CoBrowser: Surfing the Web Using A Standard Browser

K. Maly, M. Zubair, and L. Li
Department of Computer Science
Old Dominion University, Norfolk, VA 23529

Abstract: Co-browsing is a synchronous class of collaborative application, which allows a group of users to "surf" the web together. Such an application can be deployed in an education environment in several ways. For example, courses that are project oriented might require students to collectively research or explore information on the Web. In this paper, we describe CoBrowser, a prototype co-browsing system that supports multiple groups, CoPointer, and Chat sessions. In past, co-browsing architecture have been based on either a proxy server or required a specialized plug-in for the web browser. The proxy server based approach is complex and is not scalable. The proposed CoBrowser addresses both these limitations and works on any platform that supports a graphical Web browser with Java capability.

1 INTRODUCTION

Co-browsing is a synchronous class of collaborative application, which allows a group of users to "surf" the web together. Such an application can be deployed in an education environment in several ways. Few examples are: (a) Courses that are project oriented might require students to collectively research or explore information on the Web. A group of students may be working together to write a report on the subject of computer security. For this, students may decide to research the material together to identify papers, which they can use for writing the report. (b) Instructor holds a recitation session with a group of students to explain or elaborate on some concepts discussed earlier in the class. (c) An instructor may find it useful to use different media, such as animation and video available in a digital library, to explain a breadth-first search algorithm.

Recently, many collaborative web applications have been developed [1-5]. Most of these approaches either require modification of the browser, are based on the X Window System protocol. However, these approaches do not lend themselves easily to a group of users working on multiple platforms (such as Unix, Windows NT, MacOS, and Windows2000) concurrently and spontaneously. Clearly, it is no technical problem to write a distributed software system that runs on all participants' machines and has the desired functionality of synchronizing their browsing. However, in today's world, people are constantly on the move working from different machines and are unwilling to go through a complex install process to join a group session spontaneously. One should add that it is a technical problem to write such a system to work on all platforms. The second approach is based on the use of proxy server that is responsible for coordinating and delivering content to all the participant browsers. This requires only minor modifications in a user's browser but the approach has turned out to be fraught with great technical difficulties that nobody we are aware of has completely solved. In [5] we describe these and our own initial proxy approach.

This paper describes an approach that works on any platform that supports a graphical Web browser with Java and JavaScript capability and is easy to implement. Most new Web browsers, such as the Netscape and Microsoft Internet Explorer, have this capability. Also, in this approach it is not necessary for every user in the group to be on the same platform. It is feasible for some users in the group to use a browser on a Unix machine, while others use a browser on Microsoft Windows. In past, co-browsing architecture have been based on either a proxy server or required a specialized plug-in for the web browser. The proxy server based approach is complex and is not scalable. The proposed CoBrowser addresses both these limitations and works on any platform that supports a

graphical Web browser with Java capability. Another advantage of this approach is that it does not modify the browser. Coordinated browsing can be integrated with audio to enable users to speak while browsing. However, this paper will only discuss the coordinated browsing architecture and its implementation.

The rest of the paper is organized as follows: Section 2 gives an overview of the architecture of the CoBrowser, Section 3 discusses the basic operation, and in Sections 4 and 5 discuss additional features. Conclusions and future work are detailed in Section 6.

2 COBROWSER ARCHITECTURAL OVERVIEW

CoBrowser is a multi-group co-browsing system. That means it allows several co-browsing groups existing simultaneously without interfering with each other. A typical co-browsing group consists of several participants who can surf the web together. CoBrowser provides the following functionality for a co-browsing group:

- When a user in the group loads a document from a site in a co-browsing session, the same documents gets loaded on all the other users' Web browsers.
- When a user resizes his browser window or drag the scroll bars in his browser window, all other users in the same group will get the same changes in their browser windows.
- If one user loads a local file into his browser (either drag and drop a file icon or use file open), all the other users in the same group will see that file in their window as well.
- When a user points to somewhere in his web page, any of other users in the same group can see that pointer
- When a user wants to talk with other users in the group, he can enter the group's chat room.

The key idea of CoBrowser is to monitor a user's activity such as opening a new URL, resizing his browser, scrolling the browser window, and etc., and then communicate this information to other users' browsers in the group. The other users' browsers on receiving this information perform the same activity such as download the same document or resize it's own window or drag the scroll bars. We use Java and JavaScript technology to implement our system. As seen in Figure 1 the CoBrowse system consists of software for a Central Service (CS), a collocated Webserver (WS) and a Co-browsing Server (COS). At the client side we have users that have JAVA enabled browsers and that form mutual exclusive (this is only a limitation of the current implementation) groups of surfers that share the same experience.

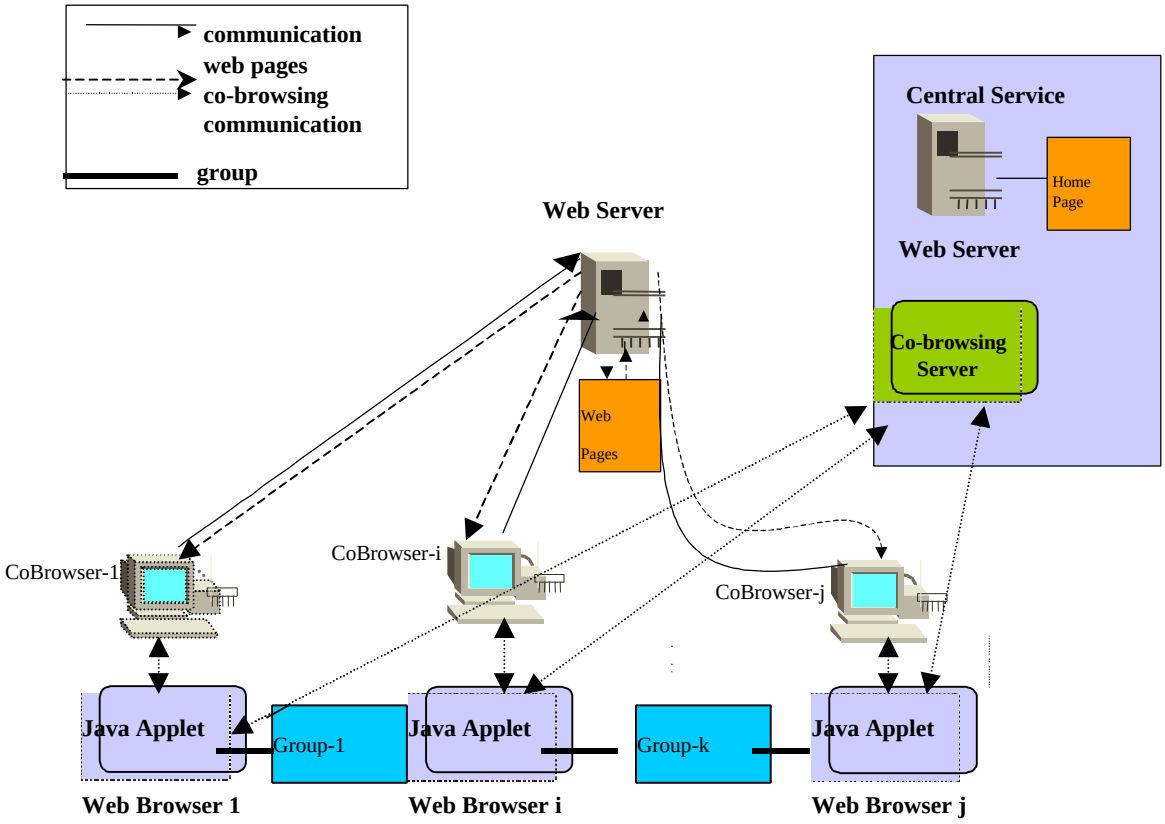

Figure 1: Multi-group architecture for co-browsing system.

3 COBROWSER BASIC OPERATION

To illustrate the working of the co-browsing architecture, we consider the following scenario where a new user joins an ongoing co-browsing session of two users. In the ensuing explanation we shall use always one group of users. Our system has been implemented to handle a number of simultaneous groups. The code to handle multiple groups is extensive and actual is the main cause of tricky bugs because we need to keep separate data structures for each group's information. However, there is no fundamental architectural design feature that is needed to go from one to two or more groups.

Registration Process. The new user downloads a registration document served by the CS-WS, see Figure 1, when the user starts cobrowsing by clicking the appropriate link on the CoBrowser home page. This document has a Java applet embedded in it that has the information about all the existing groups and all the users in each group. The browser downloads a signed applet that requires certain privileges on the user machines and runs it. The applet registers itself with the Co-browsing server. The applet registration process involves informing the CS that it is active and the port number at which it will be listening. Upon receiving this information, the CS updates its table of active Web browsers in the session. Note, due to security restriction of Java applets, the Co-browsing server (CS) and the Web server (WS) have to be on the same host. The Java applet security environment does not allow applets to communicate to any host other than the host from where it is was downloaded.

Joining a Session. A new user, user-3 joins the existing group: cobrowsing-architecture. A new window, CoBrowser Window, is opened. The Java applet gets the URL of the current document being viewed by the group and sends it to the CoBrowser window. User-3 can start co-browse in the CoBrowser window. The Java applet remains active until the user leaves the co-browsing session. Since all new documents are loaded into the new CoBrowser window, the embedded applet remains active in the old window.

Co-Browsing in a Session. User-3 clicks on a link that points to the URL http://www.some.site/doc1.html or he resizes or scrolls in the new window. User-3's applet sends the request to the co-browsing server , which broadcasts the new URL or resize, scroll event to applets running on User-1 and User-2's Web browsers. The applets, upon receiving the URL information, issue a request to their respective browsers to load the new document in the new window or resize it or scroll the bars.

4 COPOINTER SUPPORT

When a group of users use CoBrowser system, one user may want to point to a specific position of the CoBrowser window to draw the attention of other group members. To satisfy this requirement, we implemented a special feature -- CoPointer. When one user moves his CoPointer to one specific position of a Co-Browsing page, all group members' CoPointers move to the same position. The CoPointer has the following features:

- The CoPointer always floats over CoBrowser Window.
- The CoPointer can be set and released.
- The group member who sets the CoPointer is the owner. Only the owner can lead the moving of the pointer and release it.
- The CoPointer will show the owner's name.
- There is only one CoPointer in a group, so a member in a group can only set the CoPointer after an existing CoPointer is released.
- The CoPointer points to the same position of the Co-Browsing page of a group when it first appears.
- The CoPointer keeps pointing to the same position of the group members' CoBrowser windows when the owner moves it.

This is actually quite difficult to achieve within our architecture. We could not without greatly increasing the complexity make the pointer be the typical pointer shape expected in distributed collaboration system. However, by using the browser window methods and properties provided in JavaScript, we implemented the CoPointer through a browser window. To create a CoPointer, we create new browser window that is a a page with the owner's name and a pointer's image which gives the appearance of CoPointer. "Pointer" is the name of the window title. To make browser window look like a pointer,we open the browser window as small as possible. We do so by opening the CoPointer at 100* 25 at (10, 10) of the screen, which is just big enough to hold the pointer image and owner's name. This browser window has no menubar, toolbar, locationbar, personalbar, and scrollbar, and not resizable. A snapshot of a cobrowsing session along with the CoPointer is shown in Figure 2.

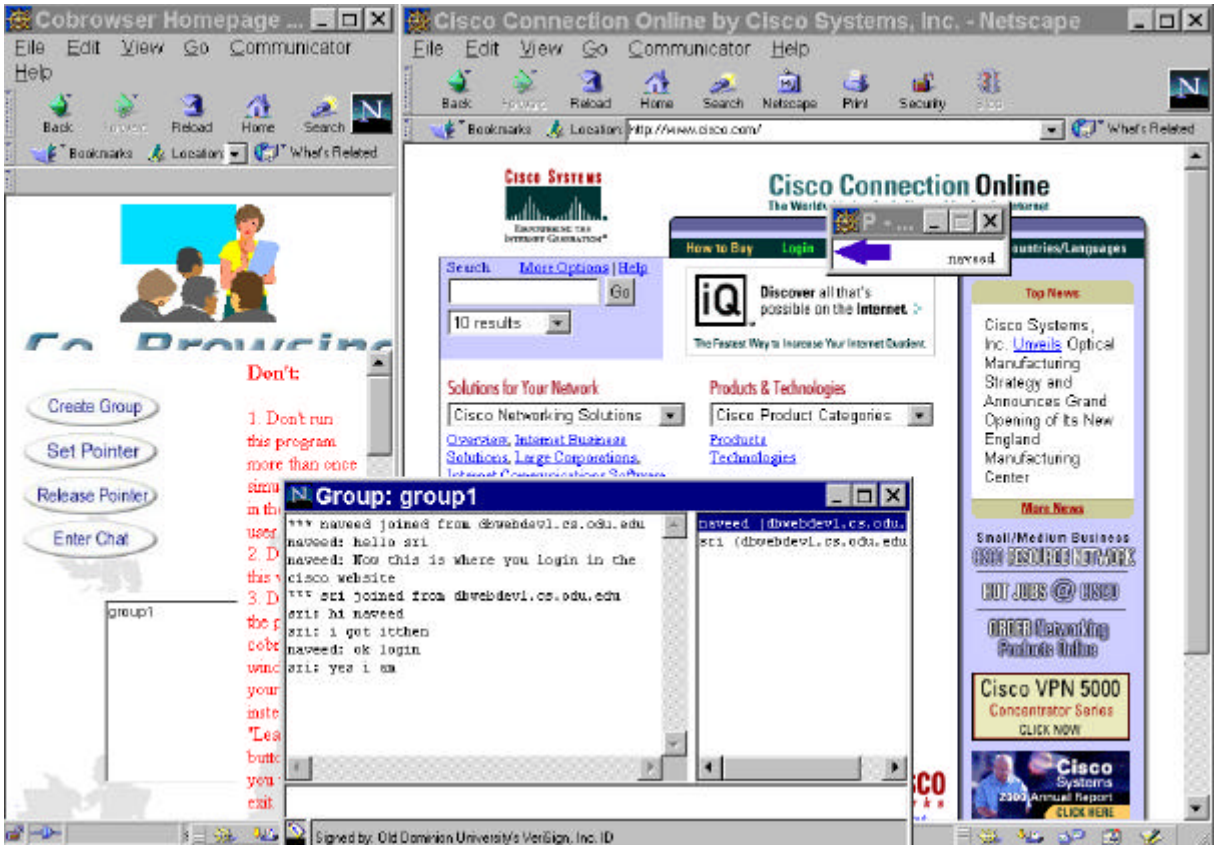

Figure 2. User 1 and User 2 in a chat session.

Because the CoPointer need to be coordinated with the instances at other users, the relative position of CoPointer to the CoBrowser window instead of the absolution position of the monitor is recorded and compared. When a CoPointer is set, a timer is turned on to check the relative position of the owner's pointer periodically. Once the relative position is changed, the new position is sent to CoBrowser server. Then CoBrowser server broadcasts the information to other group members whose pointers will be moved to the new position accordingly.

When a user joins or leaves a group, we need additional checking for CoPointer system. The group pointer status records whether a CoPointer of this group has been set, and the group pointer position records the current position of the group's CoPointer. When a CoPointer owner leaves a group, the pointer status is checked. If the CoPointer is still set, an alert pops up to warn the user to release the pointer before leaving the group. When a user joins the group, the group pointer status and position are checked. If a CoPointer has been set, a pointer window is shown on the position of the CoBrowser window indicated by the group pointer position.

5 CHAT SUPPORT

To alleviate the problem of not having audio yet available in our system we added as a short term solution a chat facility. We simply took on of the many free-ware JAVA chatbox classes and integrated it into our CoBrowser. The session management issues for Chat are similar to the CoPointer issues. A snapshot of a chat session along with a cobrowsing session is shown in Figure 2.

6 CONCLUSION

This paper describes a co-browsing architecture that allows a group of users to “surf” the web together. The architecture works with any graphical Web browser that supports Java applets. The users may potentially be geographically separated and working on different platforms. The system has been extensively tested for single and multiple groups and is available for demonstration from a server we run here at Old Dominion University: http://www.cs.odu.edu/cobrowser/. Source code licenses are available from the University (one company is using the software for a 'helpdesk' system) and we will make code available also for non-commercial, academic research teams. We are in the process to test the scalability of our architecture. Support for page annotations will also be included for the pages placed in a tour where a group of users can follow a prescribed tour. Providing a means for such asynchronous use will facilitate the creation of reusable courseware materials. It also allows greater flexibility for large groups of students working together whose respective time constraints (work, parenting, etc.) would otherwise prohibit traditional collaborative methods. Once the complete Java Media APIs are available, audio will also be supported with the co-browsing architecture. This feature will enable co-browsing participants to have simultaneous audio and web browsing explorations.

Acknowledgment. We would like to acknowledge W. Haiyan for implementing the CoPointer support.

7 REFERENCES

1. D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski and G. Premchandran, ``WebFlow -a visual programming paradigm for Web/Java based coarse grain distributed computing", http://www.npac.syr.edu/projects/webbasedhpcc/index.html
2. K. Maly, H. Abdel-Wahab, C.M. Overstreet, C. Wild, A. Gupta, A. Youssef, E. Stoica, E. Al-Shaer, ``Interactive Distance Learning and Training Over Intranets", IEEE Internet Computing}, Vol. 1, No. 1, January 1997, pp. 60-71
3. EMSL Web-Tour, http://www.emsl.pnl.gov:2080/docs/collab/webarch.html
4. TANGO–a Collaborative Environment for the World-Wide Web, http://trurl.npac.syr.edu/tango/papers/tangowp.html
5. J. Z. Davis, K. Maly, and M. Zubair, `` A Coordinated Browsing System", EDMEDIA 98, CD-ROM, Freiburg, Germany, June 98