Regular Paper

Creating Web-based Presentations by Demonstration

Yoshinori Aoki,† Fumio Ando† and Amane Nakajima†,☆

This paper describes mechanisms for recording and playing back Web browser operations. A recorder detects a user's operations on a Web browser and saves them as an event sequence called a scenario. A player plays back the scenario by controlling an actual Web browser. In addition, the recorder and player allow a user to add explanations to existing HTML contents by making "ink" annotations and attaching text, images, and hyperlinks to a Web page. The recorder and player make it easy to create a Web-based automatic presentation scenario, which can be played back later. The recorder and player run in a Java-enabled Web browser. Users do not have to prepare the Web contents to be recorded themselves; they can work with existing Web pages. This paper also describes example applications implemented on top of the recorder and player.

1. Introduction

Web browsers are now among the most popular user interfaces in network computing environments. They can be used not only to view information on the Internet, but also as client platforms for Web-based applications such as online malls and electronic banking services. We are developing a system with which we can easily create automatic Web-based presentations by demonstrations. Our system detects a user's operations on Web pages, and saves them in a text file. The system replays the recorded operations later, allowing users to create a Web-based presentation by recording their operations such as loading Web pages, scrolling, and pointing at a certain part of a Web page. With our system, users can provide an automatic navigated tour of their Web sites, or show examples of the operations needed to apply for services at their sites.

This paper presents mechanisms for recording and playing back Web browser operations. A recorder detects a user's operations such as URL transition, form input, and mouse pointer movement, and saves them as an event sequence called a scenario. A player plays back recorded operations by controlling an actual Web browser. Four major features of the recorder and player are described below.

Event recording based on the Document Object Model: A unique feature of our recording mechanism is that it is based on the Document Object Model (DOM) [23]. It is simple, but very powerful. With the mechanism, the recorder can not only detect a user's operation events, but can also obtain many kinds of data related to the Web document. For example, when a browser finishes loading a Web page, the recorder not only detects a load event, but also obtains the title of the page, the URL, the number of hyperlinks in the page, and so on. The URL of the page is needed when the player replays the load event. The title and the number of hyperlinks are not needed for playback, but are valuable for automatically creating metadata for the recorded presentation.

Working with existing HTML contents and a normal Web browser: Since the recorder and player can work with ordinary Web pages, they can handle existing HTML contents. A Web page designer need not consider any recording and playback mechanisms. Users do not need to install any special Web browsers on their machines, because the recorder and player can work with a normal Web browser, such as Netscape Communicator or Microsoft Internet Explorer, without any modifications.

Working in a Web Browser: The recorder and player can be downloaded and used in a Web browser, because they are developed in Java and JavaScript. Thus, users do not have to install any software except a Web browser. To download the recorder and player applets to a Web browser, all a user has to do is access a Web page. It is especially important for end users to be able to use the recorder easily, because many Internet users are end users.

Presentation capability: The recorder and player allow a user to add explanations to existing HTML contents by making "ink" anno-

† IBM Research, Tokyo Research Laboratory
☆ Presently with IBM Global Services - Japan

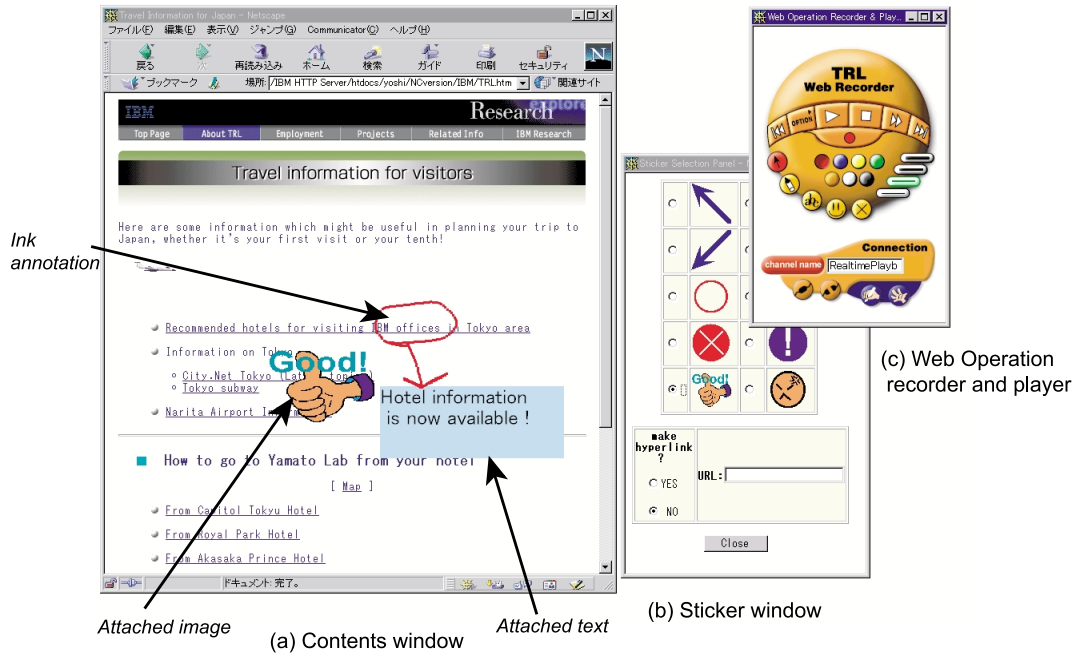


Fig. 1 Web operation recorder and player.

tations and attaching text, images, and hyperlinks to a Web page. These functions are useful for making an impressive Web-based presentation.

Figure 1 shows a screen image of our implementation. Window (c) shows the recorder and player working in a browser window. Window (a) is a Contents Window in which a user browses a Web page. Window (a) shows a Web page with ink annotations, text, and an image. Window (b) is a Sticker Window, from which a user can take images to attach in window (c).

The rest of the paper is organized as follows. The next section discusses our work and related work. In the section after that, we describe the recording and playback mechanisms. Then, in the following section, we describe some sample applications that we implemented. The last section presents our conclusions and outlines our plans for future work.

2. Related Work

Lotus ScreenCam and Microsoft Camcorder are commercial products that provide image-based screen recording and playback functions. They capture the full screen and save it as a movie, which is a sequence of captured images. The volume of image-based recorded data is much larger than that of event-based recorded data [15]: the data size of a one minute recording is normally several megabytes. However, for gathering many users' data on the Internet, a small data size is important. Another advantage of event-based recording systems is flexibility of playback. Through event-based playback, the player offers some effective ways of playing back scenarios. For example, this function can be used to play a recorded scenario at double speed or one-third speed. Through selective playback, a user can select the types of event to be played back; for example, it is possible to play back all types of events except mouse movement events. This function is very useful, because it allows a user to control the playback for a particular purpose.

Synchronized Multimedia Integration Language (SMIL 1.0) [3),8),18)] is a layout language for describing a multimedia presentation consisting of multiple elements of music, voice, images, text, video, and graphics in a common, synchronized timeline. WebStage [25)] creates an automatic Web presentation to facilitate passive Web browsing by adding images, sounds, and movies. In SMIL, it is possible to describe the layout and timing for each text, image, audio, video, or other object. However, SMIL authoring tools are as complicated as conventional multimedia authoring tools such as Macromedia Director. In such tools, we have to specify the behavior for each object in detail by us-

ing multiple time lines. In addition, it is not possible to describe a user's operations such as window sizing, form input, and mouse pointer movement.

Event-based recording has been studied in the form of programming by demonstration (PBD) systems [5),6),10),13),14),16),21),22)]. A PBD system records a user's operations and plays them back. One feature of our system is that the recording and playback mechanisms are separated from the Web browser, whereas many conventional PBD systems are specially designed for recording and playback. This allows our system to work with a normal Web browser without any modifications. Previous studies have given us ideas for applications that could run on top of our mechanisms. Two examples are Peridot [16)] and SimUI [10)], PBD systems that help users create GUIs (graphical user interfaces). Metamouse [13)] is a system for simple drawing applications that watches a user's operations and writes a program that generalizes those operations. It is similar to our recorder in that the user must explicitly indicate the start of a demonstration. Eager [5)] and DemoOffice [21)] detect a user's repetitive operations automatically, and generate a program to execute the repetitive operations. Internet Scrapbook [22)] is an application that provides a function for automating repetitive browsing tasks, using PBD techniques. An interesting function is that the system allows a user to create a personal page by clipping only the necessary portions from multiple Web pages. The personal page is automatically updated by the system.

Applications that have recording and playback functions can be developed by using special toolkits [4),14),15)]. LEDA [14)] is a programming environment on top of the window system of OS/2, and can be used to develop PBD applications. Applications developed with LEDA can record a user's operations on GUI components, such as button clicking and window resizing, but cannot detect application-specific events such as page loading event and form submission. HABANERO [4)] and Jedemo [15)] are toolkits for developing Java applications or applets that have recording and playback functions. JAMM [2)] is a Java runtime environment that supports the shared use of existing Java applets. These technologies are complementary to our mechanisms, because our recorder does not support recording and playback functions for Java applets in Web pages.

Several Web-browser-sharing technologies using operation event detection mechanisms have been developed in the last three years [7),9),11),19)] GroupWeb [7),17)] is a special Web browser for sharing Web pages with group members in real time. In CoWeb [9)], the server modifies HTML documents by replacing HTML input elements with Java applets that enable form input operations to be detected. But it is an inflexible method, because the event handlers of modified elements do not work correctly. Hooking of the message queue is used to detect a user's operations with a normal Web browser in [11),19)]. But such an approach requires hooking modules specific to each browser and each operating system, to be developed and installed in each client machine. It is not a general approach, and requires a lot of work by the user.

3. Recording and Playback Mechanisms

In this section, we describe our recording and playback mechanisms. The recorder records a user's operations on a Web browser. The recordable operations include URL transition, scrolling, window sizing, window positioning, form operation, and mouse pointer movement. Recorded operations are saved as a scenario, which is a sequence of operation events. A scenario contains URLs that a user has visited, and operations at each of the URLs. The player plays back a scenario by controlling the actual Web browser. The recorder allows users to make ink annotations and attach text, images, and hyperlinks on a Web page. Users can create scenarios manually, but do not need to do so when recording from real operations. Operations on normal Web pages can be recorded.

3.1 Recording Mechanism

The recorder has to detect a user's operation events in order to record them. Our approach is to modify HTML contents in the proxy server in order to insert event generation methods. The modified HTML file detects the user's operation events and notifies the recorder of them autonomously. **Figure 2** shows the architecture of our system. First, a user has to download the recorder page containing the recorder applet. The recorder applet opens a new browser window called Contents Window. The user browses and operates Web pages with the Contents Window as in normal Web browsing, downloading Web pages via a

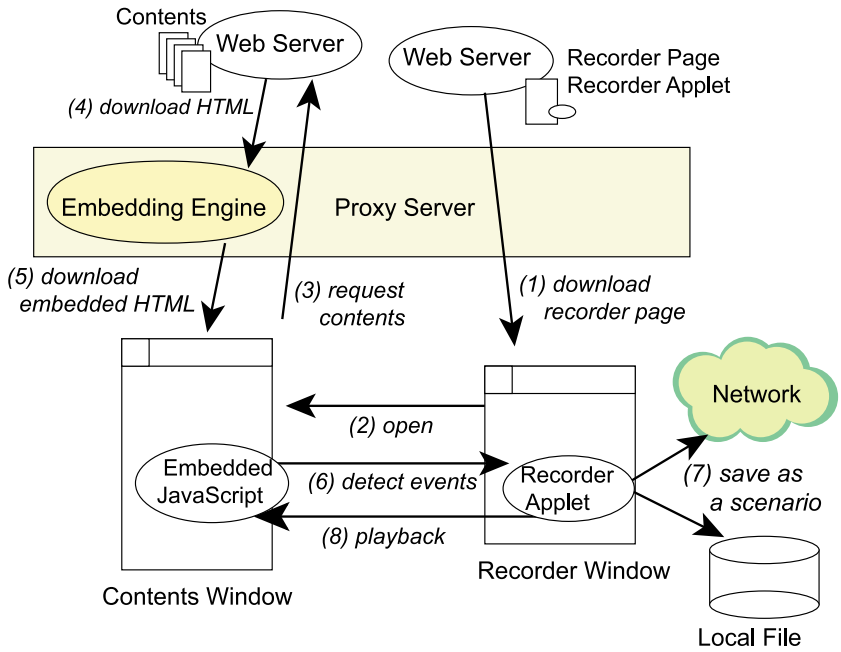


Fig. 2 System architecture.

proxy server that contains the Embedding Engine. The Embedding Engine inserts event handlers and JavaScript methods, which detect a user's operation, into the original HTML file on the fly.

3.1.1 Methods for Inserting Event Handlers

There are two alternative methods for setting event handlers in an HTML file, as shown in **Table 1**. One is to insert event handlers into each HTML tag, as in the following line. We call this the tag method.

<IMG SRC="XXX" onClick="notify()">

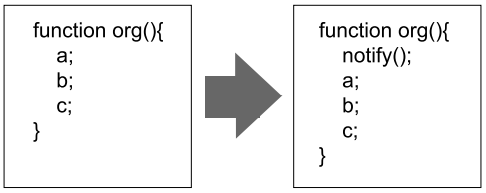
The other is to set event handlers in JavaScript, as in the following line. We call this the script method.

document.images[0].onClick=notify;

When the author of the HTML file has already set an event handler, both the original event handler and our event handler have to be executed. They do not conflict, because the function of our event handler is simply to notify the recorder applet that the event has fired. In the tag method, the Embedding Engine has to insert our event handler to execute both the original event handler, "org()", and our event handler, "notify()", as in the following line:

Table 1 Comparison of methods for inserting event handlers.

	Tag method	Script method
Conflict with original scripts?	No	No
Robustness	Weak	Strong
Implementation of the Embedding Engine	Complex	Simple



(a) Original Event Hanlder (b) Modified Event Handler

Fig. 3 Modification of event handler.

<IMG SRC="XXX" onClick="notify();org()">

In the script method, embedded JavaScript has to

(1) get the original event handler as a function object,
(2) convert the function object into a string object, as shown in **Fig. 3** (a),
(3) insert our event handler into on original event handler with string operations, as shown in Fig. 3 (b), and
(4) convert the string object into a function

object, and set it as an event handler.

Some HTML files on the Internet do not strictly conform to the HTML syntax. For example, some HTML files do not have a body tag, which all HTML files should contain. Some authors do not worry about the details of the HTML syntax, because it is possible to display loosely written HTML files on popular Web browsers. When an HTML file is loosely written, the tag method sometimes does not work correctly, because there are no tags to be inserted into our event handler, and thus some operation events cannot be recorded. The script method is robust against this problem, because all HTML objects are accessible in JavaScript if a Web browser has successfully parsed an HTML file, even if the HTML file is loosely written. In addition, implementation of the Embedding Engine for the script method is simpler than that for the tag method, because all the Embedding Engine has to do for the script method is insert a JavaScript file into the original HTML file. In Microsoft Internet Explorer 4 or 5, the script method is applicable to all HTML objects. In Netscape Communicator 4, however, the script method is applicable only to some HTML objects. Therefore the script method and tag method have to be used together for Netscape Communicator.

3.1.2 Example of Modified HTML File

Figure 4 (A) shows an example of an embedded HTML file for Netscape Communicator. In the example, a load event and an unload event are detected in the tag method, and a window-resizing event is detected in the script method.

The plus signs (+) indicate that the body tag has been modified by the Embedding Engine in the tag method. The method "org()" is the event handler set by the author of the HTML file, and the method "notify()" is the event handler inserted by the Embedding Engine. Both methods, "notify()" and "org()", are called when the Contents Window completes the loading of the HTML file and all its components such as image files and Java applets. Lines marked with '#' have been inserted by the Embedding Engine. If such lines are present, a JavaScript file named "EventDetector.js" will be included in the HTML file when it is browsed. Figure 4 (B) shows the content of the inserted JavaScript file. Lines marked with '-' show the JavaScript method that is called when a load or unload event occurs. The method "notify()" notifies the recorder of the load and unload events, and the recorder saves them in a scenario or sends them to other nodes. The recorder has to record other data required for playback: the time stamp, event type, and URL. It can also record more data based on the DOM [23], such as document title and the last date on which the document was modified, as shown in Fig. 4 (B). These data are valuable in analysis of users' behavior on Web pages. In the same way, the recorder can detect and save other events such as text field input and link clicking. The Embedding Engine inserts event handers into suitable tags to detect the user's operation events in the tag method. Lines marked with '=' contain JavaScript code that detects window-sizing events in the script method, and notifies the recorder of them. In the same way, window positioning and mouse

```
  <html>
  <head>
  <title>
  Sample page
  </title>
# <script language="JavaScript"
#    src="http://XXX/EventDetector.js>
# </script>
+ <body onLoad="notify('load',this);usermethod()"
+  onUnload="notify('unload',this)">

     original contets

  </body>
  </html>
```

(A) Example Embedded HTML

```
- function notify(type, obj){
-     var timeStamp = new Date();
-     var msg = timeStamp.getTime()+","+
-             type+","+document.location.href+","+
-             document.title+","+document.lastModified;
-     opener.record(msg);
- }

= window.onResize = notifyResize;
= function notifyResize(e){
=     var timeStamp = new Date();
=     var msg = timeStamp.getTime()+","+
=             e.type+","+e.width+","+e.height;
=     opener.record(msg);
= }
```

(B) Inserted JavaScript file "EventDetector.js"

Fig. 4 Example of a modified HTML file.

movement events can be recorded.

Since even JavaScript does not have event handers for scrolling in Netscape Communicator, the recorder periodically checks the page offset values in order to detect scrolling. JavaScript provides two properties, window.pageXOffset and window.pageYOffset, that show the page offsets on the Web browser. The recorder detects scrolling by monitoring changes in the property values.

Embedding rules are simple, but powerful enough to record all the necessary browser operations. The Embedding Engine has to insert lines marked '#' in Fig. 4 (A) and embed event handlers, "notify()," into the appropriate HTML tags.

3.2 Playback Mechanism

We implemented the Web operation player and Web operation recorder as a single Java applet. The recorder applet in Fig. 2 can be regarded as a player applet in this section.

3.2.1 Playback Using JavaScript Methods

The player parses a scenario and plays it back by using JavaScript methods. For example, a window-resizing event is played back by using the method "window.resizeTo(x,y)" provided in JavaScript. The entire recorded event can be played back with an appropriate JavaScript method. The JavaScript methods will be called from the player applet.

3.2.2 Technical Issues in URL Transition Playback

There are three major issues in URL transition playback: (1) timing of URL loading, (2) form submission handling, and (3) multiple frame support.

In our recording mechanism, all the URL transitions are recorded when all the components of the page have been loaded. The load event data include the URL. Therefore, the player can play back the URL transitions with the recorded load events. But the time stamp of the load event specifies the time at which the browser completed the loading of a Web page, not the time at which it started loading the page. The time stamp of the unload event shows the time at which loading of the next Web page started, because an unload event occurs when the previous Web page disappears from the browser. Therefore the player has to find the next URL and start loading immediately after finding the unload event in a scenario.

URL transitions are caused in several ways. A user can load a new Web page by specifying a URL directly to the browser, clicking a hyperlink, or submitting a form. Even the JavaScript program sometimes causes a URL transition. The player has to be able to handle all these kinds of URL transitions. It has to carefully handle the loading of the Web page generated by a CGI program after a form submission. There are two methods for form submission, GET and POST. Server-side programs generate an HTML page from the values of a submitted form. The player has to actually submit a form by using a JavaScript method when it finds a submit event in a scenario, because the player cannot get the Web page from only the recorded URL when the POST method is used for form submission. The player has to skip the next load event, because the next Web page has already been loaded by the playback of the form submission.

When a user loads a multi-frame Web page, load events occur from each frame. **Figure 5** (a) shows an example of a multi-frame Web page. Although the user can see only three frames in the browser, there are actually five frames, as shown in Fig. 5 (b). We call the frame that contains all the other frames the root frame. In Fig. 5, Frame 1 is the root frame. When a user loads this page, load events occur as shown in Fig. 5 (c). The load event of the parent frame occurs after all the load events of its child frames. Suppose the player plays back this load event sequence in the recorded order,

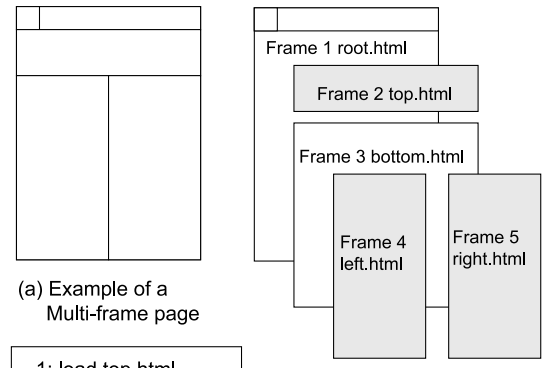


Fig. 5 Playback of multi-frame page loading.

as shown in Fig. 5 (c). Errors are caused by the playback of the first four load events, because the target frame does not exist yet. For example, Frame 2 does not exist until "root.html" for Frame 1 is loaded, but the load event of "root.html" is in the fifth line. In this case, the player cannot play back the first four load events. But the player does not know which load events cannot be played. Therefore, in this case, the recorder should record only the load events of the root frame. However, if some user's operations cause loading of a new Web page into Frame 4 or 5 after the loading event of Frame 1 has occurred, the new loading event of Frame 4 or 5 has to be recorded. In short, page loading caused by parent page loading should not be recorded, but other load events have to be recorded. The embedded JavaScript has to notify the recorder of the load event only when the parent page has already been loaded.

3.3 Time Management

Figure 6 shows an example of a scenario timetable. Figure 6 (a) shows the timetable for recording, and Fig. 6 (b) shows that for playback. The loading time is the interval between the issue of an HTTP request and the completion of the loading of all the contents, including images and applets. It depends on many factors such as the server load, network traffic, and client CPU power. Therefore the loading time at recording (T_1) may be different from that at playback (T_3); this implies that the player has to watch the behavior of the browser to avoid errors. For example, if the player tries to play back the clicking of a radio button when the Web page loading has not been completed, the browser issues an error if the browser cannot find the radio button object because it has not been loaded yet. Therefore the player does not duplicate the loading time (T_1). After detection of the load completion event, the player restarts playback of the scenario. In the playback case, each period of operations, except the loading time, is played back on the basis of the time stamps that are recorded for each operation. Therefore the browsing time during recording (T_2) and that during playback (T_4) are the same as in normal playback.

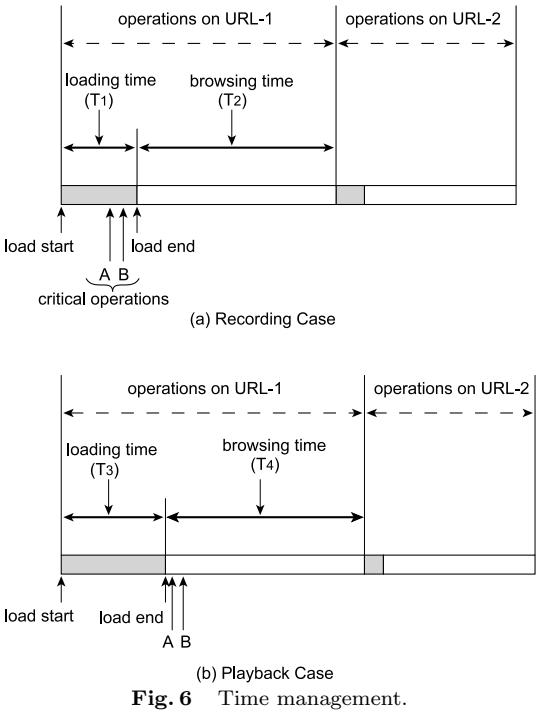


Fig. 6 Time management.

3.4 Critical Operation Events

The player removes the current Web page from a Web browser in order to load a new Web page. After loading the Web page, it creates objects such as frames, text fields, and images. During the Web page loading, the browser becomes unstable, because the objects of the old page no longer exist and it is not certain whether the objects of the new page have been created yet. But a user may be able to see a part of the Web page, even when all the objects of the new page have not yet been created. If the user performs some operations on the Web page when it has not been completely loaded, the player may cause errors by playing back such events when the page loading has not been completed. To avoid such errors, the player does not play back any operations while the browser is loading a Web page. The player has to change the playback timing of operations recorded during the following periods. In Fig. 6 (a), A and B are the operation events recorded during the page loading. The player replays them after the loading event, as shown in Fig. 6 (b).

3.5 Effective Playback

The player plays back a scenario by sending recorded events to the actual browser, not by showing a movie. Hence, it can provide some effective ways of playing back operations. The following are examples.

3.5.1 Playback Speed Control

The player can change the speed of playback. For example, a user can play a recorded scenario at double speed or one-third speed. As

explained in the previous section, the player cannot control the loading time, but it can control the browsing time. Therefore playback is controlled by controlling the browsing time.

3.5.2 Selective Event Playback

The player allows selective playback. In Web operations, many kinds of operation events are independent from the program viewpoint. For example, mouse movement, text input into a field, and scrolling of a browser window do not have specific relationships. Thus, the player can skip some kinds of operation events. This feature is useful for quick playback, because it allows a user to select the types of event to be played back; for example, it is possible to play back only the URL transitions, using the browser as an automatic Web navigation tool. By playing only the URL transition events at full speed, a user can put into the cache the Web pages he or she wants to see later.

3.6 Presentation Tools

3.6.1 Mouse Pointer

The recorder records mouse movement events detected by the embedded JavaScript. However, the player cannot control the system mouse, owing to the design of Java's security model. Since the player is written in Java and JavaScript, it plays back the mouse movement events by moving the image of a mouse pointer.

3.6.2 Annotation over the Web Page

The recorder and player provide an "ink" annotation function that allows the user to annotate Web pages. It is implemented by using the functions of Dynamic HTML. **Figure 7** shows our implementation of the annotation function. When mouse movement events are detected by the embedded JavaScript, the recorder creates small colored layers d_1 and d_2 at the points where the mouse movement events were detected on the Web page, as shown in Fig. 7 (a). In this way, colored layers are created according to the movement of the mouse pointer, and are then shaped as shown in Fig. 7 (b). The recorder also records the user's annotations, and the player plays them back.

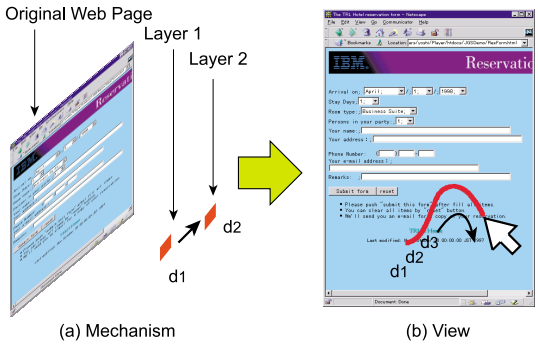


Fig. 7 Implementation of the ink annotation function.

3.6.3 Additional Objects on the Web Page

The recorder and player allow a user to attach his or her own additional objects such as text, images, and hyperlinks to a Web page. Vistabar [12] provides a function for attaching a text for a Web page. In the system, users can see attached text in a Vistabar window, while our system displays additional objects in a Web page. In our system, a user can locate an additional object anywhere in a Web page by using the functions of Dynamic HTML [1]. The recorder creates a new layer and puts HTML elements specified by a user onto that layer. A user can attach all kinds of HTML elements such as text, images, and hyperlinks, because any HTML expressions can be put onto a layer. Figure 1 (b) shows the user interface of our prototype. A user selects an image from the Sticker Window, shown in Fig. 1 (b), then specifies where the image is to be located on the Web page by means of mouse operations.

3.7 Implementation

We have implemented the recorder and player in Java and JavaScript for both Netscape Communicator 4 and Microsoft Internet Explorer 4 and 5. There are some differences between the implementation for Netscape Communicator and that for Microsoft Internet Explorer, because the Dynamic HTML compatibility between the browsers is not perfect.

4. Applications

In this section, we give some examples of applications.

4.1 Automatic Presentation

Web-based automatic presentation is one of the most promising applications for our system. A user can see an automatic presentation by simply clicking on a hyperlink for the presentation. When the user clicks the hyperlink, the player is automatically downloaded to the user's Web browser and starts the presentation. In this manner, Web sites can provide Web-based presentations such as site navigation or sample form input demonstrations for novice users. Distance learning is another important application for our system. Instructors can eas-

ily create their own scenarios by gathering useful Web pages from the Internet [20].

4.2 Analysis of Users' Behavior on the Web

It is important for Web site managers to analyze the behavior of their Web sites' users, since this allows them to evaluate the contents of their sites and the user interfaces of their Web-based applications. Some site managers now analyze the logs generated by Web servers. A log contains time stamps, accessed URLs, the types of Web browser used, and various other information. Analysis of logs provides enough information for surveying accesses to a Web site, but not enough for analyzing users' behavior in detail. The most effective way of understanding a user's behavior is to watch that user's operations on a Web browser.

With our mechanisms, an analyst can record users' operations on Web pages and replay them later. However, analysts have to be careful to warn users when they gather private information such as users' operations on a Web page [24].

4.3 Real-Time Web-Browser-Sharing System as a Collaboration Tool

Real-time Web browser sharing [7),9),11),19)] is useful in computer-supported collaboration. We realized Web browser sharing by connecting the recorder and player. **Figure 8** shows the mechanism of our implementation. To synchronize the Web browser at Node A with the one at Node B, the recorder at Node A detects the user's operations and sends them to the communication server. The communication server sends them to the player at Node B, and the player plays them back by using the Contents Window at Node B. Java's security model allows Java applets to have a connection only with the server machine from which the Java applet has been downloaded. Thus, the communication server must be located in the Web server in which the recorder and player applets are stored.

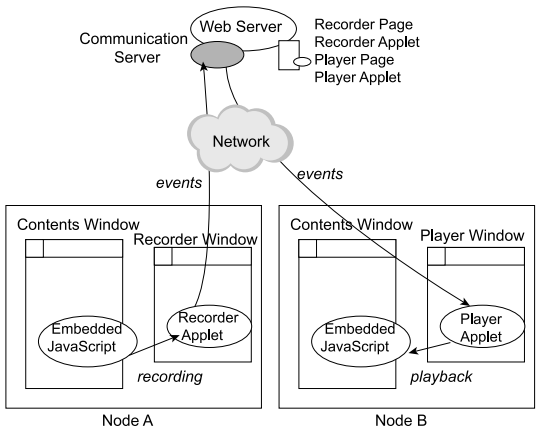


Fig. 8 Implementation of a Web-browser-sharing system based on the recorder and player.

5. Conclusions and Future Work

This paper has presented an event-based mechanism for recording and playing back Web browser operations. The recorder records a user's operations on a Web browser and plays them back by controlling the actual Web browser. Recordable operations include window sizing, window positioning, scrolling, form operation, and mouse pointer movement. In addition to ordinary Web operations, the recorder and player allow a user to make "ink" annotation and attach text, images, and hyperlinks on a Web page so that he or she can add explanations to the existing HTML contents. With this mechanism, users can easily create Web-based presentations by recording their operations. We have implemented the recorder and player in Java and JavaScript. Thus, end users do not need to install any software in their systems, except for a Web browser.

In evaluating the recorder and player with HTML contents on the Internet, we identified the following issues. Our proposed system can cope with all of them except the first:

- The recorder and player cannot support programs embedded in a Web page such as Java applets, plug-ins, and ActiveX controls. As we explained, there are several useful methods for supporting Java applets [2),4),15)].
- Some Web sites, such as news sites, frequently update their contents. When a Web page has been updated since it was recorded, the player has to warn the user. The same problem arises for Web pages generated dynamically. One solution to the problem is to save the HTML contents along with the recorded data.
- Submission of a form usually involves a transaction such as purchasing something or applying for a service. When a user does not want to generate a real transaction during playback, the player has to avoid replaying the submission event.

Despite the issues mentioned above, our experience has convinced us that our event-based recording mechanism based on the DOM is suit-

able for a Web browser. There are three reasons for this: The first is that our mechanism is very powerful, even though it is simple. The recorder can not only detect all types of operations required for a playback, but can also obtain many data related to a Web document on the basis of the DOM. The second reason is the volume of recorded data. Since it is much smaller than that of image-based recorded data, the downloading (or uploading) time is very short. The third reason is platform-independence. Our method is independent of the operating systems and browser, because HTML and Java are standardized. The recorder and player are implemented in Java and JavaScript, and can therefore work in many computing environments without requiring any client software to be installed. The only requirement for a client machine is a Java-enabled Web browser.

Our next step will be to create an editor for modifying recorded scenarios. In our experience, we have found that it is often necessary to modify a recorded scenario to improve the presentation or create a new presentation from existing ones.

Acknowledgments The authors would like to thank Dr. Toshio Souya for his implementation of the Embedding Engine, and Younosuke Furui for his implementation of the communication server. They would also like to thank their colleagues in the laboratory for their valuable comments.

References

1) Aoki, Y. and Nakajima, A.: User-Side Web Page Customization, *Proc. 8th Intl. Conf. on Human-Computer Interaction*, Vol.1, pp.580–584 (1999).
2) Begole, J.B., Struble, C.A., Shaffer, C.A. and Smith, R.B.: Transparent Sharing of Java Applets: A Replicated Approach, *Proc. UIST '97*, pp.55–64 (1997).
3) Bulterman, D.C.A., Hardman, L., Jansen J., Mullender, K.S. and Rutledge, L.: GRiNS: A GRaphical INterface for Creating and Playing SMIL Documents, *Proc. 7th Intl. World Wide Web Conf.* (1998).
http://www7.scu.edu.au/programme/fullpapers/1939/com1939.htm
4) Chabert, A., Grossman, E., Jackson, L., Pietrowicz, S. and Seguin, C.: Java Object-Sharing in HABANERO: A New Framework for Collaborative Tool Development Uses Any Platform That Supports Java, *Comm. ACM*, Vol.41, No.6, pp.69–76 (1998).
http://havefun.ncsa.uiuc.edu/habanero/
5) Cypher, A.: Eager: Programming Repetitive Tasks by Example, *Proc. CHI '91*, pp.33–39 (1991).
6) Cypher, A. (Ed.): *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA (1993).
7) Greenberg, S. and Roseman, M.: GroupWeb: A WWW Browser as Real Time Groupware, *Companion Proc. CHI '96*, pp.271–272 (1996).
http://www.acm.org/sigchi/chi96/proceedings/shortpap/Greenberg4/sg3txt.htm
8) Hoschka, P. (Ed.): Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, W3C Recommendation (1998).
http://www.w3.org/TR/REC-smil/
9) Jacobs, S., Gebhardt, M., Kethers, S. and Rzasa, W.: Filling HTML Forms Simultaneously: CoWeb – Architecture and Functionality, *Proc. 5th Intl. World Wide Web Conf.* (1996).
http://www5conf.inria.fr/fich_html/papers/P43/Overview.html
10) Kishi, N.: SimUI: Graphical User Interface Evaluation Using Playback, *Proc. COMPSAC '92* (*Computer Software and Applications Conf.*), pp.121–127 (1992).
11) Kobayashi, M., Shinozaki, M., Sakairi, T., Touma, M., Daijavad, S. and Wolf, C.: Collaborative Customer Services Using Synchronous Web Browser Sharing, *Proc. CSCW '98*, pp.99–108 (1998).
12) Marais, H. and Bharat, K.: Supporting Cooperative and Personal Surfing with a Desktop Assistant, *Proc. UIST '97*, pp.129–138 (1997).
13) Maulsby, D. and Witten, I.: Inducing Programs in a Direct-Manipulation Environment, *Proc. CHI '89*, pp.57–62 (1989).
14) Mima, Y.: A Visual Programming Environment for Programming by Example Abstraction, *Proc. 1991 IEEE Workshop on Visual Languages*, pp.132–137 (1991).
15) Miura, M. and Tanaka, J.: A Framework for Event-Driven Demonstration Based on the Java Toolkit, *Proc. APCHI '98* (*Asia Pacific Computer Human Interactions*), pp.331–336 (1998).
16) Myers, B.: Creating User Interfaces Using Programming-by-Example, Visual Programming, and Constraints, *ACM Trans. Programming Languages and Systems*, Vol.12, No.2, pp.143–177 (1990).
17) Roseman, M. and Greenberg, S.: Building Real-Time Groupware with GroupKit, A Groupware Toolkit, *ACM Trans. Computer-Human Interaction*, Vol.3, No.1, pp.66–106 (1996).

18) Rousseau, F. and Duda, A.: Synchronized Multimedia for the WWW, *Proc. 7th Intl. World Wide Web Conf.* (1998). http://www7.scu.edu.au/programme/fullpapers/1833/com1833.htm
19) Sakairi, T., Shinozaki, M. and Kobayashi, M.: CollaborationFramework: A Toolkit for Sharing Existing Single-User Applications without Modification, *Proc. APCHI '98* (*Asia Pacific Computer Human Interactions*), pp.183–188 (1998).
20) Souya, T., Ohtani, C., Aoki, Y., Masuda, Y. and Nakajima, A.: How to Show Web Pages for Learners: Teaching and Learning with Web Recorder, *Proc. ED-MEDIA 2000*, pp.1038–1043 (2000).
21) Sugiura, A. and Koseki, Y.: Simplifying Macro Definition in Programming by Demonstration, *Proc. UIST '96*, pp.173–182 (1996).
22) Sugiura, A. and Koseki, Y.: Internet Scrapbook: Automating Web Browsing Tasks by Demonstration, *Proc. UIST '98*, pp.9–18 (1998).
23) W3C (World Wide Web Consortium), Document Object Model. http://www.w3.org/DOM/
24) W3C, Platform for Privacy Preferences P3P Project. http://www.w3.org/P3P/.
25) Yamaguchi, T., Hosomi, I. and Miyashita, T.: WebStage: An Active Media Enhanced World Wide Web Browser, *Proc. CHI '97*, pp.391–398 (1997).

(Received May 1, 2000)
(Accepted October 6, 2000)

Yoshinori Aoki received the B.E. and M.E. degrees from Kyushu University, Fukuoka, Japan, in 1995 and 1997, respectively. In 1997, he joined Tokyo Research Laboratory, IBM Japan, Ltd. He has worked on Web-based interactive system designs in the laboratory. His research interests include human-computer interaction, XML, and distributed systems. He is a member of the ACM.

Fumio Ando received the B.E. and M.E. degrees in electronic engineering from the University of Tokyo in 1988 and 1990, respectively. He joined IBM Japan in 1990. He has worked in IBM Research, Tokyo Research Laboratory for 10 years. Currently, he is a manager of Technical Plans & Control. His research interests include groupware, human-computer interaction and distributed systems.

Amane Nakajima received the B.E. degree in electronic engineering in 1983 and the M.E. degree in electrical engineering in 1985 from the University of Tokyo. He joined IBM Japan in 1985. He worked in IBM Research, Tokyo Research Laboratory for 15 years. Currently, he is a program manager in IBM Global Services. His research interests include human interaction systems and distributed systems. He received the Best Paper Award from the Institute of Electronics, Information and Communication Engineers of Japan in 1987. He is a feature editor of IEEE Communications magazine. He is a member of the IEEE and the Association for Computing Machinery.