Vol. 43 No. 2 IPSJ Journal Feb. 2002

Regular Paper

Building a Collaborative Web Environment for Supporting End Users

Yoshinori Aoki[†]

This paper describes methods for developing a Web-based collaborative environment for call center agents supporting end users by using real-time Web browser sharing techniques. The collaborative functions should be available for end users without any preparation on the client side. This is because it is unacceptable for end users, especially novice users, if the system requires the users to download and install software on their client PCs in order to use the collaborative functions when they run into problems on the Web site. On the other hand, it is important for Web site developers to separate content design and collaborative-function development. If they are not separated, content designers have to create collaboration-aware content, and existing content cannot be reused with the collaborative functions. This paper discusses three approaches for developing real-time browser sharing systems, and shows why the proxy-based approach is the best to meet the above requirements. Collaboration tools, such as telepointers and annotations, can help an end user communicate with a call center agent. However, if the layouts of the same page are different among the shared browsers, coordinate-based telepointers and annotations will not be displayed at appropriate positions. This paper also explains methods for synchronizing Web page layout.

1. Introduction

Many services, such as Internet auctions and online banking, are provided as Web-based services. Users access such services by using Web browsers. Users often have to go through awkward steps to use the services or fill out complicated Web forms to apply for the services. Hence, some of the users abandon their purchases midway, or even change service providers to find more user-friendly services. In such cases, the Web sites are losing business opportunities. Therefore, it is very important for Web-based service providers to build a user-supportive Web site.

This paper explains a system that supports an end user's operations using a Web browser by synchronizing the user's browser with a call center agent's browser as shown in **Fig. 1**. This system allows end users to collaboratively work on a shared browser when they cannot complete their tasks by themselves or have some questions about the content.

Microsoft's NetMeeting and Lotus's Sametime are commercial products that allow users to share desktop applications with remote users. They capture the screen as an image and send it to the other users. The advantages of this approach are: (1) Users can share any desktop applications, and (2) Users can share their applications even though they are not installed on the partners' client machines. They naturally allow users to share their Web browsers, too. However, their performance is inadequate over the Internet because they exchange large volumes of data.

Another approach is exchanging events between applications that are running on each client. The advantage of this method is its high performance, because the amount of data actually exchanged is very small. Some real-time browser sharing techniques have already been proposed[5),16),23),38)]. However, they have the following problems:

(1) Users have to install collaboration-aware browsers[15),16)] or plug-ins[23),26),27),33)] in their clients in advance. Therefore they cannot seamlessly start using the collab-

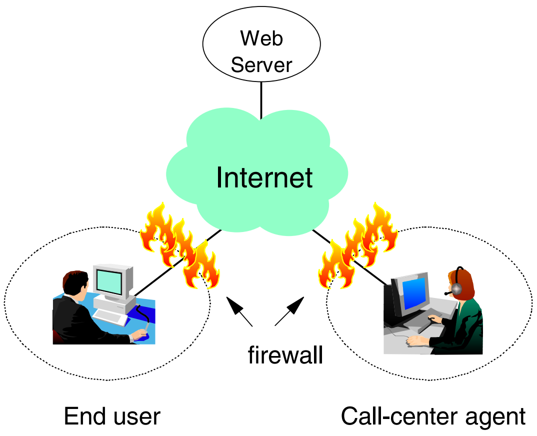


Fig. 1 Application scenario.

† IBM Research, Tokyo Research Laboratory

orative functions when the need arises.

(2) Content designers have to develop collaboration-aware content in a special manner [5]. Therefore content designers have to be familiar with the collaborative functions. In addition, existing content is not reusable in such systems.

(3) When browsers' font configurations and other settings are different among the shared browsers, Web pages are displayed with different layouts [20]. Therefore, it becomes impossible to show coordinate-based telepointers and annotations in appropriate positions.

This paper describes a novel technique for synchronizing Web browsers that solves the above problems.

The rest of this paper is organized as follows. The next section explains the requirements for the application shown in Fig. 1. The following section explains related work and classifies the approaches into three categories. This paper then explains the details of our system and presents my conclusions.

2. Requirements

This section describes the requirements for building the Web-based collaborative environment shown in Fig. 1. In Fig. 1, the call center agent is supporting the end user by looking at the same Web page on the shared browser. The agent and end user may both be working behind firewalls. The following are the requirements for the application.

- **No special installation:** Collaborative functions should work with normal Web browsers without any plug-ins. Some of the previously proposed systems require end users to install collaboration-aware Web browsers [15),16] or plug-ins [23),26),27),33] to provide collaborative functions for Web browsers. However, such installations bother end users, especially novice users. Collaborative functions should be available without any installations when the need arises.
- **Separating content design and collaborative function development:** Collaborative functions should be separated from content, so content designers do not need to be aware of and include the collaborative functions in the content. The ideal is that content designers can create content without being conscious of collaborative functions, because content designers and developers usually work separately. The separation also allows the system to reuse existing content.
- **Web page layout sharing:** If browsers' default fonts, text sizes, or window sizes are different, the layouts of the same Web page will be different among the shared browsers. This is a problem because some of the collaboration tools, such as telepointers and annotations, are based on window coordinates. **Figure 2** shows an example Web page with some annotations in different layouts. Hence, the layouts of the same Web page should be the same among the shared browsers.
- **Session management:** The sessions shared between browsers should be managed to support (1) dynamic Web pages, (2) transaction management, and (3) the SSL (Secure Sockets Layer) protocol [9]. Some Web pages are dynamically generated by server-side programs such as servlets [21] or CGI [19] programs. If shared Web browsers independently access a Web server, they may receive different Web pages. In addition, if shared Web browsers independently submit a shared Web form, multiple transactions will take place. The SSL protocol should also be supported for secure transactions in a session.

3. Approaches

Many real-time Web browser sharing techniques have been proposed, and have classified these approaches into the following three categories:

(1) Client-based approaches.

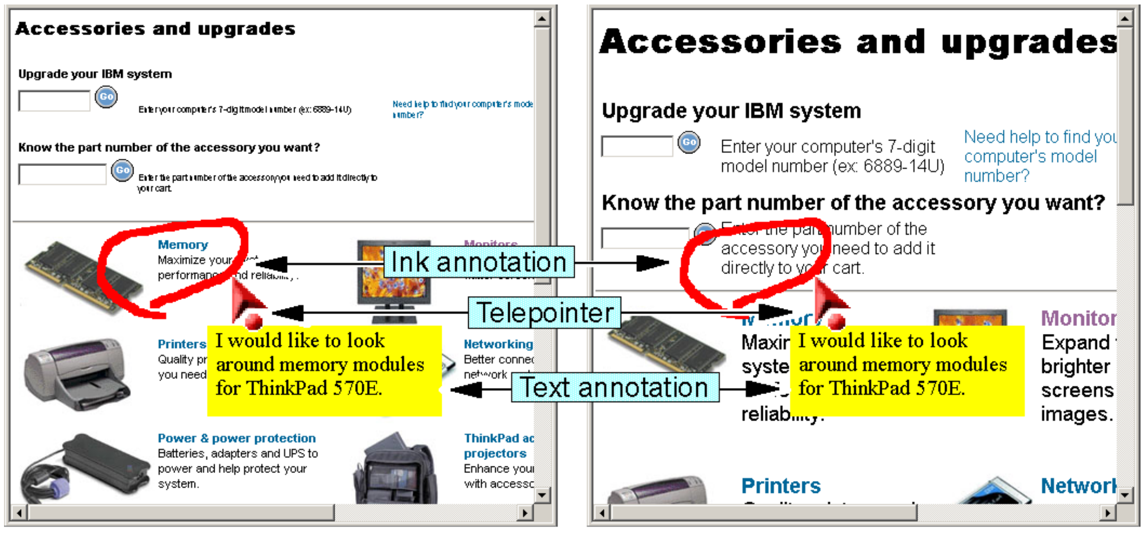
(2) Server-based approaches.

(3) Proxy-based approaches.

We have adopted the proxy-based approach. The following sections explain the three approaches in detail and why I chose the proxy-based approach.

3.1 Client-based Approach

In the client-based approach, collaborative functions are implemented in client-side software, and users have to install it in advance. An advantage of this approach is that the existing content can be used with the collaborative tools, because the collaborative functions are isolated in the client-side software. The main disadvantage is that users have to install the software in advance, which prevents end users



I would like to look around memory modules for ThinkPad 570E.

(a) Layout in a small text size

(b) Layout in a large text size

Fig. 2 Shared Web page with different font configurations.

from using the collaborative functions spontaneously.

GroupWeb[16] and GroupSpace[15] are systems implemented using this approach (GroupSpace is actually a mix of the client- and server-based approaches). Original Web browsers were developed to provide the collaborative functions in these systems. Many toolkits[26],[31] have been developed, and developers can reduce the cost of developing collaboration-aware applications by using such toolkits. However, it is very costly to develop original Web browsers that fully support recent standards such as HTML, HTTP, XML, JavaScript, Java, SSL, etc. Therefore, another client-based approach has been proposed, which adds collaborative functions to existing Web browsers without any modifications by installing plug-ins in the clients[23],[26],[27],[33]. To realize the collaborative functions, the plug-ins control the browsers via IPC (Inter-Process Communication) calls such as DDE[30] in Windows. Sakairi, et al. proposed a toolkit with which developers can add multi-user functionalities to an existing single-user application[32]. WebShare[33] is implemented with the toolkit, and is an add-on module that enables existing browsers to provide synchronous-browsing capability. Another disadvantage of this approach is that it is impossible to capture events in accord with the application semantics and control the application properly if the application does not provide an interface for the plug-ins.

3.2 Server-based Approach

In the server-based approach, collaborative functions are tightly integrated into the content. The chief advantage of this approach is that existing browsers are already suitable, because the collaborative functions are implemented on the server side. However, content designers have to create collaboration-aware content, and hence they are required to have not only artistic design skills but also programming skills. In addition, it is impossible to reuse existing collaboration-unaware content.

Artefact[5] is implemented using the server-based approach. Artefact is an environment for developing CORBA-based[39] collaborative Web applications. In Artefact, the content itself is written in ADL, a special XML-like language. The content written in ADL is transformed to HTML documents by server-side applications. Users can issue events for the server-side CORBA applications by clicking hyperlinks or by submitting forms from their Web browsers. GroupSpace (introduced in the previous section) extends the HTML format by adding two tags with which we can add collaborative functions to an HTML document[15]. Therefore GroupSpace can also be categorized as using a server-based approach.

3.3 Proxy-based Approach

In the proxy-based approach, the collaborative functions are embedded into the content while passing through a proxy server, and the proxy server supports the requested collaborative functions. This approach is similar to the server-based approach in that collaborative functions are embedded in the content. However, while collaborative functions are tightly integrated into the content in the server-based approach, they are automatically inserted in the proxy-based approach. This means that there need be little dependency between the collaborative functions and the content, and therefore the content designer can create the content without being conscious of the collaborative functions, and existing collaboration-unaware content is reusable with minimum effort. In addition, users can use existing Web browsers without any modifications or plug-ins, because collaborative functions are dynamically embedded into the content. I adopted the proxy-based approach because of these advantages.

WBI[3),4)] provides a framework with which developers can build new functions on a conventional proxy server. Hence it allows developers to reduce the cost of developing a customized proxy server.

Several proxy-based systems have been proposed[6),22)]. In CoWeb[22)], the proxy server replaces all input fields of an HTML form by Java applets that provide collaborative input capability. However, CoWeb does not support general Web pages except for HTML forms. In addition, some HTML forms include JavaScript code, typically calculations of a total price or input value validation, and they may not work correctly if the input fields are replaced by Java applets. In Ref. 6), a Java applet is inserted into an HTML document, and the Java applet exchanges the URL of the page with other browsers to display the same Web page. However, the collaborative functions are not adequate to support end users on a Web browser, because though users can see the same Web page they cannot share the form input, scrolling, Web-page layout, and window operations. In addition, this system also does not support telepointers nor annotations.

Many Java-applet-based systems have been proposed to realize collaborative environments using Web browsers[7),10),11),24),25),28),29),34),36)]. These systems try to realize shared workspaces using Java applets, and their view of Web page sharing is limited to URL synchronization. This is because Java applets lack the capabilities required to control Web browsers. Java applets can only identify the URL of the Web page in which the Java applet is embedded or load a Web page into the frame where the Java applet is embedded[8)]. On the other hand, our system tries to use Web pages as shared workspaces. The proxy of our system embeds not only Java applets but also JavaScript programs. The Java applets provide only communication capabilities and the collaborative functions are basically implemented in JavaScript. This is because JavaScript can access the objects, such as images and link objects, in a Web page via the DOM (Document Object Model)[37)] interface, which provides methods for capturing events on a Web page and for directly controlling the objects in the page. Therefore our system provides not only URL synchronization, but also synchronization of form input, scrolling, and window operations. In addition, our system also supports telepointers and annotation functions.

3.4 Contribution

This section summarizes the problems in the previous studies, and explains the contribution of this paper.

The main disadvantage of the client-based approaches is that such systems force end users to install software in advance and prevent end users from using the collaborative functions spontaneously. The main disadvantage of the server-based approaches is that content designers have to create collaboration-aware content in special manners because the content and the collaborative functions are tightly integrated. The advantage of the proxy-based approach is that such systems can solve the above two major problems. On the other hand, the main problem of existing proxy-based systems is poor functionalities for browser synchronization. This is because they are based on Java applets, and the Java applets lack the capabilities for detecting operations on Web pages and controlling the Web browsers.

The main contribution of this paper is a novel method for event detection and browser control by combining JavaScript and Java. With this method, our system realizes comprehensive browser synchronization capabilities without losing the advantages of the proxy-based approach including (1) end users need not install any software in advance, and (2) content design and collaborative function development are completely separated. Our system supports synchronization for not only URLs, but also form inputs, scrolling, and window operations.

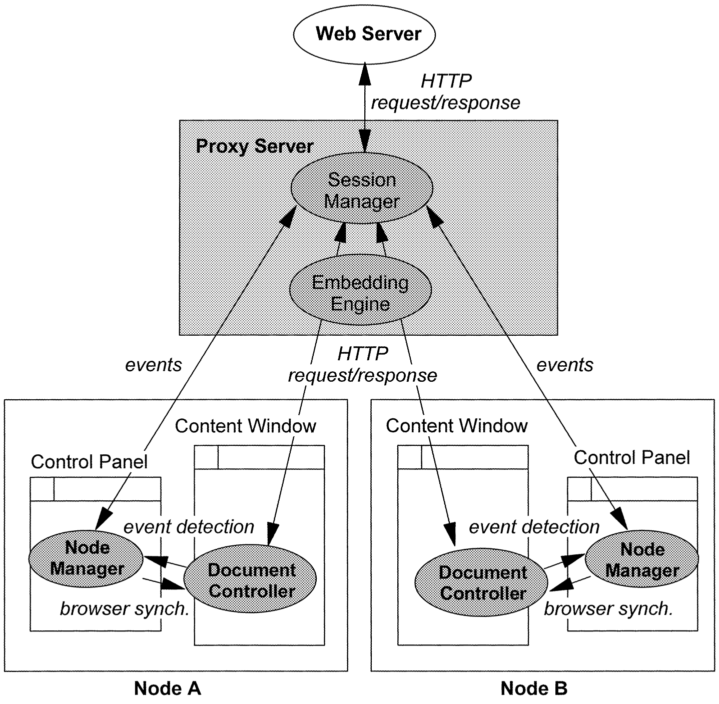


Fig. 3 System architecture.

In addition, our system also provides collaboration tools including telepointers, image annotations, text annotations, and ink annotations directly attached to Web pages. The SSL protocol support, intranet user support, and Web-page-layout-sharing technique are also contributions of this research.

4. Collaborative Environment on the Web

This section describes our general solution for building collaborative environments for Web users.

4.1 Architecture

Figure 3 shows the architecture of our system. In Fig. 3, two Web browsers are running on Nodes A and B, sharing one application, and both users can access the collaborative functions using their *control panels*, and see the Web pages in their *content windows*.

Our system consists of three components. One is a proxy server and the others are client-side programs called the *document controller* and the *node manager*. The proxy server includes the *embedding engine* and the *session manager*. The embedding engine inserts the document controller into an HTML document, and the session manager provides session and security management. Each node runs a node manager that communicates with the session manager to support that node's participation in the shared session. Every HTML document (or frame of a compound document) has a document controller embedded in it to control the display of that document.

When a user takes an action on a Web page, the document controller detects the event, and notifies the node manager. For example, when the user inputs a value into a text field on a Web form, then the document controller detects that the value has been changed and sends to the node manager that information, including the frame id, form id, input-field id, and input value. The node manager then sends this to the session manager, and the session manager distributes it to the other node managers. Each node manager executes an event to synchronize its own browser by using the methods provided by the document controller. The document controller synchronizes its own browser using JavaScript methods. For example, when a value "Japan" was input into a text field on Node A, the browser on Node B can be synchronized by using the following JavaScript call.

```
document.forms[0].elements[2].value="Japan";
```

With the control panel, a user can select between normal operation mode, in which the user can browse as usual, and annotation mode, in which the user can attach annotations to Web pages. When the user changes the mode, the control panel notifies the document controller of the mode change. The user can also close the collaborative session with the control panel. When the user closes the session, the node manager notifies the session manager and the session manager distributes the status change to any other node managers in the session.

Both the control panel and content window are browser windows, and the node manager and the document controller are implemented in Java and JavaScript, and executed in the Web browsers. The node manager has to be downloaded directly from the proxy server to communicate with the session manager under the Java security model 8). Users do not have to install any software to use the collaborative functions, because all the client-side components are downloadable.

When a Web page is requested, the proxy server obtains the Web page from the Web server, parses the HTTP response, and embeds a document controller into the HTML document. The document controller is embedded into every HTML document. When a Web page consists of multiple frames, every frame has to contain a document controller. After an HTML document is downloaded, the document controller is activated, parses the HTML document, and sets up event handlers for the appropriate objects in the page. The document controller then detects events via the DOM interface, and controls its HTML document by using the DOM interface. The document controller consists of one JavaScript file and one Java applet, and must be embedded at the end of an HTML document by using SCRIPT and APPLET tags. The best place is just before the end tag of the body tag, "⟨/BODY⟩." This is because if the document controller is embedded at the beginning of an HTML document, the document controller starts parsing the HTML document before it has finished loading the HTML document. This may cause failures in setting up the event handlers. An alternative method is calling the event-handler-set-up method as an onload event call. Details of the event detection and the program insertion in the embedding engine appear in Refs. 1), 2).

4.2 Session Manager

A user has to establish a session to start collaboration by loading the node manager in a Web browser. After the node manager is activated, it communicates with the session manager to find partners. In the application shown in Fig. 1, a proxy server will be managed by the call center, and the session manager finds an available agent to help the user. The session manager maintains the session with the participating node managers running on the shared browsers. The session manager provides the following functions:

(1) Dynamic Web page support.
(2) Transaction management.

Many Web servers dynamically generate Web pages using server-side programs such as servlets and CGI programs. If the shared browsers independently request Web pages from such a Web server, the browsers may receive different Web pages even though they request the same URL, because each browser is uniquely identified by the Web server. In addition, when a user submits a shared form, multiple transactions will take place, because all the shared browsers will submit their forms. Master-slave browsing can avoid such problems. In master-slave browsing, only one user acts as the master and the rest of the users act as slaves. When a master or a slave clicks a link or submits a form, the session manager distributes the event to all the shared browsers and each of the browsers executes the event and then sends an HTTP request to the session manager. However, the session manager relays only the master's HTTP request to the Web server though it accepts the HTTP requests from all the shared browsers. After receiving the HTTP response from the Web server, the session manager distributes the HTTP response not only to the master browser but also to the slave browsers. In this way, the session manager provides dynamic Web page support and transaction management.

4.3 SSL Protocol Support

The SSL protocol is used to encrypt HTTP requests and responses when a browser and a Web server exchange confidential data such as user names and credit card numbers. The SSL sessions are usually established between the browser and the Web server. However, in naive master-slave browsing of a secure site, the following problems occur, because of the data encryption between browsers and the Web server.

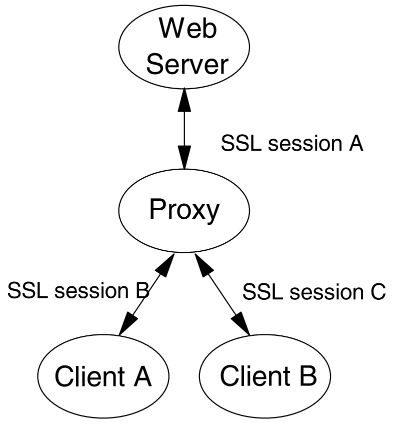


Fig. 4 SSL protocol support.

(1) The proxy server cannot examine or manipulate the encrypted data. Hence, the proxy server cannot insert a document controller into the HTML document.
(2) The session manager relays only the master's HTTP requests to the Web server. Hence, the HTTP requests and responses are encrypted for the master browser. It means the slave browsers cannot decrypt the HTTP responses, even if the session manager sent the HTTP responses not only to the master browser but also to the slave browsers.

To solve the above problems, the proxy server has to decrypt the HTTP responses received from the Web server, insert a document controller into the HTML document, and then encrypt the modified HTTP responses for each browser again. As shown in **Fig. 4**, independent SSL sessions have to be established between the proxy server and the Web server, and between the proxy server and the browsers.

4.4 Proxy Configuration

The proxy server plays the central role in our system, and strongly affects the application scenario. This section discusses the proxy configuration.

4.4.1 Problems

In the application shown in Fig. 1, both the end user and the call center agent are in different intranets. **Figure 5** shows the proxy configuration in such case. There are the following two problems in the proxy configuration shown in Fig. 5.

(1) Intranet support.
(2) SSL protocol support.

In Fig. 5, each proxy server prevents its intranet user from directly accessing Web servers outside the intranet. The user has to access the Web servers via the proxy server. In this situation, the normal proxy servers as shown in Fig. 5 do not work for collaboration with the other intranet users. This is because intranet users have to use their own proxies, and hence they cannot establish a collaborative session by sharing a proxy server that includes the session manager.

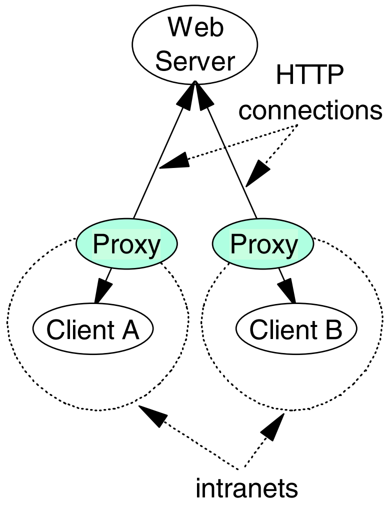


Fig. 5 Proxy server.

As mentioned in Section 4.3, independent SSL sessions have to be established to support the SSL protocol, because the proxy server has to modify the HTML documents to embed document controllers. However, in Fig. 5, HTTP connections are established directly between browsers and an actual Web server. Hence, the browsers will try to establish SSL sessions directly between the browsers and the Web server. This means that the proxy server cannot insert a document controller into the HTML document, because the data is encrypted between them. Therefore, the proxy servers cannot support the SSL protocol in the way required here.

4.4.2 Solution

Our solution for the problems described in the previous section is to implement the proxy server as a reverse proxy server.

Reverse proxy servers are generally used for load balancing, caching, and redirection. The number of HTTP connections is the major difference between a normal proxy server and a reverse proxy server. In a normal proxy server, a Web server is a destination address of an HTTP request sent by a browser. The nor-

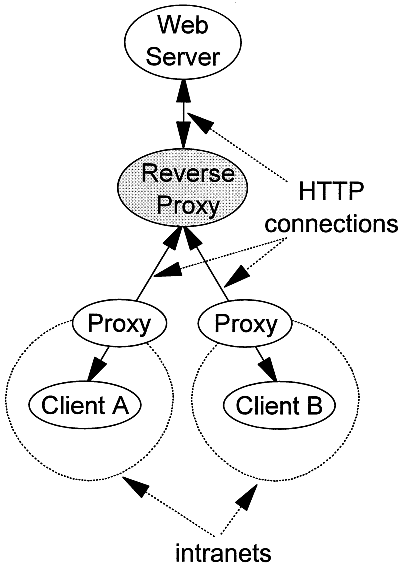


Fig. 6 Reverse proxy server.

mal proxy server just fetches the HTTP request and relays it to the Web server. Hence, there is one HTTP connection between the browser and the Web server. On the other hand, in a reverse proxy server, the reverse proxy server is a destination address of an HTTP request sent by a browser. The reverse proxy server creates a new HTTP request and sends it to a Web server. The Web server is a destination address of the new HTTP request sent by the reverse proxy server. Hence, there are two HTTP connections, one between the browser and the reverse proxy server, and another between the reverse proxy server and the Web server. This is because a reverse proxy server acts as the Web servers' proxy, and works like a Web server. The reverse proxy server accepts HTTP requests as though it were a Web server and sends the HTTP responses to browsers after getting the Web pages from backend Web servers. Hence, Web browsers regard a reverse proxy server as a Web server.

Figure 6 shows how the reverse proxy server works in our system, and the followings explain why the reverse proxy server can solve the problems.

(1) **Intranet support:** The reverse proxy server is accessible for users of different intranets as shown in Fig. 6. This is because browsers regard a reverse proxy as a Web server, so even if there is a proxy server between a browser and the reverse proxy server, the browser can reach the

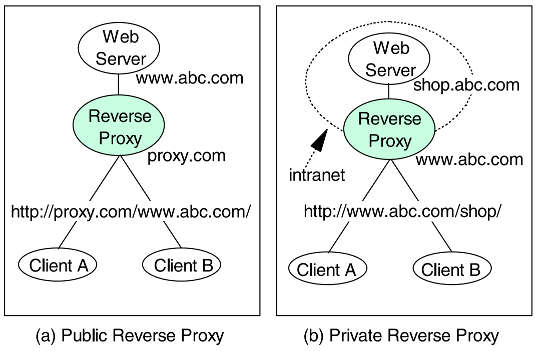


Fig. 7 Public and private reverse proxies.

reverse proxy server in the same way as for normal Web accesses. Therefore the reverse proxy allows users to establish a collaborative environment with users of other intranets.

(2) **SSL protocol support:** When a reverse proxy server is used, the browsers will try to establish the SSL sessions between the browsers and the reverse proxy server as shown in Fig. 6. This is because HTTP connections are established between the reverse proxy server and the browsers. The reverse proxy server can then establish another SSL session between the reverse proxy server and the actual Web server. After establishing the SSL sessions, HTTP requests and responses can be sent via the secure sessions.

4.4.3 Usability of Reverse Proxies

All end users have to do to use a proxy server is configure their browsers to make all requests through the proxy server. On the other hand, how users work with a reverse proxy server depends on the proxy configuration.

When we provide user-support services that are available for any Web site on the Internet, the proxy server has to be located on the Internet. End users can visit any Web sites via the proxy server. We call this a *public reverse proxy*. **Figure 7**(a) illustrates an example of accessing a Web server "www.abc.com" via a public reverse proxy "proxy.com." Users have to visit all Web sites via the reverse proxy in order to remain in the collaborative environment. In the example, the client sends an HTTP request to the reverse proxy. The URL of the HTTP request is "http://proxy.com/www.abc.com/," and the first half indicates the name of the reverse proxy, and the latter half indicates the URL of the Web page that the user actually trying to see. After receiving the HTTP request, the reverse proxy extracts the URL of the actual Web page, and sends a new HTTP request to the Web server to receive the actual Web page. Therefore users are not allowed to load a Web page by using their bookmarks or by directly typing a URL in the address bar of their browsers. One implementation approach is for users to log on from the start page of the reverse proxy, and then input a URL in the form provided by the reverse proxy. Because of this, end users should be aware of the mechanisms involved in order to use this system with a public reverse proxy and the use of a public reverse proxy is very complex.

When a Web site provides user-support services only for its own content, the reverse proxy is better as its Web server. The reverse proxy works only for the Web site; hence users cannot establish a collaborative environment with other Web sites. We call this a *private reverse proxy*, as shown in Fig. 7 (b). In Fig. 7 (b), the reverse proxy "www.abc.com" acts as a Web server for the domain "abc.com." The actual content is stored or generated in the other Web servers such as "shop.abc.com" within the intranet. Therefore, end users do not have to be aware of the existence of the Web servers behind the reverse proxy, and the usability of a private reverse proxy is much better than a public reverse proxy.

4.4.4 SSL Protocol in Reverse Proxies

When the SSL protocol is used, the browser establishes an SSL session with the reverse proxy as shown in Fig. 4. In the SSL session, the browser receives the Web server's certificate issued by a CA (Certification Authority) and allows the end user to look at the certificate [12]. In Fig. 7 (a), the end user can only look at the certificate of "proxy.com," even though the actual content is from "www.abc.com." Therefore, end users cannot be sure what Web site is providing the actual content when they are using a public reverse proxy. On the other hand, a private reverse proxy allows end users to look at the certificate of its domain. In Fig. 7 (b), the user can look at the certificate of "www.abc.com."

When the user uses a private reverse proxy, it is not necessary to establish an SSL session between the reverse proxy and the Web server, because the session is inside the intranet. The SSL protocol, especially the encryption module, usually makes heavy demands on CPU resources, and therefore the performance of the private reverse proxy is better than the public reverse proxy.

4.4.5 Summary of the Discussion

By implementing a proxy server, including the session manager and the embedding engine, as a reverse proxy server, we can support both different intranet users and the SSL protocol.

The reverse proxy server can be classified into public and private reverse proxies. The private reverse proxy is much better than the public reverse proxy from the viewpoints of usability and also the SSL protocol support.

4.5 Collaboration Tools

Our system provides the following features to communicate with other users. Figure 2 (a) shows examples of these collaboration tools.

- **Telepointer:** A telepointer is displayed on a Web page by using an IMG tag. When the mouse pointer moves on the Web page, the document controller detects the movement and sends it to the other nodes via the session manager. The document controllers on the other nodes move the telepointer by using a JavaScript method.
- **Image annotation:** When a user attaches an image to a Web page, the document controller dynamically creates a new layer for the image on the Web page.
- **Text annotation:** A user can position text on a Web page as shown in Fig. 2. First the user can create a new colored panel within a layer in a text-annotation mode, and then a text field will be created in the layer. The user then can directly write text in the text field.
- **Ink annotation:** When a user moves a mouse pointer on a Web page in the ink annotation mode, small colored layers will be created along the path of the mouse movement.

These functions are implemented in JavaScript by using Dynamic HTML functions [13]. Many toolkits and frameworks have been described for developing Java-applet-based collaborative applications [7),10),11),24),25),28),29),34),36)]. It is also possible to implement other collaboration tools, such as a shared chalkboard and chat functions, by using such toolkits or frameworks.

4.6 Web Page Layout Sharing

The layout information in the HTML document is not the only constraint on the display of Web pages. Browsers also use local state information like font configurations and window sizes that affects the display. There are two methods for sharing pages in an environment in which page layouts can be changed according to the window size and font configurations:

(1) Strict layout sharing.
(2) Relaxed layout sharing.

The following sections explain the above two methods and the method actually used in our system.

4.6.1 Strict Layout Sharing

In a strict layout sharing system, the page layouts are completely synchronized, and users see the same page in the same layout. It is possible to achieve strict layout sharing in Web browsers by synchronizing the window sizes and by specifying fonts and text sizes for all objects in the Web pages by using CSS (Cascading Style Sheets)[14] in advance. Fonts and text sizes specified in CSS take precedence over a browser's font configurations. Therefore content designers can strictly specify page layouts without consideration of the browsers' font configurations. However, there are some disadvantages in the CSS-based strict layout sharing as follows.

- Users cannot see Web pages according to their own font preferences. Different users naturally prefer different configurations.
- It is extra work for content designers because they have to specify fonts and text sizes for all objects in each Web page.
- It is not possible to reuse existing content in which fonts and text sizes are not specified in CSS.

4.6.2 Relaxed Layout Sharing

In a relaxed layout sharing system, the page layouts are not synchronized, and users see the same page with different layouts. In GroupWeb, a semantic telepointer is implemented as one of their browser's functions[17),18]. The telepointer indicates the same object even when the page layouts are different between the shared browsers. Relaxed layout sharing system like in GroupWeb can solve the problems described in Section 4.6.1. However, it is difficult for relaxed layout sharing systems to appropriately display text, images, and ink annotations that are directly attached to a Web page as shown in Fig. 2. Such annotation functions are very useful to communicate with other users in a collaborative environment. (Tang reported that annotating text and graphics directly on pages accounts for 65% of all actions in conventional collaborative environments[35].)

4.6.3 Dynamic Strict Layout Sharing on Web Browsers

As described in the previous sections, strict layout sharing has to be implemented within Web browsers to support text, image, and ink annotations on Web pages.

In the application shown in Fig. 1, CSS font specifications should not be used, avoiding the problems explained in Section 4.6.1. Our system realizes strict layout sharing by dynamically synchronizing the agent's page layout with the end user's page layout. Hence, end users can see Web pages according to their font preferences.

The agent's browser can display Web pages in the same layouts as the end user's layouts by using the following steps:

(1) When a master browser (the end user's browser) loads a new page, the document controller embedded in the page extracts the font and text size information from all objects in the Web page by using the DOM interface.
(2) The document controller sends the font information to the session manager via the node manager. The session manager sends it to the node manager of the slave browser (the agent's browser).
(3) According to the font information, the node manager of the slave browser instructs the document controller to dynamically change fonts and text sizes by using the DOM interface.

The problem with this method is that the slave browser cannot immediately display the Web page in the same layout as the master browser's after loading a new page. This is because the slave browser first displays the Web page according to its own font configuration, and after receiving font information from the master browser, the slave browser dynamically changes the page layout. In the application shown in Fig. 1, the issue is not a serious problem if the agent is aware of it in advance.

4.7 Implementation and Evaluation

The document controller, node controller, embedding engine, and a part of the session manager have already been implemented, and we can establish a collaborative environment with normal Web browsers. By implementing and evaluating our prototype systems, we found several issues as described below.

(1) There are incompatibilities in the JavaScript between Microsoft Internet Explorer and Netscape Communicator. Hence, we have developed separate versions of the system for the two major browsers, even though the basic architectures are the same. Our architecture theoretically allows us to establish a collaborative environment with an Internet Explorer user and a Netscape Communicator user. However, it is impossible to synchronize Web page layout, because of the different implementations of the page layout rendering engines between the two browsers. This implies that telepointers and annotations cannot be displayed in correct positions in such environments.

(2) We found that too many events are captured by the document controller and the performance of the system is harmed if a document controller sets up event handlers for all of the objects in the Web page. We avoided this problem by tuning up the event-handler-set-up code. For example, we removed all mouse-move event handlers from all objects except body objects. This is because a body object represents the whole body of a Web page, hence we can capture coordinates of the mouse pointer from the body object, wherever on the Web page the mouse pointer is positioned.

(3) The performance is poor for the ink annotation function in the Internet Explorer, although it works well in Netscape Communicator. This is because may colored layers are dynamically created along the path of mouse movement, and the performance in creating new layers depends on the rendering engine of the browser.

5. Conclusions

This paper describes some requirements for building a collaborative environment for supporting Web users by using a real-time browser sharing technique. I classified the previously proposed systems into three approaches: (1) client-based approaches, (2) server-based approaches, and (3) proxy-based approaches, and explained the features of each approach. I also explained why the proxy-based approach is the best in order to reuse the existing content and support normal Web browsers without any modifications or plug-in installations, and our system is accordingly designed using the proxy-based approach.

Some proxy-based systems have already been developed[6),22)]. Java applets are used to synchronize the URLs among the shared browsers[6)]. However, other operations, such as form input and scrolling, cannot be shared, because Java applets cannot detect such operations on Web pages. Our system embeds not only Java applets but also JavaScript programs that extract events from objects in a Web page and directly control the objects by using the DOM interface. Therefore, our system can synchronize not only URL transitions but also form input, scrolling, and window operations. In addition, our system also provides telepointers within Web pages, and allows users to directly attach text, images, and ink annotations to Web pages.

This paper also discusses proxy configuration. Reverse proxies are appropriate for supporting collaborative work through firewalls and with the SSL protocol. The private reverse proxy is the best for building a collaborative environment for call center agents to use in supporting end users.

When font configurations are different among the shared browsers, each browser displays Web pages in different layouts. This paper describes a method for dynamically synchronizing the page layout of a call center agent's browser with an end user's browser. The method allows end users to see Web pages according to their font preferences.

Our system allows developers to add collaborative functions to existing Web applications with minimal efforts. The system is acceptable for novice users, because the users do not have to install any software in their clients in advance in order to use the collaborative functions. They can use the collaborative function only when they need help in their browsing. Future work will involve supporting not only PC users but also other devices such as PDAs and cellular phones.

Acknowledgments The author would like to thank Dr. Toshio Souya for his implementation of the embedding engine, and Younosuke Furui for his implementation of part of the session manager.

References

1) Aoki, Y., Ando, F. and Nakajima, A.: Web

Operation Recorder and Player, *Proc. Intl. Conf. on Parallel and Distributed Systems* (*IEEE ICPADS2000*), pp.501–508 (2000).

2) Aoki, Y., Ando, F. and Nakajima, A.: Creating Web-based Presentations by Demonstration, *IPSJ Journal*, Vol.42, No.2, pp.155–165 (2001).

3) Barrett, R., Maglio, P.P. and Kellem, D.C.: How to Personalize the Web, *Proc. CHI'97*, pp.75–82 (1997).

4) Barrett, R. and Maglio, P.P.: Intermediaries: New Places for Producing and Manipulating Web Content, *Proc. 7th Intl. World Wide Web Conf.* (1998).

5) Brandenburg, J., Byerly, B., Dobridge, T., Lin, J., Rajan, D. and Roscoe, T.: Artefact: A Framework for Low-overhead Web-based Collaborative Systems, *Proc. CSCW'98*, pp.189–196 (1998).

6) Cabri, G., Leonardi, L. and Zambonelli, F.: Supporting Cooperative WWW Browsing: A Proxy-based Approach, *Proc. 7th Euromicro Workshop on Parallel and Distributed Processing* (*PDP'99*), pp.138–145 (1999).

7) 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).

8) Flanagan, D.: *Java in a Nutshell: A Desktop Quick Reference*, O'Reilly & Associates, MA (1996).

9) Frier, A.O., Karlton, P. and Kocher, P.C.: *The SSL Protocol Version 3.0*, Netscape Communications Corp. (1996). Available at http://home.netscape.com/eng/ ssl3/draft302.txt.

10) Fuentes, L. and Troya, J.M.: A Java Framework for Web-based Multimedia and Collaborative Applications, *IEEE Internet Computing*, pp.55–64 (1999).

11) Gall, U. and Hauck, F.J.: Promondia: A Java-based Framework for Real-time Group Communication in the Web, *Proc. 6th Intl. World Wide Web Conf.* (1997).

12) Garfinkel, S. and Spafford, G.: *Web Security & Commerce*, O'Reilly & Associates, MA (1997).

13) Goodman, D.: *Dynamic Html: The Definitive Reference*, O'Reilly & Associates, MA (1998).

14) Graham, I.S.: *HTML Stylesheet Sourcebook*, John Wiley & Sons, NY (1997).

15) Graham, T.C.N.: GroupSpace: Integrating Synchronous Groupware and the World Wide Web, *Proc. IFIP TC.13 Intl. Conf. on Human-Computer Interaction* (*INTERACT'97*), pp.547–554 (1997).

16) Greenberg, S. and Roseman, M.: GroupWeb: A WWW Browser as Real Time Groupware, *Companion Proc. CHI '96*, pp.271–272 (1996).

17) Greenberg, S., Gutwin, C. and Roseman, M.: Semantic Telepointers for Groupware, *Proc. Sixth Australian Conf. on Computer-Human Interaction* (*OzCHI'96*), pp.24–27 (1996).

18) Greenberg, S.: Collaborative Interfaces for the Web, Forsythe, C., Grose, E. and Ratner, J. (eds.), *Human Factors and Web Development*, pp.241–254, LEA Press (1997).

19) Gundavaram, S.: *CGI Programming on the World Wide Web*, O'Reilly & Associates, MA (1996).

20) Gutwin, C. and Greenberg, S.: Design for Individuals, Design for Groups: Tradeoffs Between Power and Workspace Awareness, *Proc. CSCW'98*, pp.207–216 (1998).

21) Hunter, J., Crawford, W. and Ferguson, P. (eds.): *Java Servlet Programming*, O'Reilly & Associates, MA (1998).

22) Jacobs, S., Gebhardt, M., Kethers, S. and Rzasa, W.: Filling HTML Forms Simultaneously: CoWeb—Architecture and Functionality, *Proc. 5th WWW Conf.* (1996).

23) 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).

24) Lee, J.H., Prakash, A., Jaeger, T. and Wu, G.: Supporting Multi-user, Multi-applet Workspaces in CBE, *Proc. CSCW'96*, pp.344–353 (1996).

25) Marsic, I. and Dorohonceanu, B.: An Application Framework for Synchronous Collaboration Using Java Beans, *Proc. Hawaii Intl. Conf. on System Sciences* (*HICSS-32*) (1999).

26) McKinley, P.K., Barrios, R.R. and Malenfant, A.M.: Design and Performance Evaluation of a Java-based Multimedia Browser Tool, *Proc. 19th IEEE Intl. Conf. on Distributed Computing Systems*, pp.314–322 (1999).

27) McKinley, P.K., Malenfant, A.M. and Arango, J.M.: Pavilion: A Middleware Framework for Collaborative Web-based Applications, *Proc. ACM Group'99*, pp.179–188 (1999).

28) 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).

29) Moatti, Y., Orell, D., Rochwerger, B., Shuklin, G. and Wecker, A.: *Remote AWT for Java.* Available at http://www.alphaworks.ibm.com/tech/remoteawtforjava.

30) Netscape Communications Corp.: *Netscape's DDE Implementation.* Available at http://

developer.netscape.com/docs/manuals/communicator/DDE/index.htm.

31) Roseman, M. and Greenberg, S.: Building Real Time Groupware with GroupKit, A Groupware Toolkit, *ACM Trans. Computer Human Interaction*, Vol.3, Issue 1, pp.66–106 (1996).

32) 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).

33) Shinozaki, M., Kobayashi, M. and Sakairi, T.: A Web-based Application Framework of Synchronous Collaboration, *Proc. Intl. Conf. on Computer Communication* (1999).

34) Shirmohammadi, S., Oliveira, J.C. and Georganas, N.D.: Applet-based Telecollaboration: A Network-centric Approach, *IEEE Multimedia Magazine*, Vol.5, No.2, pp.64–73 (1998).

35) Tang, J.: Findings from Observational Studies of Collaborative Work, *Intl. Journal of Man Machine Studies*, Vol.34, No.2, pp.143–160, Academic Press (1991).

36) Trevor, J., Koch, T. and Woetzel, G.: MetaWeb: Bringing Synchronous Groupware to the World Wide Web, *Proc. ECSCW'97* (1997).

37) W3C (World Wide Web Consortium): *Document Object Model*. Available at http://www.w-3.org/DOM/.

38) Wolf, C.G., Lee, A., Touma, M. and Daijavad, S.: A Case Study in the Development of Collaborative Customer Care: Concept and Solution, *Proc. IFIP TC.13 Intl. Conf. on Human-Computer Interaction* (*INTERACT'99*), pp.54–61 (1999).

39) Zahavi, R. and Linthicum, D.S.: *Enterprise Application Integration with CORBA Component and Web-based Solutions*, John Willey & Sons, NY (1999).

(Received May 23, 2001)
(Accepted November 14, 2001)

Yoshinori Aoki received the B.E. and M.E. degrees from Kyushu University, Fukuoka, Japan, in 1995 and 1997. 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 and the IEEE.