MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Instant Co-Browsing Lightweight Real-Time Collaborative Web Browsing

Alan W. Esenther

TR2002-19 May 2002

Abstract

A lightweight collaborative web browsing system, that targets casual (non-technical) web users, is presented. This system allows remote participants to easily synchronize pointing, scrolling and browsing of uploaded content in their web browsers. Since instant messenging systems have become a very popular method for remote participants to engage in real-time text chat sessions, it is conjectured that this simple co-browsing system which allows remote participants to share and point at their pictures and web content (instead of just sharing text) could prove useful and fun, too. The collaboratively viewed web content could either pre-exist on a host web server or, in a more typical scenario, be dynamically uploaded by the remote participants themselves. A specific goal of this system is to keep the interactions extremely simple and safe. Any user should be able to use it intuitively with a single click; there are no pre-installation or pre-registration requirements. This system is based on simple polling-based scripting techniques that avoid intrusive mechanisms based on proxies or events. Most significantly, there is no reliance on any controls, applets, plug-ins or binary executables, since these would require the trust of participants and are virus-prone. It is the reliance upon such inconvenient helper programs, along with any pre-installation or pre-registration requirements, that makes existing co-browsing offerings more ”heavyweight”, and limits their appeal for casual collaboration.

WWW2002 Conference Proceedings

Instant Co-Browsing: Lightweight Real-Time Collaborative Web Browsing

Alan W. Esenther
Mitsubishi Electric Research Laboratories
201 Broadway
Cambridge, MA 02139 USA
esenther@merl.com

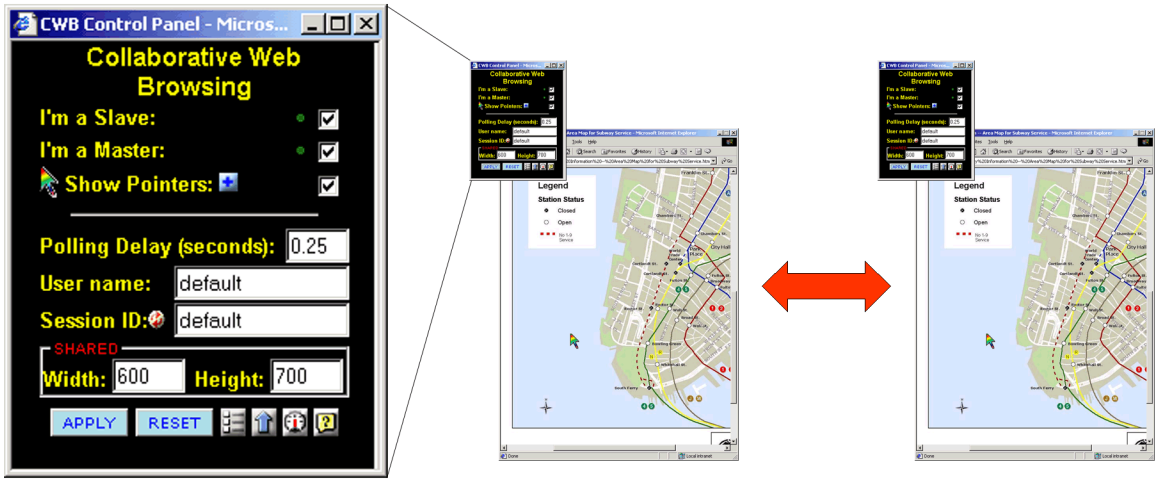

Figure 1: A representation of a CWB collaborative web browsing session. On the left is a close-up of the CWB Control Panel. This is a pop-under window that users never actually have to see. As any user in the session browses, scrolls, points, or fills in a form in any shared collaboration window (the windows displaying a map image above), the changes occur in everyone else's browser, too.

ABSTRACT

A lightweight collaborative web browsing system, that targets casual (non-technical) web users, is presented. This system allows remote participants to easily synchronize pointing, scrolling and browsing of uploaded content in their web browsers. Since instant messenging systems have become a very popular method for remote participants to engage in real-time text chat sessions, it is conjectured that this simple co-browsing system which allows remote participants to share and point at their pictures and web content (instead of just sharing text) could prove useful and fun, too. The collaboratively viewed web content could either pre-exist on a host web server or, in a more typical scenario, be dynamically uploaded by the remote participants themselves. A specific goal of this system is to keep the interactions extremely simple and safe. Any user should be able to use it intuitively with a single click; there are no pre-installation or pre-registration requirements. This system is based on simple polling-based scripting techniques that avoid intrusive mechanisms based on proxies or events. Most significantly, there is no reliance on any controls, applets, plug-ins or binary executables, since these would require the trust of participants and are virus-prone. It is the reliance upon such inconvenient helper programs, along with any pre-installation or pre-registration requirements, that makes existing co-browsing offerings more “heavyweight”, and limits their appeal for casual collaboration.

Keywords

Web collaboration, synchronized browsing, real-time distributed collaboration, shared web browsing, instant messenging, co-browsing

Word count

Approximately 7500 words

1. INTRODUCTION

Real-time web collaboration software solves the problem of how to allow multiple users to synchronize their views of web pages. For example, if any of the users in a collaborative session browses to a new page, scrolls a page, drags a shared pointer, or types in a form input field on a page, then all of the other users will simultaneously see that change in their browser windows, too (see Figure 1).

Typically the users would be talking on the telephone while collaborating, although some form of Internet telephony might be used instead. This paper presents a lightweight approach to collaborative web browsing, herein referred to as CWB (Collaborative Web Browsing). CWB does not require any virus-prone binary

programs, applets, controls, or browser plug-ins. It is based on a polling architecture and Dynamic HTML (HTML, CSS and Javascript) [1]. It is intended to facilitate spontaneous collaboration between arbitrary users, from arbitrary locations on the Internet, using web browsers on arbitrary computers. As such, it can be applied as a means for "instant co-browsing" - allowing users to collaboratively browse instantly. As with instant messenging systems (such as AOL Instant Messenger [2], ICQ [3], MSN Messenger [4], and Yahoo! Messenger [5]), CWB is intended for "casual" collaboration -- safe, immediate sharing of user content between arbitrary users (perhaps strangers). In some sense CWB is more "instant" than instant messenging applications because no pre-installation or helper program download is necessary to co-browse via CWB. Note that one requirement incurred by the exclusive use of Javascript (without any applets or controls) is that only content that resides on the same web server as the CWB scripts can be co-browsed. CWB is not designed for co-browsing to arbitrary websites. Rather, inspired by instant messenging systems, CWB is designed to facilitate spontaneous interaction with content that is dynamically uploaded by session participants. In some sense this is an effort to push the limits of the co-browsing experience that can be provided simply by fetching a plain HTML web page. The user is never prompted to grant permissions to an ActiveX control, a Java applet, or an installer for a binary executable -- yet the user can co-browse immediately. Also note that this approach is intended for sharing web page content and should not be confused with distributed whiteboard applications that allow sharing of scribbles, text, and pictures only on a special (non-HTML-based) electronic canvas. This paper presents CWB along four dimensions: collaboration features, user interface, page non-interference mechanism, and security. The design philosophy for each of these dimensions can be briefly summarized as follows:

Collaboration Features:
: "1-click collaboration" that replicates every detail of the shared web page, and adds shared pointer support.

User Interface:
: The best UI is no UI -- a simple pop-under control panel is available, but no user should have to see or interact with it.

Page non-interference mechanism:
: Only read - don't modify -- the shared web pages.

Security:
: Don't use any (virus-prone) helper programs (e.g. controls, applets, plug-ins, etc.).

2. EXISTING SOLUTIONS

Existing web collaboration solutions may be broadly classified in terms of how "lightweight" they are. A more heavyweight solution has more demanding requirements that must be met before collaboration may begin (e.g. software pre-installation, user pre-registration).

Heavyweight solutions such as Microsoft NetMeeting [6] rely on operating system support to recreate the user's entire screen (or just one or more open windows) across the network. This insures that all users are seeing precisely the same view of the web browser (or of any application). However, while appropriate in many cases, this solution may not be ideal when the goal is spontaneous casual collaboration from anywhere. NetMeeting is a proprietary solution that only runs on the Windows platform. It requires each user to pre-install a binary application on their computer, and to pre-register before collaboration may begin.

Another class of heavyweight solutions, such as the Albatross system [7] and GroupWeb [8] embed synchronization mechanisms directly into the WWW client. This approach, however, obviously adds the requirement that each user use the special WWW client. Rather than requiring a whole new WWW client, it is also possible to add a plug-in to an existing standard browser to help coordinate synchronizations. This was the approach used for a web collaboration home banking system [9]. A plug-in can also be considered heavyweight, though, since it requires pre-installation and browser modification. A plug-in also increases the disk and memory footprint of the browser just so that it will work with sites designed to use that particular plug-in. Certainly there are cases, though, where these heavyweight approaches are acceptable and suitable.

Even a lighter-weight solution, such as that provided by Hipbone [10], downloads a Java applet into the user's web browser. A web collaboration banking kiosk system (also described in [9]) and the Artefact framework [11] are additional examples of applet-based solutions. This approach obviously requires Java support on the client computer, and it adds delays while the Java environment is loaded. Also note that the Hipbone product, as well as a similar offering by WebDialogs [12], is targeted at customer support centers. As is typical for this type of product, the collaborative session is coordinated via a binary executable client program that is installed and run on a customer service representative's computer. This may not be suitable for casual users who wish to collaborate quickly and independently, without the requirement that one of them runs a coordination program. In this respect, collaboration via CWB is perhaps analogous to instant messaging: if two instant messenger users wish to exchange text messages, there is no need for a third party (a customer service representative) to be present in order for the session to proceed.

Another common requirement that is prevalent with existing solutions is that users have to take turns making changes. Typically the user interface has some type of control to specify which user is currently allowed to make changes (such as browsing to a new page or scrolling the page). A more flexible solution would allow any user to make a change at any time, rather than first having to request the privilege. If the users have to pass some token to allow them to make changes, then their attention must be diverted from the shared web page to the token control, which distracts them from the collaborative session.

An even lighter-weight solution (used by CWB) could use just Dynamic HTML scripts to peek into the shared web page and extract and apply changes. A patent by IBM [13] explores this idea, although it relies on a Java applet in the client to coordinate updates between the client web browser and the web server. That solution may be characterized as "intrusive", however, because it modifies the contents of the shared web page by placing it into a special layer. This is done so that layers with pointers and other collaboration elements can be placed above the shared web page. However, this technique can break the behavior of scripts in the target web page. Such scripts might not expect that the containing document has moved to a new layer.

An older mechanism, also mentioned in the IBM patent, uses a proxy approach. Before a web page is sent to the client, a web server-based proxy program rewrites all of the URLs in the page such that they reference the collaboration web server. The CoWeb system [14] also uses a proxy approach, though it replaces HTML elements with Java applets. However, this is even more intrusive and more likely to change the behavior of the existing web page, partly because it is no trivial task to detect all of the myriad ways in which the HTML and scripts in an arbitrary shared web page might create external references. Pages that use scripts to dynamically create new external references and HTML elements are particularly problematic for these systems.

Another technique for monitoring changes within a shared web page is to use event handlers. A script inserts or chains it's own event handler into various elements of the web page, essentially creating a callback to the monitoring script. (The WebVCR system [15] uses this technique to send information about user actions to a companion applet rather than a script.) As an example, the monitoring script might want to intercept scroll events so that it will know when to replicate scrollbar changes to other users. This technique can also be considered intrusive and may change or break the behavior of the web page. For example, the callback may execute in response to an event before returning control to the shared page's event handler. However, the shared page's handler might abort the event, or trigger some change that might not be detected and thus might not be replicated to other users. A script in a shared page might also dynamically create a new page element that the monitoring script will not know about.

Finally, existing solutions may not support all of the fine-grained collaboration features discussed later.

3. HIGH-LEVEL ARCHITECTURE

Most of the novelty of the CWB solution is embodied in the design philosophy used for each of four dimensions of web collaboration, as is described in subsequent sections. The high-level architecture is described in this section.

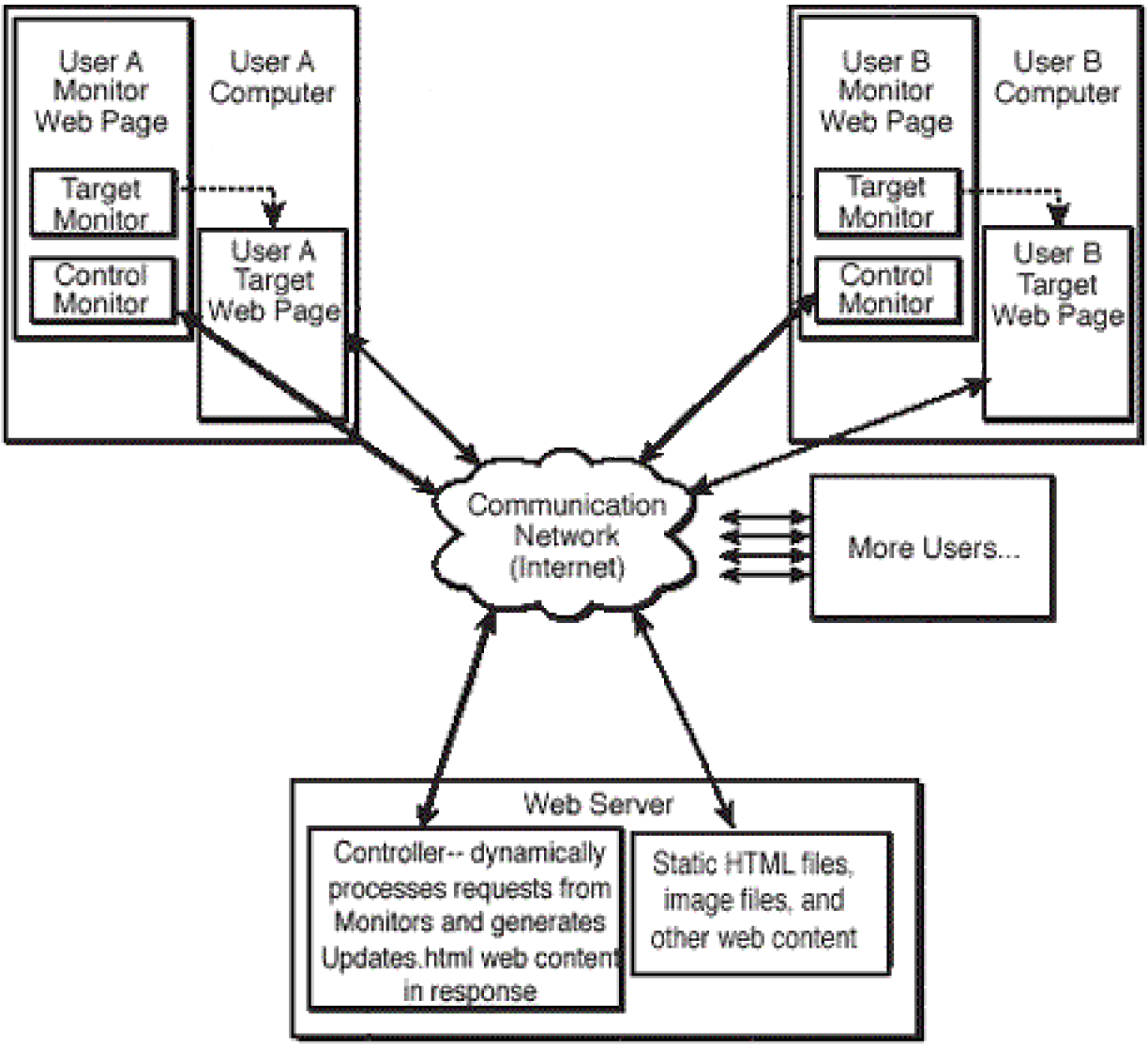

Figure 2: CWB Architecture Diagram

In the CWB architecture, each user's computer has two instances of the web browser running. (Alternatively, it could be implemented with multiple frames in a single instance, but that would require the shared web page to be placed into a frame, potentially interfering with scripts on the page.) One instance contains the target web page that is being collaboratively viewed. It is worth noting that this instance is just a regular browser window that can be used as usual even if there is no collaborative session underway. The other window contains a control panel and two monitor routines written in Javascript. The first monitor routine, the Target Monitor, periodically analyzes the browser instance containing the target web page, to see if any changes have occurred since that last time it was analyzed. If a change is detected, such as if the user has scrolled a scrollbar, then that change is transmitted as an update to a Controller program running on the host web server. Note that each update contains the entire state of the system so that users can join the session at any time and still become fully synchronized. The second monitor routine, the Controller Monitor, contains a routine that periodically polls the Controller program on the web server for changes made by other users. If any changes are detected, then they are applied to the window containing the shared web page via standard Javascript mechanisms.

To avoid browser compatibility issues, communications between the Controller program on the web server and each client web browser are implemented as simple page fetches to a hidden frame. Either HTTP GET or POST requests can be used, generally depending on the size of the message being transmitted to the Controller. This method of communication can work through firewalls since all requests are originated by the client browser and work over the standard HTTP port. Alternatively, for added security, HTTPS could be used to encrypt all communications in the session.

Each user may be configured to run as a master, as a slave, as both, or as neither. This configuration can be changed "on the fly" (dynamically). A master user is able to make changes that will be seen by other users. A slave user is able to receive changes that have been made by any of the master users. Typically, and by default, each user is configured as both a master and as a slave. The master routines are implemented in the Target Monitor, and the slave routines are implemented in the Controller Monitor. It is anticipated that users may wish to temporarily browse independently (and privately) for a while, and then to rejoin the collaborative session in progress later. Support is also included for allowing new users to join an existing session in progress at any time.

Note that one current restriction of this architecture is that only web page content which is stored on the local host web server (that contains the Controller program) can be collaboratively viewed. This is due to important security restrictions that are built into web browsers that prevent a script in one window from peeking into another browser window containing a web page that has been retrieved from a different domain. It would be relatively straightforward to add a control or an applet that (if explicitly permitted by the user) peeks into other browser windows. Then co-browsing of arbitrary domains could be allowed. This would invite viruses, though, and perhaps delay the onset of collaboration, and may be beyond the scope of the lightweight approach to collaboration for which CWB is designed. For now, the light weight of CWB is considered more important for the types of collaboration envisioned than the ability to co-browse arbitrary domains.

Currently, the Controller program is implemented as a Java servlet [16]. Session status, configuration, and file upload web pages are also implemented as interfaces to the servlet. The servlet is easily deployed as a simple Java servlet WAR (Web Application Archive) file on any web server running an appropriate servlet engine. This requires Java support on the web server, but not on the client machines. It should be straightforward to implement the Controller in an alternative language, such as PERL.

4. DESIGN PHILOSOPHY

This paper presents CWB along four dimensions: collaboration features, user interface, page non-interference mechanism, and security. The design philosophy for each of these dimensions is discussed in this section.

4.1 COLLABORATION FEATURES

Collaboration features are features of the web collaboration implementation that facilitate an enhanced collaboration experience. The design philosophy for this dimension of web collaboration was to provide "1-click collaboration", to replicate every detail of the shared web page, and to add pointer support.

The system provides "1-click collaboration" in that even first-time users need only to click a button on a Welcome page (or to execute a bookmark/Favorite) to start collaborating in a default session. Clicking the button opens the Control Panel and then the shared collaboration browser window, and immediately synchronizes the user with the collaborative session. The session is created on-the-fly if necessary. After the "1-click", the user is a member of the collaborative session, so if the user scrolls (for example) then other members of the session will see the scrolling action, too.

Early web collaboration efforts suffered because they only shared the URL of the page being shared between participants. If one user scrolled down a ways, then that user would (perhaps unknowingly) be looking at different content than other users. If a page used frames, then the pages loaded into each frame were not synchronized because changing frame sources does not change the top-level URL of the web page loaded into a web browser. CWB strives to synchronize all static and dynamic attributes of the shared window and frames. However, since CWB uses a lightweight DHTML approach, in general it will not be able to synchronize elements on a page that rely on applets, controls or plug-ins (such as audio or video clips).

CWB synchronizes attributes such as the browser window size, all window and frame web page sources, and all scroll bar positions. It also synchronizes visual form element content (e.g. text fields, checkboxes, select lists), including character-by-character entry into text fields, and scroll positions of textareas. Since any user can make changes at any time, users can jointly fill in forms. Currently, synchronization of user text selections (typically conducted by left-clicking and dragging over text content) is partially supported.

Although shared text selections are very helpful in clarifying which section of a shared page a user is talking about, it has been found that also having a shared (distributed) pointer is invaluable for enhancing a browser session. For example, with shared pointers each user can point to a particular location in an image that he or she would like to discuss. Any master can subsequently move any pointer simply by clicking and dragging. In the current implementation, by default a shared pointer is automatically created in the upper left corner of any frame that is large enough to hold a pointer image, as shown on the left in Figure 3.

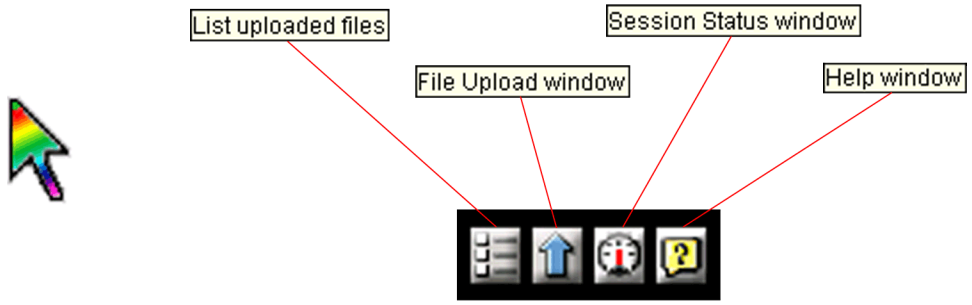

Figure 3: Interface details. Left: A shared pointer, which can be dragged by any user at any time. Right: Control Panel button details. Integrated file upload support includes automatically un-zipping uploaded .zip files.

This default would likely be configurable in a commercial environment, but ad hoc testing has shown that it is convenient to have a ready-made pointer available for dragging. Users can move or recreate a pointer by simply ALT-clicking at the location at which to create the pointer. (For a Macintosh version, a different key would be used.) CTRL-ALT-clicking deletes a pointer. Pointer visibility can also be controlled from the control panel, as is explained below. Significantly, there is no need to refer to a control panel before creating a pointer. Requiring so would inconveniently force users to divert their attention from the window containing the shared web page. Similarly, any master user can create or move a pointer at any time. There is no need to "take turns" or request permission to move a pointer (or to make any other change), as is also explained below.

4.2 USER INTERFACE

CWB is intended to allow casual (quick and lightweight) collaboration between non-technical users. Therefore the design of the user interface is simple and unobtrusive. The design philosophy was that the best user interface is no user interface. The existence of this shared pointer itself indicates collaboration. Therefore, for typical collaboration there is no new or special user interface to learn, so users aren't forced to divert their attention away from the content they are discussing.

Collaboration is initiated in a client browser simply by typing an optional username and an optional session ID into a standard HTML form on a "welcome" page, and then clicking a submit button.

This opens both the control panel window, shown on the left in Figure 1, and a new shared collaboration browser window in which any shared pages will be viewed. In Figure 1, the two browser windows containing map images represent screenshots of shared collaboration windows on two different systems.

There is no delay associated with downloading or starting applets or controls, and users do not need to pre-register in order to participate. Users are uniquely identified by the session ID, user name and their IP address. This implies that the same user can be involved in multiple sessions at the same time with the same user name, and that a user can participate in the same session with multiple usernames. The latter feature is occasionally useful both for debugging on one computer screen, and so that a user can visually verify what his or her changes look like to other users. Significantly, by default the user does not have to specify either a user name or a session ID. Thus, initiating collaboration is as simple as clicking one button (or invoking a browser Bookmark/Favorite).

CWB supports multiple parallel and independent collaboration sessions. Therefore users talking on the telephone can define and start a new session simply by entering an arbitrary unique session ID string (an alphanumeric string). Since each update contains information about the entire session state, a user can join late, or leave a session to browse independently for a while and then re-join the session later. The user name is provided only to more easily identify a particular user in the Session Status page, shown in Figure 4, which is available via the control panel. For example, by examining the Session Status page, one user can see that another user has not received the latest updates, and also notice that this is because that user has, perhaps inadvertently, turned off slave updates. If the user with this problem has typed in a user name, then it will be easier for the first user to identify which user has the problem and to tell him or her about it.

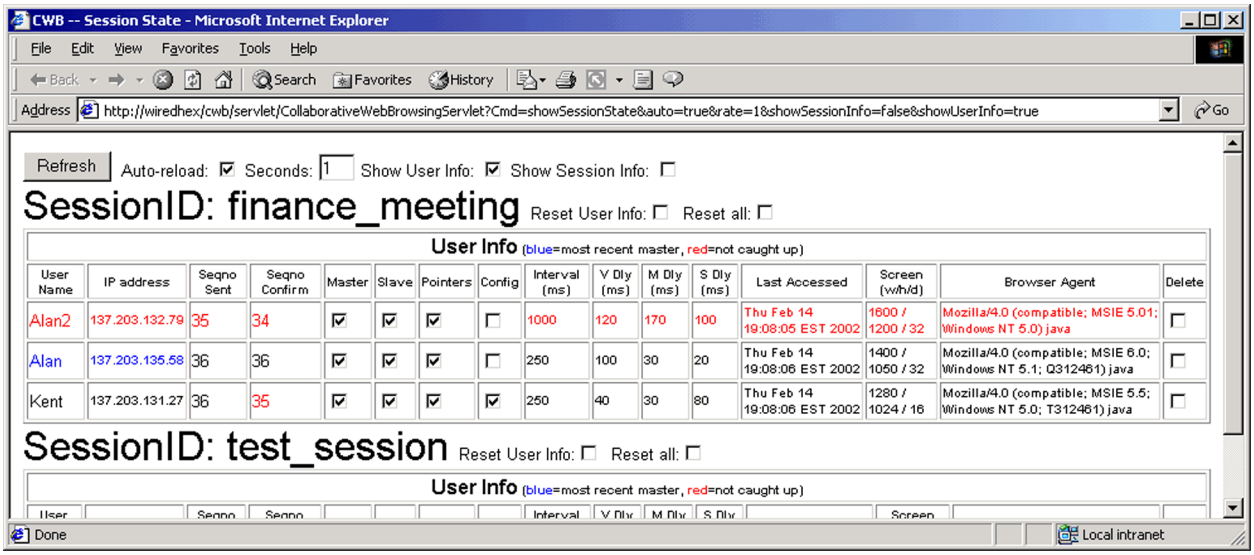

Figure 4: CWB Session Status window. Contains details about each user and session.

To keep things simple for non-technical users, the session defaults to enabling all users as both a master and a slave. By default, each user will be connected to the session as a slave first, and then, immediately, as a master -- so as not to inadvertently modify the session that they are joining. If the user wishes to change this behavior, or any other parameters, he or she can use the control panel window. Otherwise, there is no reason for any user to ever even be aware of the existence of the control panel. As was mentioned earlier, by default a pointer is created in the upper left corner of each frame, ready for dragging. This serves as a visual indicator that the page is part of a collaborative session. Also, any user can toggle the local visibility of all pointers via a checkbox in the control panel. A user might want to hide the shared pointers in this way if the pointers are cluttering up the browser window, or are inconveniently obscuring page content.

Another aspect of the user interface, present in both the distributed pointers and in the control panel, is a visual feedback indicator to show the user when all views are synchronized. The very tip of each distributed pointer alternates (with subtlety) between red and green if all users are synchronized, and between red and yellow if one or more users are not caught up, yet. In the control panel, indicators in front of the Slave and Master checkboxes flash green or red to indicate synchronized and not synchronized, respectively.

The user interface supports multiple simultaneous masters without requiring that a master first request and be granted permission to make a change. If a master user would like to make a change, such as browsing to a new page or scrolling down, he or she just makes the change. Although the semantics for resolving simultaneous change requests could be configurable, ad hoc testing has revealed a simple algorithm for honoring master updates.

If a master is configured as both a master and a slave, then the policy is that a request from a master is not honored unless that master is currently up-to-date. Each master update request increments a sequence number that is included in all messages between clients and the web server. Ignoring updates from masters that are not caught up assures that a master that is several updates behind (perhaps due to network congestion) will not suddenly apply a change that will force all slaves to suddenly see an old view of the session. Note that if one master, say, drags a pointer to the left, and another master drags that same pointer to the right at the same time, then the first update to reach the web server will be honored. This means that the "slower" master will see the pointer suddenly jump to a different position. This may be an annoyance, but it should be obvious to the slower master that someone else has made a change at the same time. This minor annoyance far outweighs the hassle of requiring users to regularly divert their attention away from the session (to another panel) to request or give up permission to make changes. For casual collaborative sessions in which all users are conversing on the telephone, ad hoc testing has shown that this simple conflict resolution scheme works extremely well.

If a master is configured as a master, but not as a slave, then it is presumed that that master (a teacher, perhaps) does not care what anyone else is doing. In this case, by default, the master updates would always be accepted and applied. If multiple users were configured only as a master, then slaves could see the shared web page sources and attributes wildly changing back and forth if masters on different pages were to continually transmit unrelated updates. However, for casual collaboration in which users are talking on the telephone, it is assumed that such situations would be quickly and simply resolved.

Since the CWB system is designed for collaborating on dynamically uploaded web content, it contains integrated file upload support. As shown on the right in Figure 3, the Control Panel contains buttons to open a File Upload window, and to immediately co-browse to a directory listing of all files that have been uploaded so far for the current session. Each collaborative session has an associated "uploaded files" directory.

In summary, the idea behind the CWB user interface is that by default there is no new or special user interface for any user to learn. There is nothing special about the browser window containing the shared web page, so users are free to interact with it just as if they were not collaborating. The only difference is that scripts in

another window are peeking into that window, too, and that a shared pointer may be overlaid onto the shared page or frames.

4.3 PAGE NON-INTERFERENCE MECHANISM

An important aspect of web collaboration solutions is the extent to which they can be used to collaborate on arbitrary web pages. It would be less than ideal if users could only collaborate on pages that had been prepared by specially trained web authors, or pages that required special tags to be dynamically inserted, or pages that had restricted characteristics. The design philosophy for CWB was that, in order to insure the integrity of the shared page, CWB should only read - not modify -- the shared web pages. CWB achieves this by regularly polling the entire state of the shared page rather than by modifying or inserting code to notify CWB of changes. (The applet in the WebVCR system [15] also used a polling mechanism, though only to ascertain whether a page had finished loading.) Years ago such polling might have been too CPU-intensive for a typical user's computer, but this is much less of an issue today. In fact, it might be argued that CPUs today are far more powerful than is necessary for typical end-user tasks such as word processing and handling email, so a solution that leverages this power is appropriate. Furthermore, during a live collaboration session the users will most likely be devoting their attention and CPU to the collaboration session rather than to some other CPU-intensive application. Of more concern, perhaps, are the network delays associated with polling the Controller on the web server for updates from other users. This is independent of the CWB system itself, though, and can be improved using standard methods to improve web access (faster web server, faster network connection, etc.).

As discussed earlier, one technique to detect user actions is to have the system insert its own event handlers into the shared web page. For example, the "onchange" event handler for every text field would have a handler added to it when the page is first loaded so that any text field changes would trigger a routine in the collaboration software. However, this technique can interfere with the behavior of existing event handlers. For example, an existing handler for such an "onchange" event might abort the change, or even execute a script that dynamically creates a new form element that might not be monitored. CWB makes no assumptions about the current state and objects on a page. Rather, CWB completely analyzes the entire browser window regularly. In other words, CWB polls the static and dynamic state of the page to see if something has changed since the last time it was polled. Any changes are transmitted to other users.

As was also discussed earlier, some earlier web collaboration solutions relied upon proxy systems. A proxy system rewrites all of the external references (URLs) in the target web page (to reference the collaboration web server) before sending the page to the client. This is done so that when a user clicks on a link, the collaboration server will know about it and be able to notify other users about the action. URL re-writing can break the intended behavior of scripts on the page, though. Furthermore, it is difficult to detect all of the obscure methods that a page might use to create a reference to other web content. For example, mouse rollover images are specified using scripts, not HTML, so parsing the HTML is not sufficient to detect the image references. Also note that a URL that was re-written by a proxy system might subsequently be over-written by a script in the shared web page. Currently CWB largely avoids the need for URL rewriting by restricting shared pages to those residing on the collaboration web server. This restriction is necessary to avoid security measures built into browsers that prevent scripts from peeking into web pages from different domains. For lightweight collaboration, the ability to co-browse to foreign domains is not considered critical, though this is an area of continuing investigation. Nonetheless, by polling the current state of the web page, CWB can non-intrusively detect changes as they occur, regardless of what caused them to occur. (For performance reasons, CWB does not currently share mouse rollovers or, more generally, changes to the inner HTML of shared pages, though the CWB architecture makes it trivial to add this functionality.)

It must be noted that there is one case in which CWB fails to meet the design goal of never modifying the existing shared page. In order to implement a shared pointer, CWB must add a layer containing a pointer image (on a transparent background) above the existing layers in the shared web page. This modifies the shared web page in a number of ways. It adds one new layer and one new image for each pointer, and it adds an "onclick" event handler to the body element in each shared frame. If the existing frame already has a custom "onclick" event handler, then CWB will execute it's own handler first, and then the page's custom "onclick" handler. In this way, the effect of CWB's modification to the page is minimized. However, this is still a clear violation of the "don't modify" rule, so it may be best to make shared pointing a dynamically configurable capability of the CWB system.

Incidentally, another advantage of the polling approach (and an example of it's resiliency) is that if a script on the shared page overwrites the CWB pointer event handler, it is okay because CWB will detect this and re-chain it's event handler back in during the next poll. In general, during development it was found that if some sort of (usually timing-related) scripting error occurred it could just be dismissed and ignored because CWB would automatically recover on the next poll. More recent browsers contain versions of Javascript with support for structured exception handling, so that timing-related errors can automatically, and thus transparently, be caught and ignored.

4.4 SECURITY

Another aspect of collaborative web browsing systems is security. Since CWB is intended for casual, spontaneous collaboration, the design philosophy is that it should not use any (virus-prone) helper programs (e.g. applets, controls, plug-ins, etc.). This lack of helper programs is what defines CWB as a lightweight solution, and it is what makes it safer than other solutions. Any time a user installs a binary program, installs a browser plug-in, or grants privileges to an applet or a control the user is allowing that program critical access to his or her computer. Since CWB is intended for "casual" collaboration (perhaps with complete strangers), it is highly undesirable to ask the user to take such risks. Even if CWB is used for collaboration between trusted parties, users may be averse to granting powerful privileges to a helper program if private information will be handled by it. CWB also avoids using helper programs because of the undesirable pre-collaboration requirements (software installation / configuration / maintenance, user registration, download delays). Figure 5 highlights the fact that, compared to other co-browsing approaches, CWB is safer because it does not bypass browser security restrictions.

Approach:	Polling & DHTML	Stand-alone Binary Executable	Custom WWW Browser	Browser Plug-in	Java Applet or Control	Lightweight Client Plus Special Privileged User
Example System:	CWB	Microsoft NetMeeting	GroupWeb	Home banking system	Hipbone	WebDialogs
Disadvantages:	Minor polling delays Users can only co-browse their own uploaded content, or pre-existing content on a host web site	Requires installation of a new program on each client Pre-registration required User must trust the software	Requires installation of a new program on each client User must trust the software	Requires pre-installation Requires browser modification User must trust the software	Requires Java support on each client Delays to load Java environment User must trust the software	Requires a customer service representative (running a special binary executable) User must trust the software

Figure 5: A comparison of co-browsing approaches. Because CWB does not bypass security restrictions built into web browsers, it does not require users to trust others with potentially dangerous privileges.

If security is an overriding concern, as it may well be when collaborating about an internal project, then CWB can be used with an SSL-enabled web server to encrypt the messages sent between the browsers and web server (i.e. use the HTTPS protocol rather than the HTTP protocol). Standard browsers already support HTTPS so no change to the client browser is necessary to support secure collaboration. In fact, the choice of protocol is completely transparent to CWB so deployment is the same.

Most likely, if secure collaboration is a concern, then access to the web server will also be password-protected. Again, this is independent of CWB itself, and only requires configuration changes to the web server software. Alternatively, a password protection scheme could be integrated directly into the CWB Controller servlet.

There may be some privacy concerns since the CWB servlet is literally monitoring each client's traversals through the website. However it should be pointed out that this "monitoring" is essentially happening anyway since web server logs already contain a record of all page requests from clients.

CWB works through firewalls because all of the message transmissions are originated as simple GET or POST requests by the client browser and travel over the standard HTTP (or HTTPS) port.

CWB does not use cookies or require that cookie support be enabled in client browsers.

CWB does not use or require support for either Java applets or ActiveX controls to be enabled in client browsers.

5. IMPLEMENTATION AND PERFORMANCE ISSUES

So far, CWB has been implemented for the Internet Explorer browser versions 4 and later on Win32 platforms. Although it has not yet been adapted for use with other browsers, such as Netscape, it should be straightforward to do so since the server-side components are identical and only browser-specific client-side Javascript code needs to be tailored. The unoptimized size of the client-side control panel and scripts is currently about 100k bytes. The server-side component is a Java servlet, though an alternate language could be used. In ad hoc testing CWB has successfully been used between the east and west coast of the United States just as easily and about as well as on a corporate LAN. It has been used to collaborate on Microsoft Office documents that were saved as HTML, and was also used during technical paper presentations so that remote participants could join in. It has also been used while reading a web-based storybook to children across a cable modem to point to the current text position.

The performance of a polling system such as CWB is dependent on the polling interval, network delays, and client CPU speeds. The frequency with which masters poll for local changes and for which slaves poll for remote changes can be controlled. Specifically, a user can specify, via the control panel, the time to wait in between polls. This is the amount of time the system should wait after checking for changes (and applying them, as needed) before it should check again. The default is 0.1 seconds. (To the eye, a setting of 0.1 seconds appears just as fast as a setting of 0 seconds, but using a non-zero value decreases the CPU load

noticeably.) If all users are using the default setting, then there could be as much as a 0.2 second round-trip delay, independent of network and CPU delays, before a change is detected and sent from a master user and detected and applied to slave users. This worst-case delay would occur if a change occurred on a master just after a check for changes had occurred on that master, and if that update then arrived at the slave side just after that slave had finished checking for updates. In ad hoc testing with the default 0.1 second setting over non-dialup connections, round-trip delays of less than 1 second to propagate changes between users are typical and hardly noticeable. Slower client systems, with 200MHz and slower CPUs, may have trouble sustaining such short update intervals without delaying the next polling session, but the system will still work -- running as fast as it can. If such immediate responsiveness is not critical, a user can take the burden off of their CPU by increasing the update interval setting to 1 second or more. At longer update interval settings (such as 1 second or more) responsiveness will be degraded and the delays may be annoying, but still may be acceptable for casual collaboration wherein instantaneous synchronizations are not critical. Even on slower systems, the delays may be acceptable in order to achieve the benefits of the CWB system (e.g. safety, simplicity). It is recommended that a decent network connection (cable modem or better) be used. Although CWB works over dialup connections, the time to download web pages themselves (independent of CWB) can sometimes be long enough to deter comfortable real-time collaboration. Performance will be improved in future versions of CWB.

Incidentally, CWB has built-in mechanisms to deal with unreliable network connections. If a user's network connection goes down temporarily, CWB will automatically recover and continue without user intervention.

The scalability of the system has not yet been stressed. CWB is targeted at teleconferencing groups, the size of which do not tend to be excessively large. In the event that a group with a very large set of collaborators must be supported, standard methods to improve web access may be applied (faster web server, faster network connections, etc.).

6. DISADVANTAGES & RESTRICTIONS

CWB is intended to be a simple, lightweight provider of real-time, web-based, multi-master collaboration for arbitrary users of arbitrary HTML-based web content. As such it considers some drawbacks to be acceptable under the realm of a lightweight approach. For example, shared pointing (and scrolling) will not be precise if the pixel positions of web page elements are not identical in each user's browser. This is the basis for most of the restrictions inherent in CWB. Since different web browser brands tend to flow text and other content somewhat differently, the location of shared pointers (relative to surrounding page content) might not be precisely identical in different browser brands used in the same collaborative session. Therefore, it is generally intended that all participants in a particular collaborative session are using the same browser brand. However, if shared scrolling and pointing is not a requirement, then different browser brands may be used. For example, even if two participants are using different browser brands, then they can still share URLs, form element content, and text selections. Some degree of shared scrolling and pointing should be possible even between users of different browser brands, but this has not been investigated, yet. Also note that if scrolling and pointing are to be synchronized, then users are discouraged from changing browser attributes which modify the viewable space for web pages in the browser window (e.g. default font size and the number of toolbars). Since collaboration begins in a browser window that is created for the user by the CWB scripts, the scripts insure that each user's collaboration browser window is identical.

Along those same lines, if shared scrolling and pointing is a requirement then the browser window size of each participant must be synchronized. Since it is difficult for a Javascript routine in an unmodified browser to accurately determine the current browser window size, the window size is currently specified in the control panel rather than by resizing a window by dragging it's border. This implies that some (vocal) communication between users might have to be conducted at the beginning of a collaborative session to make sure that everyone is satisfied with the window size. With a reasonable default window size, hopefully this limitation will not be too troublesome for users. Also note that when collaboration is initiated by a user (by opening the control panel) a new browser window is opened in which to collaborate.

As has been discussed, due to security restrictions that are built into browsers, the server-side CWB component should be placed in the same domain as all web pages that are to be shared during the collaborative session. Co-browsing to arbitrary domains would be a nice feature, but many meaningful collaborations are likely confined to content on one web server anyway. This is an area of ongoing investigation.

7. CONCLUSION

Inspired by instant messenging systems, CWB provides a quick, simple and safe way for non-technical users to casually co-browse their content while avoiding the pre-collaboration and trust requirements of existing co-browsing offerings. Its' non-intrusive nature facilitates collaboration over existing web content without modification.

Future work may focus on issues such as performance, reliability, scalability, supporting new and alternate environments (browsers, operating systems, web servers, handhelds), server-side support for shared cookies and form submissions, and new collaboration features.

8. REFERENCES

1. Goodman, D. Dynamic HTML: The Definitive Reference. O'Reilly & Associates, Inc., Sebastopol CA, 1998.
2. America Online Inc., AOL Instant Messenger. Available at http://www.aol.com.
3. ICQ Inc., ICQ. Available at http://www.icq.com.

4. Microsoft Corp., MSN Messenger. Available at http://www.microsoft.com.
5. Yahoo! Inc., Yahoo! Messenger. Available at http://www.yahoo.com.
6. Microsoft Corp., NetMeeting. Available at http://www.microsoft.com.
7. Ping-Jer Yeh, Bih-Horng Chen, Ming-Chih Lai, and Shyan-Ming Yuan. "Synchronous Navigation Control for Distance Learning on the Web." Proceedings of the Fifth International World Wide Web Conference, May 1996. URL http://www5conf.inria.fr/fich_html/papers/P28/Overview.html
8. S. Greenberg, and M. Roseman, "GroupWeb: A WWW Browser as Real Time Groupware." ACM SIGCHI'96 Conference on Human Factors in Computing System, Companion Proceedings, April 1996.
9. Makota Kobayashi, Masahide Shinozaki, Takashi Sakairi, Maroun Touma, Shahrokh Daijavad, and Catherine Wolf. "Collaborative Customer Services Using Synchronous Web Browser Sharing." ACM 1998 Conference on Computer Supported Cooperative Work Proceedings, pages 99-108, Nov. 1998.
10. Synetry CoBrowse. Hipbone, Inc. Web page. http://www.hipbone.com.
11. Jeff Brandenburg, Boyce Byerly, Tom Dobridge, Jinkun Lin, Dharmaraja Rajan, and Timothy Roscoe. "Artefact: A Framework for Low-Overhead Web-Based Collaborative Systems." ACM 1998 Conference on Computer Supported Cooperative Work Proceedings, pages 189-196, Nov. 1998.
12. WebIntermeet. WebDialogs. Web page. http://www.webdialogs.com.
13. G. Pacifici, U.S. Patent No. 6,230, 171 707/512. G06F 017/21. Markup system for shared HTML documents. May 8, 2001.
14. Stephan Jacobs, Michael Gebhardt, Stefanie Kethers, and Wojtek Rzasa. "Filling HTML Forms Simultaneously: CoWeb - Architecture and Functionality." Proceedings of the Fifth International World Wide Web Conference, May 1996. URL http://www5conf.inria.fr/fich_html/papers/P43/Overview.html
15. Vinod Anupam, Juliana Freire, Bharat Kumar, and Daniel Lieuwen. "Automating Web Navigation with the WebVCR." Ninth International World Wide Web Conference Proceedings, May 2000.
16. Sun Microsystems, Inc., Java Servlet technology. Available at http://java.sun.com/products/servlet/index.html.