src="/wash.js"></script>
 <script language="JavaScript">
   function Initialize(){
     UpdateWindow('box1', '/servlet/Wash?appID=test1&
targetID=box1&eventName=redraw');
   }
   function Uninitialize(){}
 </script>
</head>
<body onload="Initialize();" onunload="Unini...>
 <form>
   <table border="0" cellpadding="10" ...>
   ...
   </table>
 </form>
</body>
</html>
```

Figure 4: A HTML description raising a redraw event when loaded

3.4 Window States Management

A page description can have special elements called controls, such as the `TEXTAREA` element and the `INPUT` element, in a HTML form. Each control has a value that can be modified by a user action. When a form containing controls is submitted to a HTTP server, current values of the controls are sent together with a URL of the form. Therefore, even if values of multiple controls in one form are locally changed before submission, a HTTP server can notice all of their changes. If controls are distributed in different frames of a web page such as that shown in Figure 3, this form behavior, however, does not work since a form is not allowed to contain frames. When a link is activated in one frame, a HTTP server is never notified of value changes of the controls in other frames. In addition, it is generally not a simple task to keep track of control values before and after a submission [4].

The WASH proxy maintains the control values in a page by collecting all values and sending them to a HTTP server together with an activated URL. *StateValues* in section 3.2 represents the current values of all controls in a web page. *StateValues* takes the following form:

ComponentID1 = *Value1* &
ComponentID2 = *Value2* &
...
ComponentIDN = *ValueN*

StateValues in a URL is extracted by the system controller, and it is transformed into events for reflection. The events are then sent to the event manager to dispatch them to corresponding GUI components.

3.5 Communication Optimization

All action events do not always imply an update of GUI presentation of a web application. If an action event affects only the internal states of some GUI components but not their presentations, there is no need to transmit the presentation data of GUI components to a web client and redraw a client screen. However, if no presentation data is sent back to a web client due to a lack of presentation changes, it is impossible for a web client to construct the appropriate presentation. This is because available JavaScript implementations do not allow conditional update of a web page when a HTTP request is generated by substituting a value of the `href` property of the `location` object as follows:

window.`location.href` = *NewURL*

The WASH optimizes communication between a web client and the server by using a small Java applet class named `Connector`, shown in Figure 5. The `Connector` applet retrieves a content for a given URL by communicating with a HTTP server directly. The `SendEvent` function of the WASH proxy accesses an instance of the `Connector` class embedded in a web page [9] in order to obtain presentation data for a given URL instead of operating a `location` object. The display manager on the server side returns empty data if and only if there is no change in the presentation data of all GUI components. The WASH proxy updates a web page by operating a `document` object only when the returned data is not empty. Figure 6 shows a simple implementation of the `SendEvent` function of the WASH proxy.

```
public class Connector extends Applet {
  public String getContent(String event)
  {
    URL url = new URL(event);
    InputStream is = url.openStream();
    BufferedReader reader =
      new BufferedReader(new InputStreamReader(is));
    try {
      StringBuffer s = new StringBuffer();
      String l = reader.readLine();
      while(l != null){
        s.append(l).append("\n");
        l = reader.readLine();
      };
      return s.toString();
    } finally {
      reader.close();
    }
  }
}
```

Figure 5: Connector applet

```
function SendEvent(url){
  // Collect state values of controls.
  var states = CollectStates(window.top);
  // Build a complete URL for Connector.
  url = window.location.protocol +
        "//" + window.location.hostname +
        ((window.location.port == "")?"":
         (":" + window.location.port)) +
        url + (states==""?"":"&"+states);
  // Retrieve contents of URL using Connector.
  var newtext =
    window.document.connector.getContent(url);
  // Update a document if non-empty.
  if(newtext != null && newtext != ""){
    document.open("TEXT/HTML");
    document.write(newtext);
    document.close();
  };
}
```

Figure 6: SendEvent function of the WASH proxy

4 Java Implementation of the WASH

The WASH has two implementations. One is an implementation in a multi-threaded Prolog environment [10], and the other is that in Java Servlet. This section describes the design and implementation of the WASH servlet and the GUI library in Java.

4.1 WASH servlet

The WASH servlet is an implementation of the WASH in Java Servlet. Figure 7 illustrates a system configuration of a web application using the WASH servlet. The application library stores classes of web applications, and the GUI library stores classes of GUI components for web applications.

The `Wash` class is an implementation of the system controller, that extends the `HttpServlet` class and implements the `SingleThreadModel` interface. The `doGet` method of this class is a main function for processing HTTP requests. The `doGet` method analyzes a HTTP request, and calls for the appropriate methods of the `ApplicationManager`, `EventManager`, and `DisplayManager` classes as described in section 3. For example, an `ApplicationManager` loads a requested application class from the application library, and invokes it by calling the `public static void main(String[])` method of the application. A stand-alone application of Java is invoked in the same way.

4.2 Servlet Window Toolkit

The design and implementation of the WASH servlet are straightforward, but those of the GUI library are not. Java has Abstract Window Toolkit (AWT) and Swing [11] as the standard GUI programming library for applets and stand-alone applications. Many development tools are also based on these libraries. If

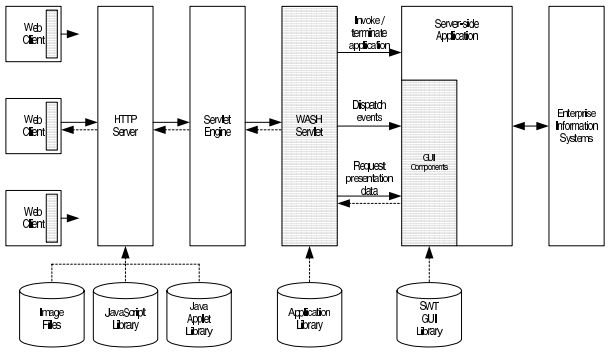

Figure 7: System configuration of a web application using the WASH servlet

the libraries and tools can be used for the development of web applications, their programming and debugging become much easier than those of traditional servlet applications. Consequently, AWT and Swing are chosen as a model of the GUI library instead of a newly designed one, and the application programming interface is carefully designed so that stand-alone applications are easily transformed into web applications.

A GUI library for the WASH servlet is named Servlet Window Toolkit (SWT) and placed under the package `swt`. The structure of SWT is almost the same as that of AWT and Swing so that stand-alone applications are transformed into web applications by renaming imported packages, `java.awt` and `javax.swing`, at minimum. The design policy of the class structure of SWT can be summarized as follows:

1. Component classes of AWT and Swing are redefined using the same structure in the package `swt`.

2. If possible, non-component classes of AWT and Swing, such as the `AWTEvent` class, are extended using the same name in the package `swt`. Otherwise, they are redefined using the same structure in the package `swt`.

3. If possible, interfaces of AWT and Swing are used as is. Otherwise, they are redefined using the same structure in the package `swt`.

The `LayoutManager` interface is an example of the redefined interface. This is because its methods take an instance of the `Component` and the `Container` classes of AWT. These methods have to be rewritten so as to take an instance of the corresponding classes of SWT.

Needless to say, it is difficult to imitate all parts of AWT and Swing using HTML and the HTTP. The following sections describe events handling, presentation and layout, and rendering operations in SWT.

4.3 Events Handling in SWT

Events handled in the WASH are limited to a mouse or keyboard action which activates a link. Therefore, an action event of the WASH can be mapped naturally into semantic events which indicate that a component-defined action has occurred. The `ActionEvent`, `ItemEvent`, and `TextEvent` classes are such examples. However, low-level events defined as subclasses of the `ComponentEvent` class are difficult to map completely. For example, it is difficult to generate symmetrically the `FocusEvent` which indicates that a component has gained or lost a keyboard focus. In this implementation, the semantic events of AWT and Swing are only realized. This restriction on event handling in SWT means that a stand-alone application, which uses low-level event classes, does not work without many changes. However, most stand-alone applications, which use only semantic event classes, work without any changes in regard to event handling. Figure 8 shows methods of the `Button` class of SWT for handling an `ActionEvent`. These are the same as those of AWT.

```
protected void processEvent(AWTEvent e){
  if(e instanceof ActionEvent){
    processActionEvent((ActionEvent)e);
    return;
  }
  super.processEvent(e);
}

protected void processActionEvent(ActionEvent e){
  if(actionListener != null){
    actionListener.actionPerformed(e);
  }
}
```

Figure 8: Event handling methods of SWT Button class

Figure 9 shows how the `ActionEvent` class of SWT is used in a Java program. When a `Button` is clicked, an `actionPerformed` method of the `ActionListener` interface is called for with an instance of the `ActionEvent` class as described in Figure 8. This fragment of a Java program works for both SWT and AWT without any changes, even though their look and feel are different.

```
Button myButton = new Button("Click me once");
myButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
      (Button)(e.getSource()).setLabel("Thanks!");
    }
  }
);
```

Figure 9: An example of Java program handling a semantic event

4.4 Presentation and Layout of Components

HTML is capable of representing rich graphical contents using tags and style sheets. Presentation data of SWT classes have to be represented by these tags and style sheets.

Most component classes can be represented by using one or more tags and optional style sheets. For example, an instance of the `Button` class can be represented as follows:

```
<INPUT TYPE="button"
 NAME="button1" VALUE="A label"
 ONCLICK="SendEvent('/servlet/Wash?...>
```

On the other hand, layout managers which implement the `LayoutManager` interface have some difficulty in reproducing a look and feel similar to that when using HTML. This is because directives regarding layout constraints are limited in HTML. In this implementation, the `FlowLayout`, `BorderLayout`, and `GridLayout` classes are realized using the `TABLE` tag, even though its look and feel is slightly different. This may produce strange results when automatic positioning by layout managers is combined with absolute positioning.

4.5 Graphics Context and Rendering Operation

The `Graphics` class of AWT represents a graphics context which encapsulates the state information needed for rendering image data to various devices and off-screen images. The `paint` method of a component paints an image of the component using a `Graphics` object. On the other hand, components of SWT are never rendered to display devices directly since a web client renders to these devices instead. The original use of the `Graphics` class is meaningless in SWT. Therefore, a `Graphics` object is instead associated with a network stream bound for a HTTP request so that the same programming style can be realized in SWT. Figure 10 shows the `paint` method of the `Button` class of SWT. A `Button` renders its image by printing a HTML description to an associated stream.

```
public void paint(Graphics g){
  g.println("<INPUT TYPE=\"button\" " +
            "NAME=\"" + getComponentID() + "\" " +
            "VALUE=\"" + getLabel() + "\" " +
            "ONCLICK=\"SendEvent('" +
            g.getBaseURL() +  // http://xxx/wash?...
            "&targetID=" + getComponentID() +
            "&eventName=ActionEvent" +
            "&eventData=" +   // No event data
            "');\">" );
}
```

Figure 10: The `paint` method of SWT Button class

5 Application Examples

In this section, experiments regarding practical web application programming using the WASH servlet and SWT are described. Two types of applications are used in the experiments; a simple calculator which is developed only for the experiments and an accounting application which was developed in an independent research project.

The calculator is initially created as a stand-alone application using AWT and Swing. This program consists of approximately 100 lines and uses the `JButton` class for representing the number and operator keys, the `TextField` class for representing the display, the `JPanel` and `JFrame` classes as their containers, and the `BorderLayout` and `GridLayout` classes for layout. Figure 11 is a snapshot of the stand-alone calculator application. This calculator application is then transformed into a web application by substituting only the packages `java.awt`, `java.awt.event`, and `javax.swing` for the package `swt`. Figure 12 is a snapshot of the calculator using the WASH and SWT. It functions in the same manner as the stand-alone calculator application, even though their appearances are slightly different.

The accounting application is designed to edit vouchers of payment and sales using GUI, and to store them in an accounting database constructed using a commercial database software. It consists of three functional modules: those for GUI, vouchers management, and data management. It is originally designed for the use of a single user, and multiple access is out of consideration. Therefore, the vouchers management and data management modules may have to be redesigned for practical use over the WWW, but this problem is beyond the scope of this paper. In this experiment, the GUI module is only adapted to the WWW using SWT. Figure 13 is a snapshot of the stand-alone accounting application. This accounting application is transformed similarly into a web application. Figure 14 is a snapshot of the accounting application using the WASH and SWT. It also works functionally in the same manner as the stand-alone accounting application, even though their appearances are considerably different. The reason why their appearances are different is that the use of absolute positioning is combined with the use of layout manager classes.

The results of these experiments show that (1) the transformation of programs using AWT and Swing into programs using SWT can be achieved by only substituting a small number of lines regarding imported packages, (2) the transformed programs work functionally in the same manner as the original programs, and (3) the appearances of the transformed programs are almost similar if layout manager classes are only used instead of absolute positioning.

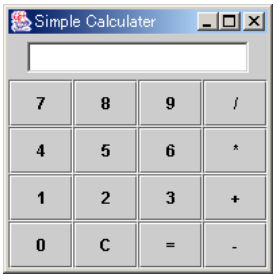

Figure 11: Stand-alone calculator application

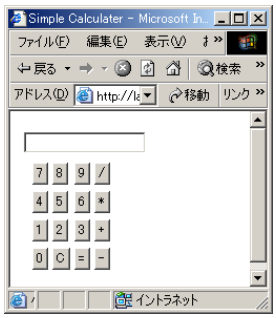

Figure 12: Calculator using the WASH and SWT

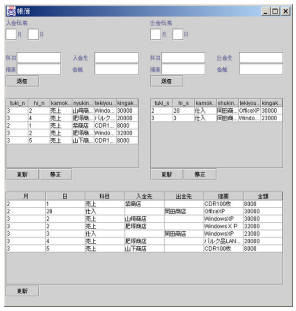

Figure 13: Stand-alone accounting application

Figure 14: Accounting application using the WASH and SWT

6 Discussion

Java applets and .NET applications using C# can provide full-featured GUI functions over the WWW using traditional GUI programming style and a single programming language. However, they still do not become major. This is because applet involves an overhead cost which cannot be disregarded at a program loading time, and full-featured GUI functions are rarely needed in practical web applications. Consequently, the importance of both HTML-based web applications and a framework for supporting their development does not change.

JSP and ASP are techniques to separate a server-side program from HTML descriptions. For example, the JSP container generates a Java Servlet program automatically from a JSP description. This approach reduces a mixture of different languages, but a JSP description still contains fragments of a Java program. JSTL and WebMacro are intended to reduce a mixture of HTML and Java further by encapsulating either of them. These approaches have much effect on further reduction, but instead a new ad-hock macro language has to be introduced. JavaServer Faces [12] well separates descriptions in HTML and Java; however, both descriptions are tightly coupled with each other and they need to be developed in tandem. In either case, the identity management and state management of web clients are beyond their scope. In contrast to these approaches, the WASH and the GUI library encapsulate HTML descriptions and the identity and state management completely from application programmers, and enable web application development using traditional GUI programming style and a single language.

HTML has the capability of describing web pages, but descriptions of other GUI entities, such as menus and tool bars, are beyond its scope. It is impossible for a web application to have its own menus and tool bars as does a stand-alone application. XML-based User Interface Language (XUL) [13] is a language for describing a user interface without being hardwired into an application. XUL allows the development of a web application which looks like a stand-alone application in terms of GUI's use of a web client. The native use of XUL is, however, as difficult as that of DHTML. The WASH and the GUI library can encapsulate the complexity of XUL, and enable application programmers to utilize the distinctive features of XUL and DHTML without knowing any of their details. As a result, an interactive web application which looks and acts like a usual stand-alone application can be developed more easily and efficiently than ever.

This paper describes an implementation in Java. The proposed framework can be applied to other programming languages and GUI libraries. In fact, we have implemented similar functions in a multi-threaded Prolog environment [10].

7 Conclusion

This paper proposes a new framework for building interactive applications using the WWW. This framework introduces the Web Application Shell (WASH) and the GUI library for web applications. The WASH is a web application container which, instead of UIMS, provides several basic services to web applications. The GUI library enables toolkit-based GUI programming of web applications in the manner of traditional stand-alone applications. The WASH and the GUI library encapsulate page descriptions in HTML and scripting languages, the representation of a client state using cookies and the URL, and the management of state integrity between a client and a server. Unlike other approaches such as JSP, JSTL, and WebMacro, application programmers need not know how and what page descriptions are generated, and how session and client state are managed. In this manner, abstraction and modularization can be achieved in the GUI programming of web applications in the same way as that of traditional stand-alone applications.

This paper also describes the design and implementation of the WASH servlet and Servlet Window Toolkit (SWT) in Java. SWT is designed so that it is practically compatible with AWT and Swing. In many cases, Java applications using AWT and Swing can be easily transformed into web applications by

renaming only imported packages in terms of GUI.

This framework will improve the productivity of web application development, and also the quality and maintainability of web applications. It is especially significant in that the same programming model of the GUI and development tools as those for traditional stand-alone applications can be applied to web application development.

The current implementation of SWT does not support menus and tool bars. As XUL is capable of defining these components, it is valuable to incorporate it into the library. Limitations of GUI events and layout management are subjects of future research.

References

[1] Dave Raggett, Arnaud Le Hors, and Ian Jacobs, "HTML 4.01 specification," *W3C Recommendation*, December 1999.

[2] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," *RFC2396*, August 1998.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, L. Masinter, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," *RFC2616*, June 1999.

[4] Ben Forta, Scott Stirling, Edwin Smith, Larry Kim, Roger Kerr, David Aden, and Andre Lei, *Java Server Pages Application Development*, Sams Publishing, 2000.

[5] Sun Microsystems Inc., "JavaServer Pages Standard Tag Library," http://java.sun.com/ products/jsp/jstl/index.html, 2002.

[6] Semiotek Inc., "WebMacro: Web Macro," http://www.webmacro.org, 2002.

[7] Nicholas Kassem and Enterprise Team, *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*, Addison Wesley, 2000.

[8] Netscape Communications Corporation, "DevEdge Online - Dynamic HTML developer central," http://developer.netscape.com/ tech/dynhtml/index.html, 1999.

[9] Marty Hall and Larry Brown, *Core web programming*, Prentice Hall, second edition, 2001.

[10] Keiichi Katamine, Masanobu Umeda, Isao Nagasawa, and Masaaki Hashimoto, "Integrated development environment for knowledge-based systems and its applications," in *Proceedings of the ACIS Second International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing*, August 2001, pp. 739–745.

[11] Patrick Chan and Rosanna Lee, *The Java Class Libraries*, vol. 2, Addison Wesley, second edition, 1997.

[12] Sun Microsystems Inc., "JavaServer Faces Technology," http://java.sun.com/j2ee/ javaserverfaces, 2002.

[13] XPToolkit Project, "XPToolkit: Cross-platform UI toolkit," http://www.mozilla.org/ xpfe, 2002.