ENHANCEMENT OF WEB BROWSER PROGRAMMING WITH GUEST
Visual Programinng Tool for Non-programmers

Yuka Obu, Kazuhiro Maruo, Tatsuhiro Yonekura
Graduate School of Science and Engeneering, Ibaraki University, Ibaraki, Japan

Masaru Kamada, Shusuke Okamoto
Graduate School of Science and Engeneering, Ibaraki University, Ibaraki, Japan

Department of Computer and Information Sceince, Seikei University, Tokyo, Japan

Keywords: Visual programming, state-transition diagram, Web programming for non-programmers, GUEST (Graphical User interface Editor by State-transition Diagram), New user interface design concept.

Abstract: Many people have recently become interested in Web 2.0, which is a new Web service concept. Web sites have become sources of information and functionality that enables users to create new content of their own. Using this capability, users have been customizing Web pages as they like. Users are now looking for more versatile browsers that will enable them to edit and display content based on their own creative concepts and preferences. Motivated by this demand, we have been working on a project to develop a state-transition diagram-based Web browser programming scheme that supports participatory Web use and enables the end-user to interact with Web content. We implemented a prototype of our scheme called GUEST. Using GUEST, users can define behaviors of a Web browser easily even if they have no programming experience. However, there are parts of the scheme that are not easy for beginners to use. That is, there are complicated user interfaces that prevent the user from easily gaining an intuitive understanding of how to use GUEST. Therefore, in this paper, we focus on users' difficulties in using the interface, and introduce a new concept of the design.

1 INTRODUCTION

A great deal of interest has been generated over the last few years in the World Wide Web-based services referred to as Web 2.0. Web 2.0 is a new Web services concept that includes a range of Web-related technologies and Web sites and services. One salient feature of Web 2.0 is the concept of the Web as a subjective participation platform. In marked contrast to the earlier Web service, where information was unilaterally delivered to users from isolated sites, Web 2.0 sites are sources of information and functionality that enable users to create new content of their own (O'Reilly, 2005).

New demands have been placed on browsers by widespread use of this capability. They must now display the information as it is provided from Web sites and enable users to customize it as they like. Users are now looking for more versatile browsers that will let them edit and display content based on their own creative concepts and preferences. Motivated by this demand, we have been working on a project to develop a state-transition diagram-based Web browser programming scheme that supports participatory Web functions and enables close interaction between the end-user and Web content. Our aim is to customize the content of the Web sites on the user side. Customizing means freely defining the action of the browser so users can browse a site as they like. For example, the user can browse link content using mini windows, or, if there are links the user does not want to display, the user can make them disappear. The most important point of our scheme is actualization of these functions on the user side of the Web browser.

To implement this concept, we have developed GUEST (Graphical User interface Editor by State diagram) (Obu, Yamamoto, Yonekura, Kamada, Okamoto 2007). GUEST is a Firefox extension that enables users to define the behavior of the browser using a state-transition diagram. Our aim is for users of GUEST to be able to define functions easily even if they have no programming experience. However, there have been some aspects of GUEST that are not easy for beginners to use. That is, there are some complicated user interfaces that prevent the user from gaining an intuitive understanding of how to use GUEST. Therefore we modified and improved some concepts of the design. Here, we introduce a new version of GUEST.

2 VISUAL PROGRAMMING

Our objective is to develop Web browser programming capabilities that will enable ordinary non-technically oriented end-users to freely customize the behaviour of the browsers so they can edit and display Web content according to their own preferences.

2.1 State-transition Diagram Model

To enable users to easily customize the behaviour of browsers, we used the state-transition diagram model. This has been used in object oriented software design and in development as a method of representing the relations between objects.

Basically, state-transition diagrams consist of two kinds of symbols, circles representing states and arrows representing transitions. This model can be applied to many kinds of representations by relating an object to a state and changes under certain conditions to transitions. It is easy to represent something using circles and arrows. Ordinary end-users can comprehend state-transition diagrams more easily than programming languages. Our approach has the advantage of being intuitively easy to understand and easy for end-users to learn (Okamoto, Kamada, Nakao, 2005) (Okamoto, Shimomura, Kamada, Yonekura, 2006).

Researchers have long been engaged in study of state-transition diagram interfaces (Jacob, 1983) (Jacob, 1986) (Harel, 1987), and UML (Object Management Group, 2007) is regarded as one type of state-transition diagram scheme. It is widely used in object oriented software design and development. In the UML-based approach, a line called the link represents the relationship between the objects. The state-transition diagram represents the behavior of each object (Kendall, 2003). UML has been developed for software design. In the UML, there are many kinds of diagrams to represent a system, and the state-transition diagram is one of them. The state-transition diagram of the UML is used to generalize and show the method to the event and the state of the object to each event.

Meanwhile, some kinds of state-transition diagram are developed in our project. These diagrams are not for showing generalized image of the relationship between objects. They are adapted for each particular purpose and show the action of the software. For example, there are the state-transition diagrams for GUI, and interactive animation, and so on. The definition of the state and the transition are different and they are adapted for each purpose. By creating state-transition diagram specialized for each purpose, the best expression for each purpose can be achieved. It can be easy to convert into the source code, and it leads the early understanding of the source code.

2.2 Visual Programming with a State-transition Diagram

Ordinary language-based programming requires a degree of expertise in a programming language and other specialized knowledge, but the vast majority of people using the Web today do not have these special programming skills. In other words, the programming methods that are available as tools today are poorly suited to enabling ordinary end-users to customize Web browsers that are a kind of application software.

This led us to pursue an alternative visual programming approach that is not based on language. Rather than words and language, visual programming uses a system of diagrams. Compared to language-based programming, this visual approach is far easier to learn (the grammatical rules are more intuitive), involves much less abstract thinking, and requires relatively little use of the keyboard, so it is well suited to the needs of ordinary end-users.

We have exploited these advantages to define a visual language model based on state-transition diagrams. The model defines the action of the content widget, which is a set of widgets for representing Web page content – that is, HTML tags and a style sheet. A range of HTML elements are included, such as those for representing links and image tags. Using our model both clarifies exactly what is to be customized and helps the end-user understand what is going on. Combining the actions of the widgets in various ways enables a wide range of customization options.

In our programming scheme, the state of each item in the widget corresponds to the state of the state-transition diagram, and the action of the user corresponds to the transition. By using this model, a user can define the behaviour of the Web browser.

3 RELATED RESEARCHES

Islay, an interactive animation-authoring tool based on the state-transition diagram, was first proposed in 2005 (Okamoto, Kamada, Nakao, 2005) (Okamoto, Shimomura, Kamada, Yonekura, 2006). Islay uses the modern paradigm of object oriented modelling and the classical state-transition diagram to make authoring interactive animation intuitively comprehensible. By using the authoring tool, non-programmers can define the animation of the characters. As a side effect, the user may learn how to define dynamic objects while having fun with animation.

One of Mozilla Firefox's (Mozilla org., 2007) extensions, Greasemonkey (Greasemonkey, 2007), is a tool that enables users to gain valuable experiences on the Web. The user writes Greasemonkey scripts (called user JavaScripts) to change Web pages. User JavaScripts are executed every time the Web page is loaded and can utilize a full range of functions provided by Greasemonkey in the form of a library. This permits a wide range of customization options: changing the layout of a Web page, adding functions to a Web page, appropriating information from another Web site, and so on. Today, there are a great many user JavaScripts that have been made available on many Web sites. While these user JavaScripts support advanced customization, making them requires a high level of programming skill and a specialized knowledge of the Web that are far beyond the abilities of most ordinary end-users. Platypus (Platypus, 2007) is another Mozilla Firefox extension that enables users to modify a Web page from a browser called Platypus GUI and save the changes as a Greasemonkey user JavaScript so that they will be repeated the next time the user visits the page. GUI enables ordinary end-users to easily customize Web pages. The main drawback of Platypus GUI is that it was designed to customize Web page layouts, so it cannot be used to customize the behavior of Web pages.

Chickenfoot (Bolin, Webber, Rha, Wilson, Miller, 2005) is another extension of this kind. Chickenfoot consists of a library that adds new commands for web automation to the browser's built-in JavaScript language, and a development environment that allows Chickenfoot programs to be entered and tested inside the Web browser. Chickenfoot has many commands, including pattern matching, form manipulation, page navigation, and page modification. The user inputs these commands and can also use the same variables available to JavaScript to define the behavior of the Web browser. The commands make defining the Web automation easier than the original JavaScript, but the user needs to have knowledge of the commands and JavaScript variables. Therefore, ordinary non-technically oriented end-users may not be able to intuitively comprehend how to use this extension.

Client-side tools and APIs have been developed by the SIMILE project (SIMILE, 2007). Appalachian (Appalachian, 2007) and Piggy Bank (Huynh, Mazzocchi, Karger, 2005) are two such Firefox extensions. Appalachian adds the ability to manage and use several OpenIDs to ease the login parts of a user's browsing experience. Piggy Bank turns a Web browser into a mash up platform, by enabling a user to extract data from different web sites and mix it together. It also allows the user to store extracted information locally so it can be searched later and to exchange the collected information with others on demand. Timeplot (Timeplot, 2007), TimeLine (TimeLine, 2007), and Exhibit (Huynh, Karger, Miller, 2007) are APIs that enable users to create rich content on a Web page. By using these APIs, the user can create interactive widgets on a Web page. The user does not have to know database or complicated web application technologies.

In addition to these programs, we have developed GUEST (Graphical User interface Editor by State diagram). GUEST is a Firefox extension that enables users to define the behavior of the browser using a state-transition diagram. Using GUEST, the user can define behaviors easily even if he or she has no programming experience.

4 GUEST

4.1 Overview

To solve some of the problems that related programs suffer from, we have developed a prototype Web visual programming system called GUEST

(Graphical User interface Editor by State diagram), which is used as a Mozilla Firefox browser extension (Mozilla org.).

Mozilla Firefox is a Web browser developed by the Mozilla Foundation that supports plug-in and add-on program packages called extensions. Firefox extensions mainly consist of JavaScript and parts written in XUL (Mozilla Foundation, 2007), an XML language (algorithms are written in JavaScript and GUIs are written in XUL). Because the majority of Firefox operations can be controlled from extensions, the process of designing and developing software has been minimized, creating an original Web browser.

Figure 1 shows a schematic overview of how GUEST works. An overview is also presented in list form below.

(1) The user first describes the desired Web browser behavior by editing on the state-transition diagram editor.

(2) The GUEST system automatically converts the state-transition diagram to JavaScript.

(3) The newly defined behavior is executed to the Web content by the Mozilla Firefox Web browser. The defined behaviors are executed every time the page is loaded.

(4) The state-transition diagram can be saved and loaded, and the state-transition diagram can be rewritten. The rewritten state-transition diagram is converted to JavaScript, and the user can choose which JavaScript should be executed.

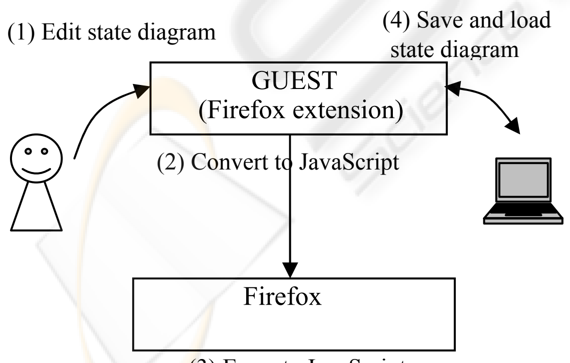

Figure 1: Schematic overview of how GUEST works.

4.2 Functions

GUEST consists of three basic elements: a state-transition diagram editor that generates state-transition diagrams, a conversion function that converts state-transition diagrams to JavaScript, and a file function that saves and reloads the diagrams.

4.2.1 State-transition Diagram Editor

Figure 2 shows a typical screenshot of the state-transition diagram editor: widgets (tags) that construct the Web page are listed in the left side bar, the upper right area is the edit window, and the lower right is a browser window.

A screenshot focused on the state-transition diagram in figure 2 is shown in figure 3. States are represented by the little squares, and transitions are represented by arrows. The little circle in the upper left corner of the screen in figure 3 signifies the initial state of the state-transition diagram. The labels in the state boxes are the state names, and the labels shown alongside transition arrows represent transition conditions. In this figure, the state name is the anchor tag, and the condition of the transition means mouse-over. States include the definition of the action of the label. In figure 3, the state on the left has no action, and the state on the right has an action that hides the tag. Consequently, the state-transition diagram in figure 3 means that, if the mouse is over the anchor tag, the tag is hidden.

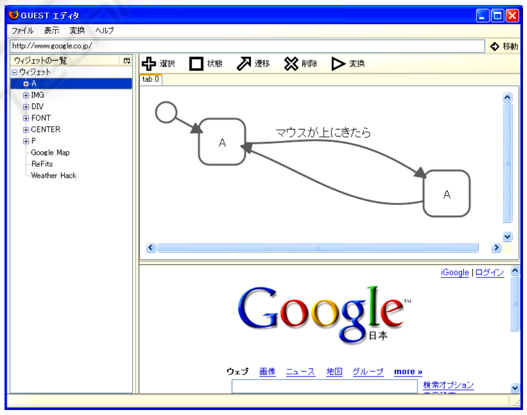

Figure 2: Screenshot of state-transition diagram editor.

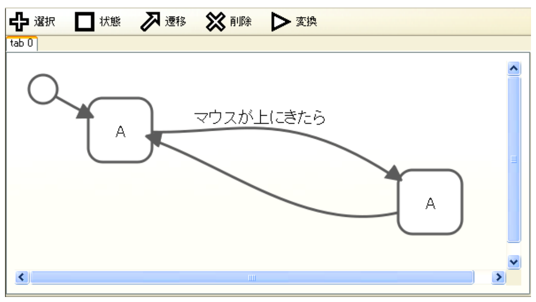

Figure 3: Screenshot focused on state-transition diagram shown in figure 2.

Tables 1 and 2 show the list of actions and conditions that can be used in the state-transition diagram editor. By using these actions and conditions in the state-transition diagram, the user can define the behavior as in the following example.

If the user mouses over the anchor tag, the tag is hidden.

If the user double clicks the anchor tag, the link opens on a new window.

If the mouse is over the images, a message box pops up.

If the mouse is over the anchor tags, mini windows open, displaying the content at the link.

Table 1: Actions used in states.

Actions
No action
Show tag
Hide tag
Show an alert message box
Send message to another edit tab
Load content of link
Open content of link on new browser's tab
Open content of link in mini window
Save image

Table 2: Transition conditions.

Conditions
No condition
User clicks on widget
User right clicks on widget
User center clicks on widget
User left clicks on widget
User double clicks on widget
Mouse is over widget
Mouse is moved to side of widget
After several times
After receiving message from another tab
After successful action

4.2.2 JavaScript Conversion Function

In the conversion from the state-transition diagram to JavaScript program, states are converted to functions (one to one), and the transitions are converted to function pointers. This simplifies the automatic conversion of state-transition diagrams to JavaScript.

Part of the script that is converted from the state-transition diagram shown in figure 3 is shown below.

```
function gst_dia0(){
  gst_initTag("A");
  gst_startDia("gst_dia0",gst_st0_0);
}
```

```
function gst_st0_0(act){
  if(act){
    return;
  }
   if(gst_tagObjectList["A"].mouseove
   r) {
     gst_statePointer["gst_dia0"]
                             = gst_st0_1;
     return;
  }
}

function gst_st0_1(act) {
   if(act){
     gst_tagObjectList["A"].hideTag(
     true);
     gst_tagObjectList["A"].setSucce
     ss();
     return;
    }
    {
     gst_statePointer["gst_dia0"]
                             = gst_st0_0;
     return;
    }
 }
function gst_startup() {
   gst_dia0();
   gst_startGuestScheduler();
   return;
 }
function gst_checkevent() {
   var checklist=new Object();
   checklist['mouseover']='check';
   return checkList;
}
```

GUEST loads and executes the converted JavaScript. GUEST has original libraries of functions to execute actions and conditions. However, the user does not need to consider the JavaScript or the function that controls the behavior of the Web browser. The only thing the user has to do is edit the state-transition diagram.

4.2.3 State-transition Diagram Save and Load Functions

GUEST uses Resource Description Framework (RDF: W3 Consortium, 2007) as the format for state-transition diagram permanent files. RDF is an XML language that describes the metadata of the resources. Therefore, we can describe information such as headlines of articles and other kinds of metadata. RDF has the advantages of reducing the implementation burden and providing a generic permanent file format.

The data stored in the RDF files includes:

(1) **State-related data**: State attributes (types of tags, actions, etc), coordinates of state image in diagrams, names, and so on.

(2) **Transition-related data**: Transition attribute (types of events, etc.), coordinates of transition images in diagrams, control points (Bezier curves), names, and so on.

(3) **State diagram-related data**: List of state transitions included in state-transition diagrams, URLs of pages for which diagrams have been generated, names, and so on.

By using these RDF files, the user can re-construct the behavior of the Web browser.

5 ENHANCEMENT OF USER INTERFACE

5.1 Some GUEST Issues

Because GUEST users do not need to consider the programming language, GUEST's functions may, to some degree, be user-friendly. However, some aspects of the program are not easy for real beginners to use.

(1) **The Meaning of the State**: In a state-transition diagram edited as in figure 3, a state has two meanings. The state name (label of the box) indicates the widget whose action is defined, and the action of the widget is defined simultaneously in a state. However, the user cannot intuitively recognize that the action of the widget is defined. For example even if the action is to hide the tag or to show an alert message box, there are no differences in the appearances. Therefore, we needed to reconsider the meaning of the state.

(2) **The Meaning of the Initial Transition**: Figure 4 shows the initial state of the state-transition editor. As can be seen, there is only a circle indicating an initial transition. The user has to connect the circle with the first state using an arrow. We gave some non-programming users a trial use of GUEST as an experiment. Our observation of the trial showed that users had difficulty understanding the initial transition.

(3) **The Relation between Tag and Web Page**: In the state-transition diagram editor, there are lists of the widgets (tags) and the Web browser area. Because the position of the widget is not indicated in the Web browser area, however, the user cannot comprehend the relation between a widget and an appearance on the Web. We had to solve that problem.

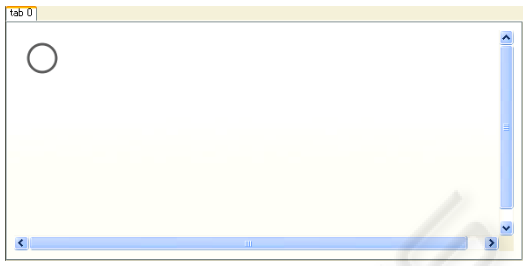

Figure 4: Screenshot of initial state of editor.

(4) **The Function of the Tab**: The state-transition diagram editor has the function of a tab. The user can edit on some tabs, and then the results are converted into one JavaScript file. Some messages can be sent from one tab to another tab, but this function is rarely used because it is a bit hard to understand, and the concept of the function is not clear.

Considering these issues, we changed some of the user interface design.

5.2 Modification of User Interface

Until now, an object and an action at the same time have always been defined in one state. In our new design concept, we considered an action as a state, and arranged the design as follows:

- A state is redefined as an action of the widget.
- The object whose action is defined by a state-transition diagram is a widget (a tag on a Web page). The action of one widget is defined on one tab. This enables control of the relation between a widget and a state-transition diagram.
- The meaning of the transition condition stays unchanged in the current version of GUEST.
- An initial state that indicates a default status is used instead of a circle and an arrow indicating an initial transition.
- Tab names are changed from tab0 (tab1, tab2. and so on) to the name of the widget (Image, link, etc.).

Figure 5 shows an image of these changes. As can be seen, since each tab handles a widget in a Web page, the relationship between the state-transition diagrams and the object is comprehensible. Moreover, because that relationship will be clearer than in the current version of GUEST, the user is more easily able to intuitively understand it.

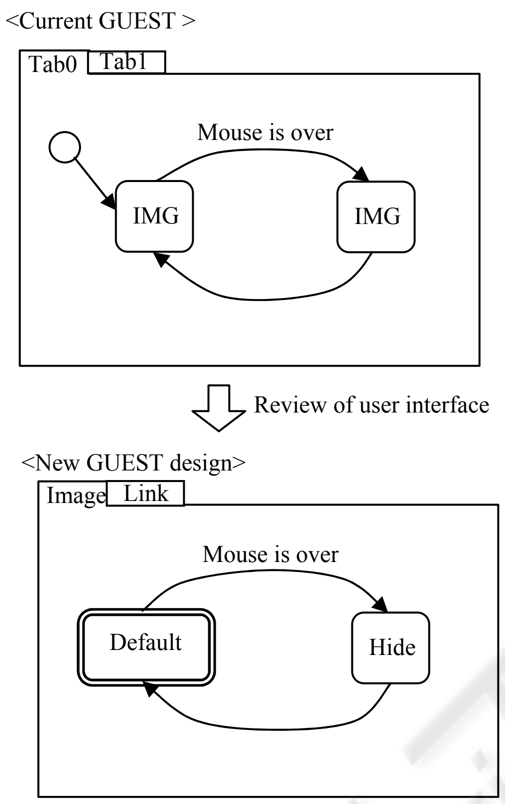

Figure 5: Image of new user interface.

5.3 Message Passing between Tabs

GUEST has a function that sends messages between tabs. The new user interface design shown in figure 5 makes it more useful in passing messages. Using tabs to handle each widget, makes message passing between objects more understandable.

Figure 6 shows how messages are passed between tabs. As can be seen, when a button is clicked, data is retrieved from the other Web page by Ajax communication.

By passing messages between tabs, the user can define more interactive behavior.

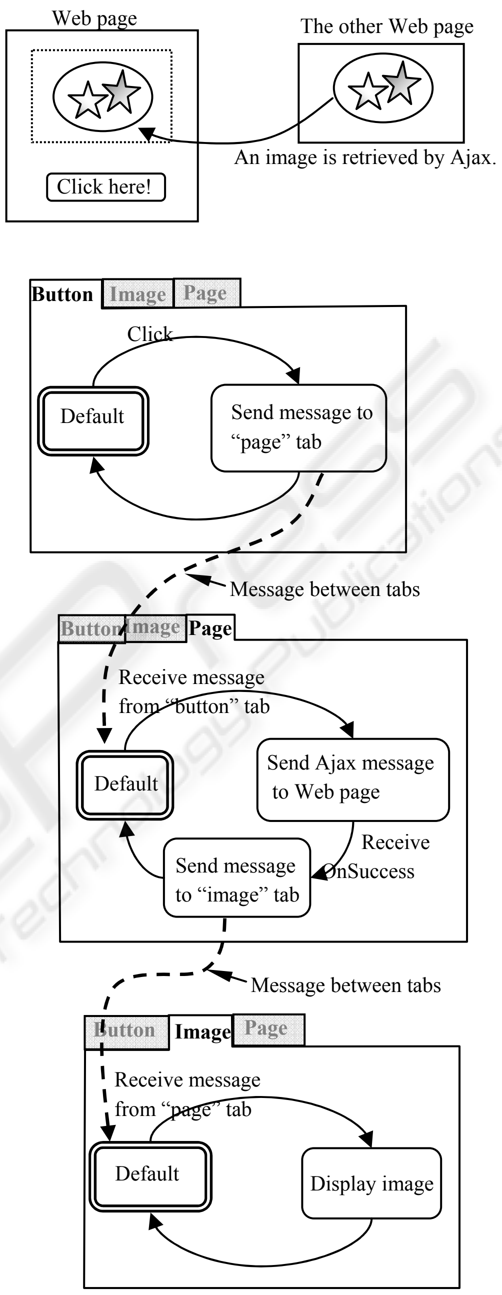

Figure 6: Example of message passing between tabs.

6 CONCLUSIONS

We discussed visual Web browser programming using a state-transition diagram model and presented a prototype of our scheme, which we call GUEST (Graphical User interface Editor by State diagram). We also discussed difficulties users have with the GUEST user interface. We then presented the new user interface design concept and the effective use of the message passing between tabs. These functions help users define and intuitively understand the behavior of the Web browser. Using GUEST with our new design concept, more users, even those with no programming experience, should be able to define the behavior of Web browsers more effectively.

In the future, we plan to implement our new design concept and evaluate the new version of GUEST, including the range of support it offers in handling Web services. We are also planning to experiment with usability testing by end-users. The latest version of GUEST will be available on http://yonex1.cis.ibaraki.ac.jp/~yuka/index.html.

REFERENCES

Tim O'Reilly, 2005. What Is Web 2.0. In *http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web20.html*

Obu, Y., Yamamoto, M., Yonekura, T., Kamada, M., Okamoto, S., 2007. Exploring State-transition diagram-based Web Browser Programming. In *CYBERWORLDS 2007, International Conference on Cyberworlds*. Cyberwrolds Press.

Okamoto, S., Kamada, M., Nakao, T., 2005. Proposal of an Interactive Animation Authoring Tool Based on State-transition diagram. *Vo.46, No.SIG 1(PRO24), pp19-27*. Information Processing Society of Japan. in Japanese.

Okamoto, S., Shimomura, T., Kamada, M., Yonekura, T., May 2006. Programming with Islay, an Interactive Animation Authoring Tool. *Vo.47, No.SIG 6(PRO29)*. Information Processing Society of Japan. in Japanese.

Object Management Group, 2007. OMG'S Unified Modeling Language. In *http://www.omg.org/*

Kendall, S., 2003. *UML Explained*, Pearson Education Japan. Tokyo, 4th edition.

Mozilla org., 2007. Mozilla Firefox. In *http://www.mozilla.or.jo/products/firefox/*

Greasemonkey, 2007. In *http://greasemonkey.mozdev.org/*

Platypus, 2007. In *http://platypus.mozdev.org/*

Bolin, M., Webber, M., Rha, P., Wilson, T., Miller, R.C., 2005. Automation and Customization of Rendered Web Pages. In *ACM Conference on User Interface Software and Technology, pp163-172*. UIST.

Mozilla Foundation, 2007. XUL. In *http://www.mozilla.or.jo/products/xul/*

World Wide Web Consortium, 2007. Resource Description Framework. In *http://www.w3.org/RDF/*

SIMILE Project, 2007. In *http://simile.mit.edu/*

Appalachian, 2007. In *http://simile.mit.edu/wiki/Appalachian/*

Huynh, D., Mazzocchi, S., Karger, D., 2005. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *International Semantic Web Conference*.

Timeplot, 2007. In *http://simile.mit.edu/Timeplot/*

TimeLine, 2007. In *http://simile.mit.edu/TimeLine/*

Huynh, D., Karger, D., Miller, R., 2007. Exhibit: Lightweight Structured Data Publishing. In *International World Wide Web Conference*. ACM 978-1-59593-654-7/07/0005.