A Framework to Integrate Synchronous and Asynchronous Collaboration

S. F. Li
Computer Laboratory
University of Cambridge
Cambridge CB2 3QG, UK
sfl20@cl.cam.ac.uk

A. Hopper
Department of Engineering
University of Cambridge
Cambridge CB2 1PZ, UK
ah@eng.cam.ac.uk

Abstract

For the last decade, the research in CSCW (Computer Supported Cooperative Work) has been focusing on synchronous collaboration, which requires the participants involved in common tasks to remotely share computer display workspaces simultaneously without leaving their workplaces. However, to support truly global cooperative work, asynchronous collaboration is equally prominent, in order to accommodate the participants who may not be available for the synchronous CSCW session. These participating individuals, whether working synchronously or asynchronously, may be mobile and may have to connect to and disconnect from the session repeatedly with ubiquitous systems. In this research paper, we describe a framework for asynchronous as well as synchronous collaboration. The framework provides facilities to transfer the screen images or frame buffers of the on-going CSCW session to remote users, allowing the available participants to share the view and the control of the session simultaneously, and to record the screen images or frame buffers for the absent participants to retrieve and playback the session at a later stage, with VCR-like control (i.e. fast forward, rewind, play and stop). The frame buffers are transferred and recorded in units of rectangles containing pixel values of the screen images. These rectangles are platform independent and can be dynamically directed to and displayed by heterogeneous systems such as X Windows or Windows NT, or by Web browser such as Netscape.

1. Introduction

In the computer supported cooperative working environment, people can work closely with each other even when they are geographically separated. To support synchronous collaboration at a distance, numerous CSCW applications (or real-time groupware [1]) exist to assist the remote sharing of workspace (i.e. computer display surfaces) between participating individuals involved in common tasks. These tools usually provide a joint viewing of the workspace in the What-You-See-Is-What-I-See (WYSIWIS) mode and the teleoperation for the participants to interact with the joint view. The Shared Whiteboard Tools and the Shared Application Tools described in [2] are examples of such applications.

The Shared Whiteboard Tools (SWTs) allow multiple remote participants to see the same window displayed on their computer screens to emulate a physical whiteboard conference. Each participant may mark up the whiteboard by using simple drawing tools, or may enter text, or may place background images on the board.

The Shared Applications Tools (SATs) result from an extension of the SWT. They allow multiple participants to share the view as well as the control of any interactive application. Windows of any shared application can be displayed on the computer screens of all the participants, although the shared application itself runs on one platform only. SATs are generally based on the X-Windows protocol, and run over the Internet Protocol. Examples include SharedX from Hewlett-Packard [3], and X Teleconferencing and Viewing (XTV) from the University of North Carolina at Chapel Hill and Old Dominion University [4].

Although the above tools which are designed for synchronous collaboration can be enhanced to accommodate late-comers of a CSCW conference [5], they don't, in general, offer support for participants who are temporally separated. World-wide specialists may reside in countries of different time-zones, hence may not be able to join a conference which requires simultaneous

participation from all individuals; part-time contractors colocated at the same company may be required to work different shifts on a system maintenance project, therefore don't interact with each other directly. A separate category of applications, known as the non-real-time groupware [1], have been developed to meet the requirement of people-to-people communication in an asynchronous mode. These applications, including electronic mail (or document exchange) and asynchronous computer conferencing (via topical lists, bulletin boards, etc.), whether in the form of monomedia or multimedia, are mainly messaging applications which provide minimum support for sharing, as compared with the existing real-time groupware for synchronous collaboration. As the participants who are absent from a synchronous CSCW session join the sharing of the remote workspace when they become available, they not only need to obtain the current state of the session, but also need to review the operations performed during their absence. For the last decade, the research in CSCW has been focusing on synchronous collaboration, but a significant portion of collaborative activities in the real world occur in an asynchronous manner. In the context of workspace and application sharing, synchronous and asynchronous collaboration are closely related, because the review of the synchronous session can provide visual cues for asynchronous collaboration which in turn provides temporally separated participants with the knowledge and the experience to catch up with and reengage in the synchronous session. Therefore the two collaboration modes should be integrated, rather than directed into two separate research streams.

In this research paper, we describe a framework supporting asynchronous as well as synchronous collaboration. The framework provides facilities to acquire and store the on-going CSCW session in units of screen image (or frame buffer) rectangles and transfers these rectangles containing pixel values to all the participants currently signed in, offering them a joint viewing of their shared workspace. On the other hand, the framework also accepts the user interaction events (e.g. key press, mouse button click, etc.) from the real-time conferees and forwards the events to the remote session, offering support for teleoperations from multiple distributed users. For late-comers, partially real-time and non-real-time participants, the framework admits them to the session as soon as they become available, by delivering them the up-to-date screen images reflecting the current state of the shared workspace and allows them to retrieve and playback the session they have missed, with VCR-like control (i.e. fast forward, rewind, play and stop). The distributed users can access the remote session through our framework with heterogeneous systems such as X Windows and Windows NT, or with their Web browser such as Netscape. The mobile users can connect to and disconnect from the session repeatedly through our framework from different endpoints.

2. The RFB protocol

Our framework is based on the RFB (Remote Frame Buffer) protocol developed at the *Olivetti and Oracle Research Laboratory (ORL)* [6, 7]. This is a client/server protocol for remote access to the Graphical User Interface (GUI) of windowing systems. The remote endpoint where the GUI is displayed and where the user interacts with the GUI is called the RFB client and the endpoint where the GUI is generated or where the frame buffer originates is the RFB server. The protocol defines the mechanism to transfer screen images or frame buffers from the server to the client. These screen images or frame buffers are transferred in units of rectangles containing pixel values and are therefore independent of the underlying operating and windowing systems such as Unix/X-Windows, Windows 95/NT, etc.. The protocol also allows the remote users to interact with the windowing system and applications by transferring the user input events (e.g. key press, mouse button click) from the client to the server.

The RFB client is stateless, allowing users to connect to and disconnect from the RFB server repeatedly from totally different endpoints with the state of the GUI preserved in the server. With the RFB protocol, we can dynamically transfer the entire computer desktop with its state and configuration (exactly the same as when it was last accessed) to a remote endpoint to achieve mobility.

The RFB client is thin. By the word "thin", we mean that the client requires minimum human resources to develop and minimum hardware resources to run. The client can be implemented in Java (including Java applications and Java applets) or in other programming languages such as C/C++, as the RFB protocol is language independent. The client can run on the widest range of hardware or Java-capable web browser. It fits into the general trend for thin-client architectures[14].

The basic data unit of the RFB protocol is a rectangle. The rectangle is a graphical element which can be specified by its width, height, the x and y coordinates of the upper-left corner, and its pixel values. Instead of sending the screen images frame after frame like sending videos, the RFB server is intelligent enough to only send the rectangles within the frame buffer that have been changed since the last transfer, in other words, it sends frame buffer updates rather than the whole frame buffers. A frame buffer update usually affects only a small area of

the frame buffer and may result in a sequence of rectangles being sent.

The RFB protocol is very similar to the T.128 (formerly known as T.SHARE) application sharing protocol used in NetMeeting [10]. Although T.128 itself is platform independent, NetMeeting only supports the sharing of Windows applications.

3. Framework functionality and architecture

Our framework seamlessly intercepts and manipulates the message streams between the RFB client and the RFB server with the following features:

- No changes are needed to the existing RFB client or server. To the client, the framework acts as a proxy server and to the server, it acts as a proxy client.
- The framework does not address security, however, the RFB server is password protected and the RFB clients have to send passwords for authentication before being connected. The passwords are routed through the framework in their encrypted forms.
- Our framework is a scalable infrastructure which can be plugged into client/server architectures to convert them to multi-client/multi-server for the purpose of collaboration. Here, we are using the RFB as an example, where we have converted the client/server into a multi-client/single-server architecture. These multiple clients receive exactly the same messages from the server at run time.

3.1 Framework functionality

We now explain the functionality provided by our framework in the two collaboration modes: the synchronous and the asynchronous modes.

In the synchronous mode, the framework turns the RFB client/server into a client/proxy/server architecture. By placing such a framework (i.e. the proxy) between the client and the server, we are able to merge all the messages from the multiple clients and forward these messages to the server as if they were from a single client; while on the other hand, we multicast the messages from the server to each client. In other words, we enable the synchronous RFB clients (also known as the Viewers, see Figure 1) to share the RFB server simultaneously, expanding a one-to-one client/server into a many-to-one multi-client/server communication channel, where each client receives exactly the same messages from the server (therefore shares exactly the same view) as the others participating in the collaborative session. However, different Viewers may experience different communication delays, especially when some Viewers are local and some are remote to the RFB server. Under these circumstances, an additional telephone line or audio channel will be helpful to coordinate user interactions. Since all the messages flow through the framework, they can be stored or recorded for future retrieval by late comers and absent participants.

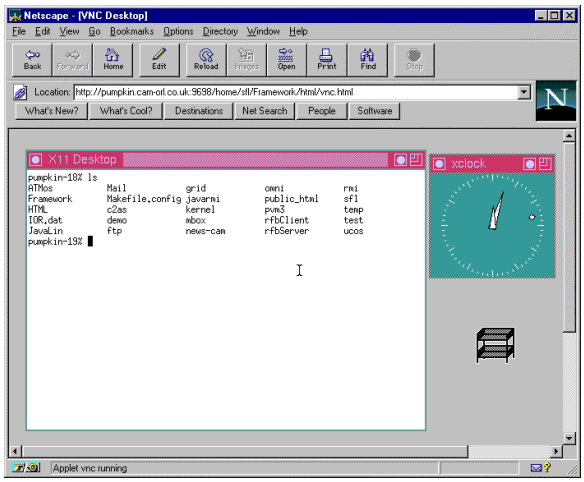

Figure 1. Viewer running as Java applet

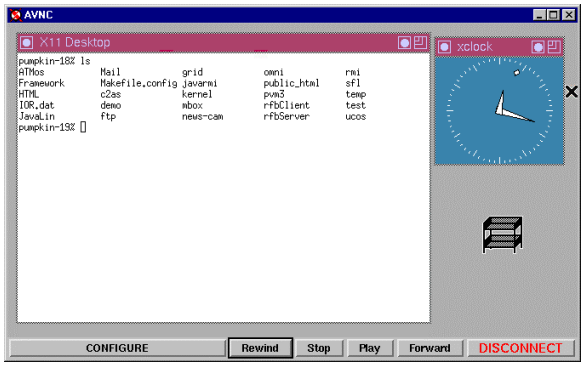

Figure 2. Reviewer running as Java application

In the asynchronous mode, our asynchronous RFB client (also known as the Reviewer, see Figure 2) retrieves the messages stored by the framework to playback the recorded session. It allows the user to interact with the recorded media with VCR-like control (i.e. play, stop, fast forward and rewind). Our recorded messages are time-stamped, therefore can be played back at the same rate as they were recorded.

3.2 Framework architecture

Our framework consists of a Server Agent which communicates with the RFB Server, a number of Client Agents, each corresponding to an RFB Client, and a Cooperation Manager which monitors and coordinates the agent activities (see Figure 3).

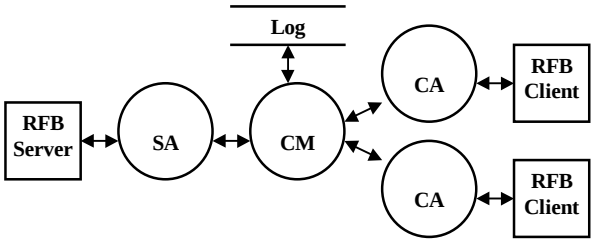

Figure 3. Framework of Server Agent, Client Agent and Cooperation Manager (in synchronous mode)

- Server Agent (SA)

  The SA acts as a proxy client to the RFB Server via a socket connection. The SA reads messages from the RFB Server and forwards them to the Cooperation Manager for routing, and delivers the messages forwarded by the Cooperation Manager to the RFB Server. The messages flowing between the SA and the RFB Server comply with the RFB protocol.

- Client Agent (CA)

  The CA acts as a proxy server to the RFB Client via a socket connection. The CA reads messages from the RFB Client and forwards them to the Cooperation Manager for routing, and delivers the messages forwarded by the Cooperation Manager to the RFB Client. And again, the messages flowing between the CA and the RFB Client comply with the RFB protocol.

- Cooperation Manager (CM)

  The CM is mainly responsible for two tasks: routing and recording. It routes the messages from the Server Agent to the Client Agent and vice versa, and records the messages that change the state of the remote frame buffer. It constructs an Agent Table where all the agents in touch with the CM are registered and a Route Table which specifies the messaging route from the source agent to the destination agent. As in the scenario illustrated by Figure 3, the Agent Table contains one Server Agent and two Client Agents, and the Route Table specifies that messages from the Server Agent will be routed to the two Client Agents and vice versa.

The recorded messages are stored in a log file in the same sequence in which the CM routes these messages. Like all other files, the log file can be transferred across the network by means of, for example, the File Transfer Protocol (FTP). The Reviewer (i.e. the asynchronous RFB client) can then open this log file to playback the recorded collaborative session. The message processing performed in the Reviewer is similar to that performed in the Viewer (i.e. the synchronous RFB client) except that the Reviewer reads the messages from the log file while the Viewer reads from the socket.

4. Experiments

Our basic data elements of transmission and recording are frame buffer rectangles. You might ask "*isn't that going to consume a lot of bandwidth and storage capacity?*". The answer is "*yes*", but we are willing to trade off bandwidth and storage capacity for a thin client architecture for the following reasons:

- We are promised with more bandwidth and storage capacity in the future. Paul Taylor reported that the telecommunication groups are investing huge sums in upgrading the Internet backbone infrastructure with fast transmission technologies such as ATM (Asynchronous Transfer Mode), Frame Relay and packet over Synchronous Optical Network (IP Sonet) [12] and Tom Foremski reported that over the last ten years, hard drive technology has been showing tremendous increases in performance and in storage capacity and will continue that way at least for several more years [13].
- Our experiments have shown that, with the built-in intelligence of the RFB server (which only sends frame buffer updates rather than whole frame buffers) and the various encoding schemes for the pixel data [7], the consumption of bandwidth and storage capacity is not as much as we think. Typically, to launch a Netscape Web browser to retrieve the home page of WETICE'98 from the session as shown in Figure 1 will cause approximately 0.5 Megabytes to be transmitted and recorded. (One side effect of placing the framework between the RFB client/server is that we are now able to generate auditing reports on messages flowing between the client and the server.)
- The performance of our system is quite acceptable. The RFB clients implemented in C appear to be as fast as an X display. The Java clients are slower, but will improve when the gap between the speed of Java

and the speed of native languages such as C/C++ closes. The original design of the framework is for the Server Agent, the Client Agent and the Cooperation Manager to be distributed objects among our LAN (Local Area Network) with Java RMI (Remote Method Invocation) [15] as the communication mechanism. However, our experiments show that the resulting framework has slowed down the work session significantly due to the transmission of multimedia streams (i.e. frame buffer rectangles) between distributed objects. Hence, we have now incorporated all the objects (the agents and the managers) within one process to reduce network traffic and the performance has been enhanced to an acceptable level.

- The advantages of a thin client architecture are described in the next section, where our approach is compared with other related work.

5. Related work

Fluckiger defined CSCW as the field concerned with the design of computer-based systems to support and improve the work of user groups engaged in common tasks and the understanding of the effects of using such systems [2]. It is a field which has been studied since the early 1980s by a large number of research individuals and organisations. Most of the research is focused on synchronous collaboration.

Our aim is to integrate synchronous and asynchronous collaboration. First of all, let's take a look at how other systems integrate these two collaboration modes. Multi-user shared editors such as the GROVE (GRoup Outline Viewing Editor) [1] are collaboration-aware tools, i.e. they are aware that a multi-user collaboration is taking place. Although they are designed to support simultaneous multi-user interactions, they also, to some extent, support asynchronous collaboration extending over a longer period. For example, the shared editors can store the result of editing for further use. Participants of a work session can enter and leave at any time. When they enter or re-enter a session, they receive an up-to-date document. However, they cannot replay the user actions that result in the current document without affecting the other users editing the same document. In other words, these tools don't support reviewing of a work session. Our framework can enable the sharing of an editor, e.g. the *vi* editor, which itself is not collaboration-aware. By recording the editing, we can replay and review the session independently from the other users, as if we are playing back a video clip. Before this project, we designed a talk tool in [8] to facilitate communication between humans through time by storing the conversation as ordered entries in a Tuple Space, and allow the participants to track down how the conversation (in term of text messages) is carried out during their absence. However, like the shared editors, the talk tool is designed for a specific task (in this case, for Internet messaging), therefore it doesn't support the sharing of ordinary collaboration-unaware applications in general.

To achieve our aim, we have made use of three technologies. The first is to separate the user interface from the application logic for mobile display, this is done by the use of the RFB protocol; the second is to place a proxy between the client/server architecture to intercept, record, redirect, merge and multicast the message streams between the clients and the servers, this is done by the use of our framework; the last is to extend collaboration to a global scale, this is done by the use of the Java and World Wide Web technology.

Let's first look at the other protocols which also separate the user interface from the application logic. The X protocol is an obvious candidate. The X Window System allows window applications or clients to access the display only through the X server, which is a separate process that arbitrates resource conflicts and provides display, keyboard, and mouse services to all clients accessing the display. Built on the X protocol, the Remote Shared Workspaces (RSW) system described in [9] is a layer of software to convert single-user applications into tools for use by a group of distributed users during a real-time collaborative session. By extending the RSW system which is restricted to support the sharing of textual applications only (e.g. the *vi* editor), the X Teleconferencing and Viewing (XTV) is a system for sharing X Window applications synchronously among a group of distributed users at workstations running X and interconnected by the Internet [4]. Like the RSW and the XTV, most of the Shared Application Tools are based on the X protocol, so why do we use the RFB protocol instead? We find the stateless and the ubiquitous nature of the RFB clients appealing.

The XTV accommodates late-comers by protocol filtering and archival which records the minimal information necessary to generate an exact replica of the shared applications' interface on the late-comer's display [5], while in our system, with the stateless property of the RFB clients, we can automatically accommodate late-comers and mobile users (who connect to and disconnect from the work session repeatedly) without any requirement for storing messages related with the state of the work session. When users join or re-join the session, they receive an up-to-date frame buffer shared by all participants. Unlike the X protocol, the RFB protocol

messages contain rectangles of pixel values with no semantics. Once recorded to persistent storage, these messages can be rewound and replayed as many times as required. Hence, a consulting or teaching session can be recorded and archived for future reference.

Although X can support a spectrum of hardware displays ranging from small monochrome units to advanced graphics systems, it requires the display platform to run a program called the X server. Hence, XTV can only work under the X Window System environment. The RFB protocol is independent of the underlying windowing system, as frame buffers can be transferred across heterogeneous platforms. Hence, the RFB clients are ubiquitous and can collaborate on various computing systems (e.g. between X Window and Windows-NT platforms).

Let's now look at the other systems that also place proxies between the client/server architecture to re-route the message streams between the clients and the servers. The XTV uses a proxy to multicast and merge the messages between the shared X applications and the X displays of the participants for synchronous collaboration [4]. ORL's Teleporting system uses a proxy to redirect the messages between the X applications and the different X displays for mobile computing [11]. Our framework serves similar purposes. While re-routing the messages between the RFB client and the RFB server to support synchronous and mobile collaboration, we also record the messages for later playback, thus supporting asynchronous collaboration in a natural way.

Finally, the introduction of the World Wide Web (WWW) and Java technology into CSCW has enabled global collaboration on Internet. For example, CoReview uses XTV to share Mosaic to facilitate creation and review of proposals by spatially separated group members [16]. Our system extends to the World Wide Web in two dimensions. Firstly, our clients can share a Web browser like sharing any other ordinary application. Secondly, our clients can be downloaded dynamically through Internet and run as Java applets in a Web browser.

Acknowledgement

Sheng F. Li is sponsored by *EPSRC* and *ORL*. Andy Hopper is the professor of Communications Engineering and the director of ORL. The authors would like to thank Sai-Lai Lo and the STAR group for supporting, and Stuart Wray and Antony Rowstron for reviewing.

Reference

[1] C. A. Ellis, S.J. Gibbs, and G.L. Rein. "Groupware: Some Issues and Experiences", *Communications of the ACM*, vol. 34, no. 1, January, 1991

[2] F. Fluckiger, *Understanding networked multimedia*, Prentice Hall, 1995

[3] D. Garfinkel, B. C. Welti, and T. W. Yip, "HP SharedX: A Tool for Real-Time Collaboration", *Hewlett-Packard Journal*, April, 1994

[4] H. M. Abdel-Wahab, M. A. Feit, "XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration", *IEEE Conference on Communication Software: Communication for Distributed Applications & Systems*, North Carolina, April, 1991

[5] G. Chung, K. Jeffay and H. Abdel-Wahab, "Accommodating late-comers in a distributed system for synchronous collaboration", *Technical Report TR91-038*, Department of Computer Science, University of North Carolina at Chapel Hill, October, 1991

[6] K. R. Wood, T. Richardson, F. Bennett, A. Harter and A. Hopper, "Global Teleporting with Java: Toward Ubiquitous Personalised Computing", *IEEE Computer*, vol. 30, no. 2, February, 1997

[7] T. Richardson, Q. Stafford-Fraser, K. R. Wood and A. Hopper, "Virtual Network Computing", *IEEE Internet Computing*, vol. 2, no. 1, January/February, 1998

[8] A. Rowstron, S. Li, and R. Stefanova, "C^2AS: A System Supporting Distributed Web Applications Composed of Collaborating Agents", *IEEE Sixth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'97)*, June 18-20, 1997.

[9] H. M. Abdel-Wahab, S.-U. Guan, J. Nievergelt, "Shared Workspaces for Group Collaboration: An Experiment Using Internet and UNIX Interprocess Communications", *IEEE Communications Magazine*, Vol. 26, No. 11, Nov., 1988

[10] *NetMeeting 2.0 Reviewers Guide*, June 1997, *http://www.microsoft.com/netmeeting/*

[11] T. Richardson, F. Bennett, G. Mapp and A. Hopper, "Teleporting in an X Window System Environment", *IEEE Personal Communications Magazine*, Third Quarter, 1994

[12] P. Taylor, "The need for speed", *Financial Times*, Wednesday November 5, 1997

[13] T. Foremski, "Bloatware drives demand for storage", *Financial Times*, Wednesday November 5, 1997

[14] B. Quinn, "Thin Windows Firms Up", *BYTE*, May, 1997

[15] A. Wollrath, R. Riggs and J. Waldo, "A Distributed Object Model for the Java System", *Computing Systems*, The USENIX Association, Vol. 9, No. 4, Fall 1996

[16] K. J. Maly, H. Abdel-Wahab, R. Mukkamara, A. Gupta and A. Prabhu, "Mosaic + XTV = CoReview", *The Third International World Wide Web Conference: Technology, Tools and Applications*, Germany, April 10-14, 1995.