APPLICATIONS OF STATELESS CLIENT SYSTEMS IN COLLABORATIVE ENTERPRISES

SHENG FENG LI
Computer Laboratory, University of Cambridge, Cambridge CB2 3QG,UK, sfl20@cl.cam.ac.uk
QUENTIN STAFFORD-FRASER
The Olivetti & Oracle Research Laboratory, Cambridge CB2 1QA, UK, qsf@orl.co.uk
ANDY HOPPER
Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, UK, ah@eng.cam.ac.uk

Key words: Internet/Intranet Computing, Computer Supported Cooperative Work (CSCW), Client/Server

Abstract: This paper identifies the current difficulties faced by the IT professionals working for collaborative enterprises and explains how we exploit and extend the so-called stateless client systems to support those individuals in cooperative work. Stateless client systems are the software tools that separate the display interface from the application logic in windowing systems. They embed a client/server architecture, where the server executes all applications and the client simply presents the frame buffers or screen images to the user and accepts user input. Since the entire system state is preserved in the server, the client becomes stateless. By providing these stateless clients with suitable coordination mechanism, we enable geographically separated users to share workspaces and applications in a work session. And by recording the messages flowing between the client and the server, we enable temporally separated users to search for and playback previous work sessions to share knowledge and experience.

1. INTRODUCTION

Teamwork is regarded as an important attribute in a successful organisation and organisations may collaborate to form strategic alliances. The IT professionals working for technology-intensive enterprises are facing great difficulties in supporting the ever-growing system infrastructure and customer base. Since they may or may not share the same terminology with their customers ranging from novices to experienced users, the text descriptions of a concept by either side, whether in a spoken form or in a written form, can be incomplete, inconsistent, and incomprehensible. Most of the time, a look at each other's screen is more illustrative than a thousand words. Therefore, providing appropriate visual cues for collaborative work such as remote teaching will enhance the performance of geographically separated users. On the other hand, problems reported by one customer may also be experienced by the others; solutions provided by one professional may be reused later on. Knowledge and experience, if not documented properly, will be lost or forgotten. So managers need to keep a record of known problems and existing solutions to avoid unnecessary reengineering. Providing appropriate visual cues in these records again plays an important role in encouraging temporally separated users to share knowledge and experience.

We envisioned a system that can separate the display interface from the application logic so that the applications can be installed and executed at one site, but their screen images can be transferred across the network to be displayed at a remote site and saved for future reference. These screen images provide valuable visual cues to users who share workspaces and applications, and to users who

share knowledge and experience. The VNC (Virtual Network Computing) developed at ORL (Olivetti and Oracle Research Laboratory) has made such a separation a reality (Richardson, 1998). Built on any windowing system, the tool breaks it into client/server pieces, where the applications run on the server and the screen images are displayed at the client. It transfers frame buffers or screen images in units of rectangles containing pixel data from the application server to the display client and transfers user events such as keystrokes from the display client to the application server. In (Li, 1998a), we extended VNC by multicasting the screen images to multiple users and merging the user events from them to support synchronous collaboration; we also recorded the screen images for future replay to support asynchronous collaboration. In this paper, we further investigate the potentials of the VNC system for CSCW and discuss our extensions to a greater width and depth. In Section 2, we will describe the protocol upon which the VNC system is built; in Section 3, we'll explain how we extend the system with suitable coordination mechanism to support collaborative work. Section 4 will present the performance results of our system and its extensions, and finally in Section 5, we'll cover the related work at other research sites.

2. VIRTUAL NETWORK COMPUTING (VNC)

When Oracle shipped the Network Computers in 1996, it brought us cheaper computing by keeping all the applications on the server and allowing them to be downloaded to and executed at the client (Halfhill, 1997). The VNC system pushed this idea to an extreme by moving the execution of applications to the server as well. When the applications are executed at one site, their frame buffers or screen images can be transferred across the network to be displayed at a remote site. Without changing any windowing systems, VNC runs on their platforms and breaks them into client/server pieces, namely the VNC server, the VNC client and the VNC protocol that connects the previous two. The VNC server executes all applications and generates the frame buffer. The VNC client displays these frame buffers and accepts user input. And the VNC protocol defines the mechanism that transfers the frame buffers from the server to the client and the user events from the client to the server. Thus the VNC makes the client/server computing model even more network-centric with the thinnest possible client (i.e. the stateless client).

More details of the VNC system and its protocol can be found in (Richardson, 1998; Wood, 1997). Here, we only emphasise two types of messages that are exchanged between the VNC client/server: the *user event* message and the *frame buffer update* message. We intercept and manipulate these messages extensively in our extensions to support collaborative work.

When the VNC client receives input such as keystrokes and mouse clicks from the user, it sends the *users event* messages to the server for processing. The *user event* can be either a key event or a pointer event, as illustrated by Figure 1a and 1b:

- The *up/down flag* indicates whether a key is pressed or released, and in most of the cases, the *key* carries its corresponding ASCII value (Figure 1a).
- The *button mask* specifies the state of the mouse buttons, i.e. whether they are pressed or released; the *x position* and the *y position* identify the current location of the mouse pointer (Figure 1b).

When the VNC server receives the request for screen update from the client, it sends the *frame buffer update* messages to the client for displaying. The message format is shown in Figure 1c and 1d.

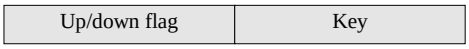

Figure 1a. **User event message (key)**

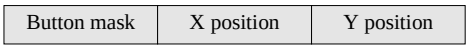

Figure 1b. **User event message (pointer)**

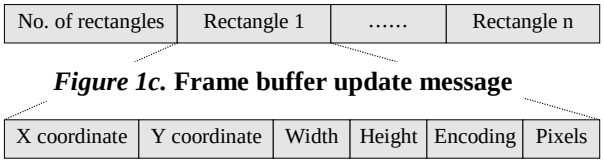

Figure 1c. **Frame buffer update message**

Figure 1d. **An update rectangle (raw encoding)**

- The *no. of rectangles* specifies the number of rectangles contained in this update message. A *frame buffer update* consists of a sequence of rectangles, each representing a rectangular area of the frame buffer. Together, these non-overlapping rectangles[1] cover the frame buffer area which has been changed since the last update (Figure 1c).
- An *update rectangle* uniquely identifies a rectangular area of the frame buffer by specifying the *x, y coordinates* of its upper-left corner, and its *width* and *height*. The *encoding* specifies the encoding scheme the pixel data contained in the rectangle will follow. In the case of raw encoding (i.e. no encoding at all), *pixels* is an array of bytes containing the pixel values within the rectangle in the scan-line order, i.e. from left to right and from top to bottom (Figure 1d).

But why rectangles? Why not triangles, or circles etc.? First of all, the rectangle is a graphical primitive that is simple to specify. Only four integers are required to specify a rectangle, i.e. the x, y coordinates of its upper-left corner, its width and its height, while triangles are more complicated. Secondly, a screen update usually affects only a small area of the frame buffer. Only this area needs to be transferred and it can be covered by a number of non-overlapping rectangles, while circles cannot cover the area in a mutually exclusive and collaboratively exhaustive way. Thirdly, pixels in rectangles can be presented continuously in a scan-line order, while in triangles or circles, it's difficult to present pixels without extra calculation or specification. And finally, most of the widgets (e.g. windows and icons etc.) in windowing systems have a rectangular shape. Therefore, it's natural to have rectangles as our basic data unit for transferring frame buffers. These rectangles form a continuous stream at real time.

[1] The original VNC protocol does not require that the rectangles contained in the *frame buffer update* message be non-overlapping, however, our recording and playback mechanism enforces such a condition.

3. SYSTEM ARCHITECTURE AND FUNCTIONALITY

The VNC client and server communicate through socket connections. We extend the client/server architecture by placing a proxy between the two components to intercept and manipulate the message streams. The proxy is where we provide the coordination mechanism to the clients to support cooperative work.

As depicted in Figure 2, our proxy consists of a number of surrounding modules and a central cooperation manager. For each VNC server, there is a Proxy Client Module (PCM) that pretends to be a VNC client and intercepts the *frame buffer update* messages. And for each VNC client, there is a Proxy Server Module (PSM) that pretends to be a VNC server and intercepts the *user event* messages. These messages will be routed to the central Cooperation Manager (CM) for manipulation. The CM multicasts the messages from the VNC server to the multiple collaborative VNC clients via their corresponding PSMs and merges the messages from the VNC clients to the shared VNC server via its corresponding PCM. While at the same time, it stores the messages to the log files.

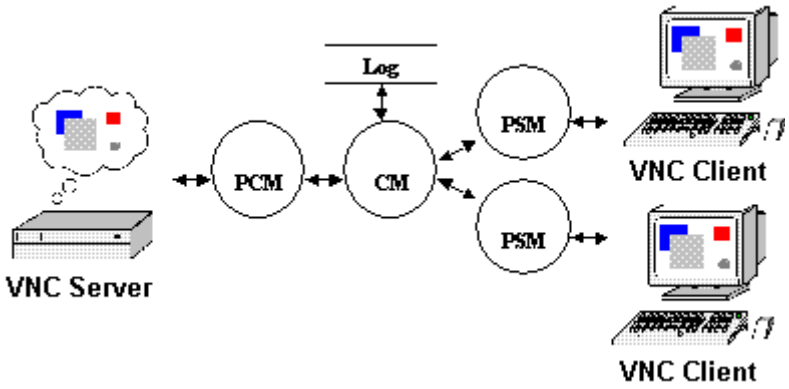

Figure 2. **Proxy Modules and Cooperation Manager**

3.1 Coordination

The Cooperation Manager monitors and coordinates the activities of the surrounding modules. It provides the coordination mechanism through the following access primitives or method invocations:

- **Register** This method allows the modules to register themselves in the CM's Module Table after they start to execute. The CM keeps track of all the modules currently connected to it.
- **Remove** This method allows the modules to remove themselves from the Module Table before they exit from the system.
- **Route** This method allows the modules to send messages to other modules in the system. The CM maintains a Route Table that specifies the destination modules for messages sent by each source module.
- **Record** This method allows the modules to record the messages sent by them. The recording is performed by the CM and stored in the log files.
- **Request** This method allows the Proxy Server Modules to request the floor for their corresponding VNC clients. After a VNC client obtains the floor, the CM only routes the user events coming from this particular client and our system enters the *supervision mode*, i.e. only this supervising client can operate the shared work session while the rest of the collaborative clients can only view the session.
- **Release** This method allows the Proxy Server Modules to release the floor for their corresponding VNC clients. When the VNC client releases the floor, our system returns to its normal mode and the collaborative clients can share the view and the control of the remote work session.
- **Reconnect** This method allows the Proxy Client Module to switch to a new VNC Server. When this method is invoked, the PCM closes the connection with the current VNC Server and reconnects to a new server. The collaborative clients can now decide whose server is to be shared and dynamically switch between various homogeneous and heterogeneous servers in a shared work session

Apart from the above methods that are invoked by the connected modules, the CM also facilitates a simple notification service to the clients. It acts as a session server where the user-related states such as the number of participants are centralised. When there is a change of the shared states, it will notify all the participating clients via their corresponding Proxy Server Modules.

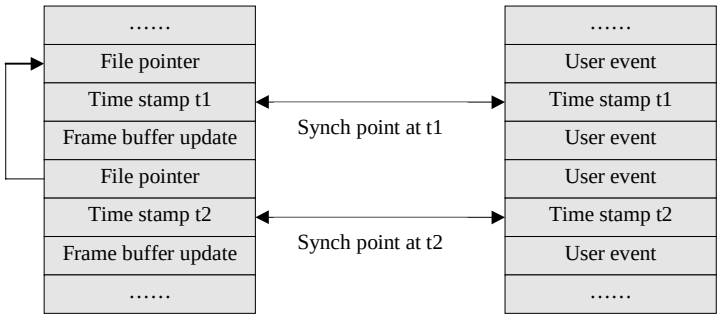

Figure 3. **The log files of *frame buffer update* messages (left) and *user event* messages (right)**

Regarding the log files, there are two types: one records the *frame buffer update* messages, the other records the *user event* messages. Both messages are time-stamped when they are recorded. These time-stamps serve as synchronisation points in the message sequences. Figure 3 shows the layout of the two log files.

The log file of *frame buffer update* messages alone is sufficient for the playback of a recorded session. An asynchronous VNC client can read these messages from the file in the same way that the synchronous VNC client does from the socket. The log file of *user event* messages is required solely for the automatic construction of the index table that maps events or groups of events to the synchronisation points in the log file of *frame buffer update* messages. In this way, we can seamlessly index the recorded session by events such as key words. To speed up the search for key words during playback, we also insert optional checkpoints to the log file of *frame buffer update* messages. A checkpoint is a *frame buffer update* message that contains the pixel data for the entire frame buffer. While searching for the key words, the asynchronous VNC client reads in the nearest checkpoint and retrieves a full screen image, then it goes through the recorded session from there with a number of screen updates until the requested screen is found.

3.2 Cooperation

We support cooperation in both the synchronous and the asynchronous mode. In the synchronous mode, our proxy enables the sharing of workspaces and applications by multicasting the *frame buffer update* messages to the multiple collaborative users and merging the *user event* messages to the shared work session. As the result, the users simultaneously share the view and the control of the workspaces and applications (Figure 4). The proxy also provides support for user awareness and floor control. Our work session exemplifies the calling-in model (i.e. the participants call into the session to indicate their interest in joining), with which awareness applications work well (Patterson, 1996). When a new user enters the work session, an existing user departs, or one of the users obtains or releases the floor, the proxy will notify all the participating individuals to update their user interfaces and promote the impression of working together. The floor control allows users to share the work session without access conflicts. It dynamically grants collaborating users with temporal permissions (or floors) to guarantee mutually exclusive operations on the shared workspaces and applications. Our extended system can support various floor control policies, for example, the *chair guidance* policy (Dommel, 1997), where the IT professional is the arbiter over the usage of the floor.

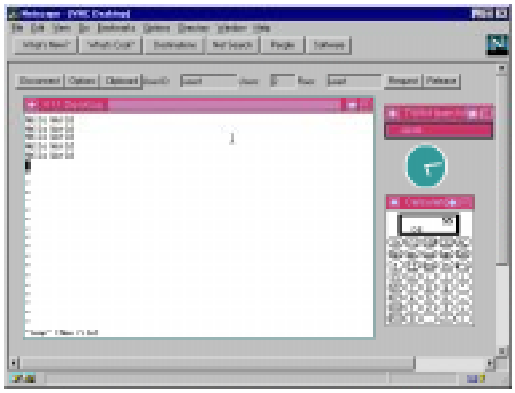

Figure 4. **Sharing workspaces and applications**

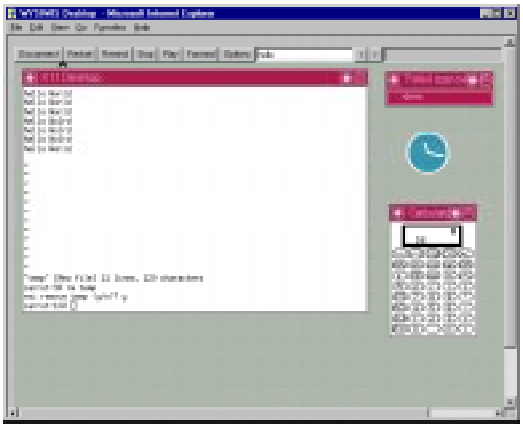

Figure 5. **Sharing knowledge and experience**

In the asynchronous mode, the log files provide the users with the recorded media. The users can retrieve and replay previous work sessions to share knowledge and experience. Our VNC client, running in its asynchronous mode, provides VCR-like control and allows users to play, stop, fast forward and rewind a recorded session like playing back a video tape. It also facilitates searching forward and backward for events such as key words. For example, while replaying the work session as depicted in Figure 5, the user will be able to search for the word "hello" typed in the *vi* editor by someone who worked in the session previously. The screen that displays this word will be fetched and presented in a way similar to word searching in a word processor document.

4. PERFORMANCE

Our applications of VNC have introduced a trade off. We are trading off bandwidth for a platform independent infrastructure in collaborative enterprises. In this section, we explain why such a trade off is acceptable by measuring the bandwidth consumption of our system with carefully designed experiments.

We designed three tasks to be performed by two collaborative users who share workspaces and applications (Figure 4) and converse with each other orally during the tasks. The VNC server and the proxy are executing on a UNIX workstation at ORL. User1 runs the VNC client as a Java application on a second UNIX workstation at the same site with a 100Mbps link to the servers. And user2 runs the client as a Java applet within a Netscape Web browser on a Windows NT workstation from the Computer Laboratory of Cambridge University with a 10Mbps link.

- In task 1, the users type 10 lines of "Hello World" in the *vi* editor. User1 types the first 5 lines and user2 types the next 5 lines. This task involves key events only.
- In task 2, user1 clicks "1+2+…+10=" in the calculator, and user2 drags the calculator to a new location after the result of the calculation appears. This task involves mouse events only.
- In task 3, user1 launches a Netscape browser with the geometry 600×400, then goes to the book-marked AltaVista search engine page and searches for the word "CSCW". Once the search results start to appear, user2 presses the "Page Down" key to view the list of the first 10 matches. This task involves key and mouse events, and image rendering.

The clock on the screen is ticking while the users perform the above tasks.

The tables below show the performance statistics of our system. The results are obtained by taking the average of three consecutive trials.

We run two sets of tests. In the first set, the frame buffers are transmitted and saved in rectangles of pixel data without any encoding or compression scheme (Table 1), while in the second set, various encoding schemes are applied to the pixel data (Table 2). The details of the encoding schemes we use can be found in (Richardson, 1998). The *Size* shows the amount of data (in megabytes) transmitted from the VNC server to the VNC clients and stored by the proxy server during the task sessions. It does not include the size of the initial whole screen image transmitted and stored (which is 0.44 Mbytes if the data is not encoded) when the user first signs into the work session. When users first connect to a work session, they will receive a whole screen image reflecting the current state of the session. The following frame buffer updates only cover the part of the screen that is changed since the last update, and their sizes are usually much less than the size of the whole screen. The *Time* is the time (in seconds) taken by the users to complete the tasks, and the *Peak Rate* shows the data transmission and recording rate at peak time.

Table 1. **Performance statistics (raw encoding)**

	Task1	**Task2**	**Task3**
Size(MB)	0.40	0.10	2.86
Time(S)	46	73	187
Peak Rate(MB/S)	0.15	0.03	0.13

Table 2. **Performance statistics (with encoding)**

	Task1	**Task2**	**Task3**
Size(MB)	0.03	0.06	0.77
Time(S)	45	39	118
Peak Rate(MB/S)	0.01	0.01	0.06

The transferring and storing of frame buffers or screen images will inevitably increase the bandwidth and storage consumption. Much effort has been spent in reducing the bandwidth and storage required. Since our workspace and application sharing is purely computer-based, we take advantage of the spatial and temporal correlation of the frame buffers. That is to say, most of the time when user interactions take place, only a small area of the screen is affected. So that we only need to transfer and record the area that has been changed since the last screen update, hence keep the data transferred and recorded to its minimal. With various encoding schemes for the pixel data (Richardson, 1998), we observe that the performance of the VNC client implemented in C is gradually approaching that of the traditional X terminal over our 100Mbps Local Area Network. Some members of our laboratory have been using the VNC system to perform daily tasks for years now. In particular, our Window NT users have been using VNC as well as or instead of Hummingbird's Exceed, an X server running on Windows systems, to access their X Windows sessions. We have found that our system has an acceptable performance in Local Area Network (LAN) and it's also useful for occasional work over Wide Area Network (WAN), however, it's too slow to be usable with dial up links. In (Li, 1998b), our experiments show that the new medium (i.e. the stream of rectangles containing pixel data of the screen images) with lossless encoding consumes less storage space than required by an equivalent MPEG video with lossy compression, when the screen updates are not too complex. Therefore we are convinced that, in the near future, when multimedia applications such as *video-on-demand* gain their popularity, our idea of "frame-buffer-on-demand" will naturally follow.

5. RELATED WORK

OK, we have sacrificed the bandwidth. What have we got in return? In this section, we explain why our trade off is worthwhile by comparing our technology with the traditional technologies.

The traditional systems that enable the sharing of single-user collaboration-unaware applications are generally based on the X protocol. For example, the X Teleconferencing and Viewing (XTV) from the University of North Carolina at Chapel Hill and Old

Dominion University is a system for sharing X Window applications synchronously among a group of distributed users (Abdel-Wahab, 1991). The reliance of XTV on the X protocol is conceptually at the same level as the reliance of our extended system on the VNC protocol. However, X protocol requires the display platform to run a program called the X server. As the result, XTV can only work under the X Window System environment. The VNC protocol is independent of the underlying windowing system, as frame buffers can be transferred across heterogeneous platforms. Hence, the VNC clients are ubiquitous and can share X Window applications on Windows-NT platforms and vice versa. Furthermore, the VNC protocol is much simpler because all the system state is maintained on the server and the client is completely stateless, which makes the *frame buffer update* and the *user event* the only two main types of messages in exchange. Therefore we encounter fewer problems while expanding the VNC system to enable workspace and application sharing. Taking the floor switching mechanism as an example, in XTV, the floor can be switched only if there is no pending reply for a request (Abdel-Wahab, 1994), while in our system there is no such dependency, as neither the *frame buffer update* nor the *user event* requires replies. In (Li, 1998a), we discussed the similarity and the difference between XTV and our system extensively.

Our applications provide the framework to integrate synchronous and asynchronous collaboration (Li, 1998a). We have compared the synchronous mode of operations with XTV. Here, we compare the asynchronous mode of operations with traditional systems that support the recording and replaying of computer work sessions. Lotus ScreenCam (Lotus, 1997) uses dynamic screen captures to create movies for training and customer support. It delivers instructions and procedures with the exact visual cues that illustrate what to do. However, a work session can only be recorded and replayed on Windows platforms. Our platform independent nature of the VNC protocol again brings us the advantage, i.e. we can record a Unix session and replay it on a Windows PC and vice versa.

Apart from the VNC, other stateless client or thin client systems also exist. Kanter presented the so-called thin-client/server computing model in (Kanter, 1998), using Citrix Systems as an example. Built on the Independent Computing Architecture (ICA) protocol, which is conceptually similar to the VNC protocol, the Citrix WinFrame allows Windows-based applications to run at one location and display the program's user interface somewhere else, even over low-bandwidth connections. This system then leads to Microsoft's Window-based Terminal Server code-named Hydra (Microsoft, 1997a), and Citrix Systems' add-on MetaFrame (formerly known as pICAsso). Our system also conforms to the thin-client computing model. We say that our clients are stateless (the thinnest), because unlike some other thin-client systems that may manage some states locally at the client to optimise for low-bandwidth situations, our system maintains all the system states in the server. With the stateless nature of the client, mobile users can connect to and disconnect from the work session arbitrarily, and every time they reconnect, they see exactly the same state as when the session was last accessed.

The thin-client systems described above can be exploited and extended to support collaborative work. Citrix WinFrame has a feature called "shadowing", which allows one user to view the video, keystrokes and mouse movements of another user's session (Kanter, 1998). NetMeeting (Microsoft, 1997b) shares the same protocol T.128 (formerly known as T.SHARE) with Hydra. T.SHARE is also conceptually similar to the VNC protocol. It enables the multiple NetMeeting participants to see the image of the shared applications. Unlike WinFrame and NetMeeting that are closely tied to Windows operating system, our system is platform independent. We can share Windows

applications on Unix workstations and share X Windows applications on PCs. We took our system a step further to facilitate recording of a synchronous session for asynchronous users, and automatically index the recorded session with user events to facilitate searching, hence provide support for synchronous and asynchronous collaboration in an integrated manner.

6. CONCLUSION

This paper describes how we exploit and extend a stateless client system known as the VNC (Virtual Network Computing) to support collaborative work. We enable users to share workspaces and applications synchronously and to share knowledge and experience asynchronously. To achieve this, we place a proxy between the VNC client and server to provide the coordination service. As the result, the system state is maintained in the server, the user state is managed in the proxy, and the client remain stateless.

Our next stage of the project is for us to create a knowledge pool using the recorded sessions as the media to improve the sharing of knowledge and experience among IT professionals and their customers. This knowledge pool will contribute to what Tanaka refers to as the Meme Pool (Tanaka, 1996). The users will be able to edit, publish, browse through, and playback recorded sessions on the World Wide Web (Berners-Lee, 1996). We believe that the seamless recording of work sessions and the visual cues provided by our media will encourage users to share their knowledge and experience.

ACKNOWLEDGEMENT

Sheng Feng Li is a Ph.D. student sponsored by EPSRC and ORL, Quentin Stafford-Fraser is a research engineer at ORL, and Andy Hopper is the professor of Communications Engineering and the director of ORL. We would like to thank the STAR group of ORL for supporting the project and Antony Rowstron from the Laboratory for Communications Engineering for his valuable comments.

REFERENCE

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

Abdel-Wahab, H. & Jeffay, K. 1994, "Issues, Problems and Solutions in Sharing X Clients on Multiple Displays", *Journal of Internetworking Research & Experience*, 1994

Berners-Lee, T. 1996, "WWW: Past, Present, and Future", *IEEE Computer*, October, 1996

Dommel, H-P & Garcia-Luna-Aceves, J. 1997, "Floor control for multimedia conferencing and collaboration", *Multimedia Systems*, no. 5, pages 23-38, 1997

Halfhill, T. 1997, "Cheaper Computing", *BYTE*, April, 1997

Kanter, J. 1998, *Understanding Thin-Client/Server Computing*, Microsoft Express, ISBN 1-57231-744-2, 1998

Li, S. & Hopper, A. 1998a, "A Framework to Integrate Synchronous and Asynchronous Collaboration", *IEEE Seventh International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'98), IEEE Computer Society Press*, June 17-19, 1998.

Li, S. & Hopper, A. 1998b, "What You See Is What I Saw: Applications of Stateless Client Systems in Asynchronous CSCW", *The Fourth International Conference on Computer Science and Informatics (CS&I'98)*, Research Triangle Park, North Carolina, Oct. 23-28, 1998.

Lotus 1997, "Lotus ScreenCam", http://www.lotus.com/products/screencam.nsf, July 1997

Microsoft 1997a, "Microsoft Windows NT 'Hydra' and Windows-based Terminals", *http://www.microsoft.com/ntserver/info/hydra.htm*, September 1997

Microsoft 1997b, *NetMeeting 2.0 Reviewers Guide*, June 1997, *http:// www.microsoft.com/netmeeting/*

Patterson, J., Day, M. & Kucan, J. 1996, "Notification Servers for Synchronous Groupware", ACM 1996 Conference on Computer Supported Cooperative Work, pages 122-129, http://www.lotus.com/research

Richardson, T., Stafford-Fraser, Q., Wood, K. & Hopper, A. 1998, "Virtual Network Computing", *IEEE Internet Computing*, Vol. 2 No. 1, January/February 1998

Tanaka, Y. 1996, "Meme Media and a World-Wide Meme Pool", *ACM Multimedia'96*, Boston MA, USA, November 18-22, 1996

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