COMPUTERS AND BIOMEDICAL RESEARCH 26, 1–27 (1993)

Xconf: A Network-Based Image Conferencing System

PETER F. LEMKIN

Image Processing Section, Laboratory of Mathematical Biology, National Cancer Institute, DCBDC, Frederick Cancer Research and Development Center, Building 469, Room 150B, Frederick, Maryland 21702

Received October 31, 1991

People often need to get together to share and discuss small amounts of image and textual data, but this is difficult when they are not located in the same place. One solution to this problem is Xconf, a multimedia computer conferencing groupware tool using existing national and international networks (the Internet). Simultaneous conferencing supports real-time interaction between multiple remote computer displays. Xconf multimedia may include conversational text, images, pointers to objects in images, and group execution of programs. Conferencing may take place with or without images. Interaction is tightly coupled with all users aware of global changes to the shared session and alternatively, individuals may monitor a specific subgroup of other users to concentrate on what they are discussing. Collaborative groups who have access to both computer networks and networked based X-Window System graphics displays can participate in a conference. Xconf is an X-Window "client" program which provides multimedia conferencing support for a group of X-Window displays. Because it is centralized, no software other than the standard X-Window System is required on any of the participants display systems. Key data structures and algorithms for image conferencing are present.

1. INTRODUCTION

Working groups often need to get together to share and discuss small amounts of image and textual data, but this is difficult when group members are not located in the same place. Leebaert (*1*) suggests a shrinking of roadblocks to group communication:

> Technology dissolves particularly as we move ever closer to real-time interaction. Newton might wait a month to get a response to one of his hypotheses. Argument is now conducted almost the same way whether it is across the planet or in one room.

Efficient collaboration within a group depends on sharing information in a timely manner. Telephone conferencing is not adequate because 2D information is not communicated well verbally. Electronic mail (E-mail) and Fax are two solutions but are subject to delays. Such "time-shifted" communication does not have the immediacy of a conference, E-mailed images require additional visualization software on the receiving end, and Fax is limited in quantitative accuracy. In

0010-4809/93 $5.00

general, a considerable effort is required for groups of people to get together physically when they are not in the same general location. Limiting factors include wasted time, expense of traveling, and simply human inertia. The result is that collaborative groups do not get together at all or do so less often than they desire.

Image Conferencing as a Collaborative Tool

This paper describes an inexpensive solution to this problem of sharing image information from a distance, the Xconf program, (*2*) first reported at a biological imaging conference. It is a simultaneous multimedia computer-conferencing system using existing national and international networks [the Internet (*3, 4*)]. It may be used with collaborative groups who have access to both computer networks and the networked-based X-Window System (*5*) graphics displays. Image data may be supplied to Xconf in machine readable form in several formats and since many inexpensive digital scanners are available, acquiring data for a conference is easy.

The Modality of Image Conferencing

The primary thrust of this paper is in describing the modality of image conferencing using a single program rather than requiring special programs on each remote display. We will present (in section 4) a few of the data structures and key algorithms used to implement it under X-Windows. However, that section may be skipped by those primarily interested in the image conferencing modality rather than its implementation.

Computer conferencing is the "medium" for which conferees supply the "messages." As with any medium, there are limits in what types and amount of information may effectively be exchanged. We will be describing the image conferencing problem and indicate some of these limitations throughout the paper. The Xconf user interface is also briefly presented since the ease of interaction is a key determinant of meeting our goal of passing the message without the medium getting in the way.

The motivation for Xconf came from our desire to collaborate with other groups of investigators using the GELLAB-II system (*6–8*). GELLAB-II is a 2D electrophoretic protein gel database software system using X-Window developed in our laboratory. [A 2D protein gel separates proteins in a sample by their apparent isoelectric point (similar to pH) and apparent molecular weight.] It allows one to compare individual polypeptide (protein) spots between sets of protein sample gels that may contain many thousands of different proteins. These gels are created under different experimental conditions so the goal is to find sets of proteins which correlate with these differences. Results are presented as images, graphs, and tables. We built Xconf to simultaneously connect remotely located collaborators to the same 2D gel database to investigate that database together as if they were in a small conference.

Image Conferencing for Biological Domains

Although we developed Xconf to do image conferencing for a specific biological problem, it is applicable to many other domains which require moderate image resolution. Image conferencing deals with images and their manipulation and dynamic visualization using flickering, movies, and other techniques.

The concept of flickering images, used in Xconf for comparing two images and for making movies, was previously used for comparing 2D protein gels with the FLICKER (*9*) and X-Window-based Xpix GELLAB-II programs. Flickering is an interactive image comparison method for aligning two images which were not aligned at the time they were scanned or generated. This situation also arises when comparing images which have nonuniform "rubber-sheet" distortion and are thus only locally alignable. Many problem domains produce misaligned or distorted images which flickering may help analyze. These include those produced by gel electrophoresis (both 1D and 2D for protein, RNA, and DNA materials), serial section images produced by various microtome methods, and problem domains which lose alignment during data acquisition.

A recent note (*10*) lists several areas in biology where movies have proven useful. These include: medical Xrays, MRI and PET scan images, and visible light and electron microscopy. Three-dimensional shaded images may be computed or "rendered" for viewing molecular models, organ structures, etc.

High resolution image images are required for some problem domains, such as diagnostic X-rays. However, for problem domains where high resolution viewing is required for part of an image, subregions could be extracted, zoomed, or color-mapped to give high resolution for those critical parts of the data.

Multimedia Conferencing—A Brief Review

Computer conferencing has been used for several years in teaching and medical diagnostic imaging as an instance of *groupware*. Groupware is a means of carrying out a focused activity among a group (*11–14*). Medical imaging typically requires expensive computer resources and special high speed networks to be usable for making exacting diagnostic decisions on high resolution images (*15–17*). Recently, many institutions have become connected by moderate-bandwidth national and wide area international networks. A special issue of *Scientific American* (September, 1991) is devoted to communications, computers, and networks. It shows how networked groupware is changing work habits (*4, 18, 19*). One network, called the Internet, has provided the basic infrastructure required for moderate resolution image conferencing. By high resolution we mean images on the order of 2K $\times$ 2K $\times$ 12-bit pixels/image (i.e., picture elements) and moderate resolution being on the order of 512 $\times$ 512 $\times$ 8-bit pixels.

Many of the conferencing and multimedia concepts used in Xconf are discussed in general in (*1, 11, 15–17, 20–26*). Multimedia conferencing involves different disciplines including man–machine interaction, group psychology

(conferee behaviors and attitudes toward Computer-Supported Cooperative Work), computers, graphics, image processing, networks (accessibility, bandwidth, and synchronization), and databases. It is necessary to pull aspects of all these areas together to construct a useful conferencing interface. Some of the issues which need to be addressed in a conferencing system are: groupware, simultaneous conferencing, types of multimedia, the dialog management and action scheduling problem, the remote display synchronization problem, image resolution domain dependencies, navigation through a session, required operations, networks, bandwidth, efficiency, and integration with databases.

In a comprehensive paper, Ellis and co-workers (*11*) review many aspects of the area of computer conferencing called groupware. Computer conferencing in turn is part of the field called Computer-Supported Cooperative Work, for which several conferences have been held. Among several definitions for groupware, they quote Gruden (*27*) who defines it as "software for small or narrowly focused groups, not [for] organization-wide support." Ellis *et al.* note that software which is a candidate for groupware may be viewed "as the class of applications, arising from the merging of computers and large information databases and communication technology. These applications may or may not specifically support cooperation." They then define groupware as software that supports a group's common task by providing an interface to a shared environment in which several areas are required for supporting group interaction: communication, collaboration, and coordination. They suggest that "effective collaboration demands that people share information." Groupware, then, is the *tool* that would help the group do this. Coordination implies an environment "that unobtrusively offer[s] up-to-date group context and explicit notification of each users' actions when appropriate." This effectively helps coordinate actions within the group and minimizes conflicting or repetative actions.

Software which people *want* to use may not be created in isolation. Kling (*12*) reviews many social issues of a computerized society including using computers to improve communication within cooperating groups. He suggests that efficiencies gained with specific computer tools may not be imposed on a group—other factors may play a role which lessens the desirability of those efficiencies. He raises other issues such as assumptions about social interactions when a group is presented with a groupware tool. Social conditions must be right for people to want to use it and to cooperate in using it. Social conditions—not just a program's features—determine its usefulness.

Related to this, Hiltz and co-workers (*21*) compare the computer communication (CC) process and the face-to-face (FtF) communication process. They find that FtF is more efficient—although not as economical in both participants' time and expense (travel, etc.). Even though it is less efficient than FtF conferencing, CC provides a facility for short, quick conferences that might not take place otherwise. Other issues such as synchronizing students' answers in response to a teacher's request are discussed by Hiltz (*22*) for summarizing group consensus. She also discusses a user's level of comfort or discomfort in using CC vs FtF and the level of difficulty encountered in leading a CC session.

Goals of Collaborative Conferencing

Based on our perception of simultaneous conferencing, we have set several goals for our image conferencing system. These are: (1) using X-Windows because it is a standard and has multiple-display capability; (2) sharing conversational text from all participants so it is viewed simultaneously by all, making for rapid response to points or queries; (3) comparing images from a picture file database viewed by all in which "push-pin" marks may be placed in each image by all users to point to different parts of objects for discussion; (4) aligning subregions of any two image frames by interactive flickering which involves moving one image relative to the other; (5) viewing and commenting on user-created movie image sequences; (6) sharing windows generated by users' local software and exporting them to all other conference users; (7) running application programs such as database programs with all users seeing the keyboard interaction as it occurs.

In our design, text is shared at the keystroke level rather than at the line or paragraph level. Each character typed by any user is immediately visable by all users. This lets everyone be more aware and involved as they see ideas develop. Meeting goals (3) to (5) lets individuals share thoughts on image data.

An implication of goal (6) is to allow *any* user to import a snapshot of *any* window as a conferencing system image window. Because it is just another image window, we may stick pins in it, flicker it, or make movies with it. This capability allows us to share other software which generates X-Window graphics (e.g., molecular models, image processing, database graphics systems, etc). Because of bandwidth limitations, our conferencing design does not currently update its copies every time the original window changes—only when it is directed to.

The last goal (7), is met by providing a shared Unix shell to the conference to run *any* Unix program under user keyboard control. This user program might result in X-Windows being displayed on the user's system which could then be selectively exported to the *entire* conference.

2. Xconf—Multimedia Image Conferencing System

Xconf was created to meet these goals. Although it provides less image resolution capability than that required by diagnostic medical imaging systems, it is sufficient for other problem domains. Because it does not need special hardware and networks, it may be used on a much wider range of existing hardware and networks without *any* change to existing hardware or software. Only one computer in the conference needs to run the actual Xconf program.

It works by connecting users together to create a real-time conference which takes place simultaneously on all remote displays illustrated in Fig. 1. Displays from different computer vendors can work together because of the portability of the X-Windows System. Interaction is tightly coupled so users are always aware of global changes to the conference session. In addition, any user may monitor subgroups of users' conversations in local windows. By allowing con-

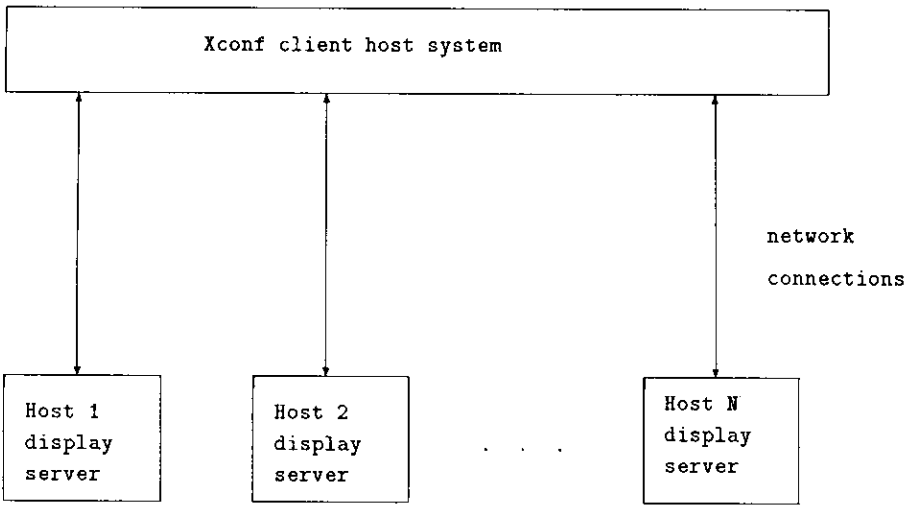

FIG. 1. Xconf-centralized computer conferencing client–server model. Each host display is either an X-terminal or a computer workstation display (B&W or color) with a mouse and keyboard. The Xconf client must be a computer but need not have a display. It is the central hub that broadcasts all window changes to all the remote host display servers connected via networks. Xconf functions as a conference transaction processor by responding to interaction on each display and redisplaying information on all of the other displays. Each user can type on their keyboard, move their mouse, press the mouse buttons, or press "pushbutton" commands on the screen to control their local session. When users in turn acquire permission, as *Master* user, they can control the global session.

ferees to concentrate on what they are interested in, this may help minimize distractions and avoid overloading participants with extraneous information.

Multimedia here include conversational text, text files, grayscale or color images, movie sequences of images, pointers to objects in images, and the abstract notion of group execution of programs. Computer conferencing may take place with or without images depending on the needs of a particular conference.

X-Window System and Computer Conferencing

The popular MIT X-Window System provides the basis for Xconf by its client–server computing model. In X-Windows, a program called the *client* generates display window data. This window data is then automaticaly sent to another program called the *display server* which displays it, thereby "serving the client's need for a display." A display server may be an X-terminal or a workstation (display, keyboard, and mouse hardware) running the X display server program. These two programs may be run on the same or different machines called *host* system(s) because they are hosts for those programs. When the client and server programs are on different computers, a network

is used to connect them. The X display server is started when X-Windows is started on a display, normally when a user logs onto their workstation.

In addition, the X-Window client server mechanism permits windows created by a client program on one host system to be displayed on several X display server host systems (*28*). This latter capability is required to implement simultaneous conferencing systems such as Xconf. Under X-Windows, data is transparently passed from one system to another using standard TCP/IP or DECnet protocols. A client program accesses X-Windows through procedure calls to the basic X-Windows "Xlib" library or higher level "widget" tool kits. We wanted efficient simultaneous control of many displays, so Xlib rather than a tool kit was used with our initial Xconf design. In future versions of Xconf, we will probably use the X Windows tool kit "Xt" using the methodology described by Jones (*29*) in a multiple display voting application.

To simplify this discussion, when we mention display, participant, host, or user, we are referring to a remote X-Window display server host system. The client refers to a single host system running the Xconf program and need not have a display itself. For example, Xconf could be run on a computer holding an image database. It is possible for a user to participate in multiple conferences by running several copies of Xconf with their display included in each conference. However, because of limited display screen space, memory, network bandwidth, and probability of the user being confused, this is not very desirable.

Centralized vs Decentralized Conferencing

A network-based conferencing system may be either centralized or decentralized (*21*). Both methods have advantages and disadvantages, and we will be discussing some of the issues throughout the rest of this paper. A major advantage of using a centralized client system such as Xconf is that no additional software is required on the remote X-Window display servers to support the client program. A major disadvantage is that it limits the effective bandwidth of interaction and thus the type of operations which may be performed quickly.

In order to participate in a centralized conference, each user needs to allow the central Xconf client host access to their display. This is normally done using the standard X-Window System tool called **xhost.** However, Xconf is able to add additional user displays during a running conference and let users leave the conference at any time without disrupting the conference.

Simultaneous Conference Schema

Face-to-face conferences are by definition simultaneous. However, because of store-and-forward computer technology (e.g., E-mail), it is possible to have nonsimultaneous conferencing which is a form of time-shifted communication. Both modalities may exist in the same system. For example, BBN/Slate (*30*) is a document design conferencing system which may be used for either simultaneous conferencing or time-shifted mailing of multimedia documents. Sun Microsystems has a product called ShowMe which lets users on a network view

and annotate an image of a spreadsheet, document, or graphic. Other products include Communique from InSoft for Sun computers and FarSite from DataBeam for PCs. Many of these human factors issues are reviewed in (*11, 15, 21*).

The simultaneous conference schema used in Xconf shows the same global windows to all users. All communication between users is controlled by a central client program that behaves like a switchboard. Figure 1 illustrates the centralized conference Xconf client schema which handles transactions from a set of remote host display servers. A noncentralized conferencing system might distribute computational and image storage capabilities across the network on remote display host computers. That would result in minimizing network bandwidth requirements. However, this would require more software at each collaborator's computer and is beyond our primary goal of a simple system.

3. Examples of a Conference

An example of a four-user session with two images is shown in Fig. 2. Global windows include: a collective user conversation *input window* in the upper left corner where the set of current partial input lines of all users are displayed for carrying on conversations; a scrollable *session window* in the upper right corner containing output from Xconf as well as completed lines from the input window; and a small conference-control status *icon window* at the top of the screen. An optional scrollable *pop-up window* with the same format as the session-window (not shown) may be used for local help, information, status, and sub-session logging information.

Another example user session in Fig. 3 shows a conference session with a navigation image–icon map of all 2D gel images currently in the conference. Figure 4 shows a three user conference session with two magnetic resonance imaging heart images and a navigation image–icon map of the five images in the database. Figure 5 shows a three-user conference session with two frames from an 18-frame crystalographic protein molecular model showing 20° differences in a 360° rotation with all frames shown in the navigation map.

When image files are specified at the time Xconf is started, undisplayed windows containing these images are created and downloaded into each user's display. This is done *before* the conference starts thereby avoiding distractions while the conference is being created. Once started, images may *then* be sequentially displayed, flickered, or used for movies by animating sequences of image frames, or objects in these images may be "pin-marked" by users. After the conference is initialized, it displays only the icon window on each remote display. Users who wish to join or leave the conference click on their icon window. Alternatively, after the conference has started we may specify that images be read from files on the Xconf client system and added to the conference. We may also export non-Xconf windows from any conferee's display into the conference.

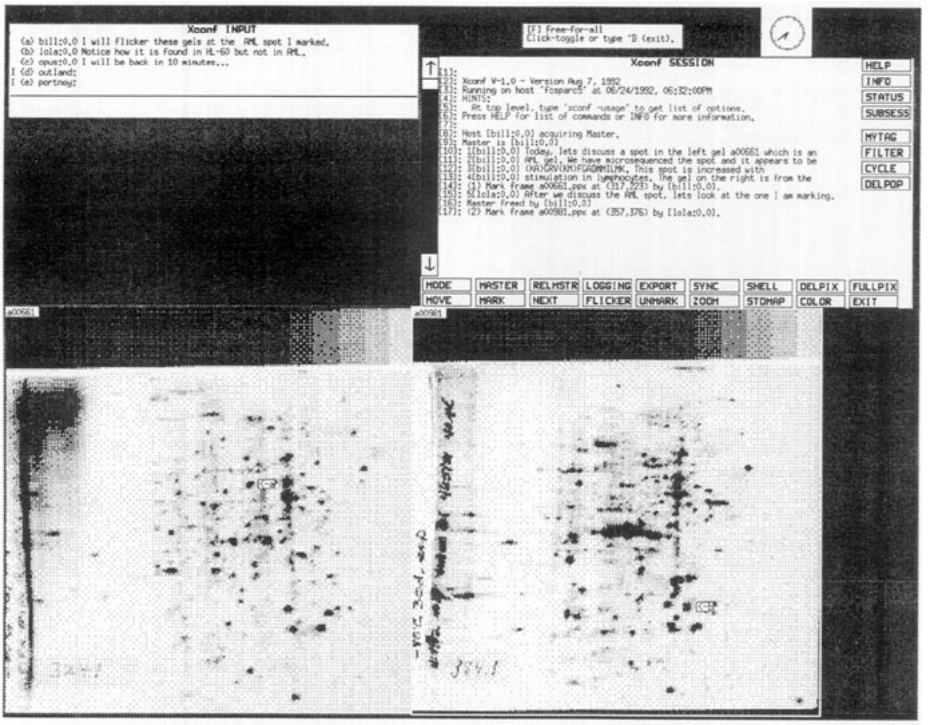

FIG. 2. Screen dump of a five-user conference. An icon window at the top of the screen shows the status of the conference (which is Flicker mode—the **[F]** indicates Flicker mode and user **bill** is the Master user) and gives instructions for entering or leaving the conference. The current users' keyboard inputs are shown in the **Xconf-INPUT** window (including their typographic errors!); combined session information is in the **Xconf-SESSION** window that has scrollbar control. Three sets of control pushbuttons: local (right column), global (top row), and mode-specific (bottom row) are available in the **Xconf-SESSION** window. Two 512 × 512-pixel image frame windows of 2D gel electrophetic gels (lymphocyte samples from a leukemia gel data base (*32*)—acute myelocytic leukemia on the left and HL-60 cell line on the right) are shown on the bottom part of the screen. These have several colored *pin* mark labels inserted by different users "<-a" for user **bill** and "<-b" for user **lola.** The names in the pin labels correspond to the names in the table entries {a}, {b}, etc. in the **Xconf-INPUT** window. The coordinates of the pins are logged in the session window. Conversational text and operations are logged in the session window labeled by host name tag, e.g., **[bill], [lola], [opus], [outland], [portnoy].** Flicker mode allows locally aligning two images for comparison purposes by alternately displaying one and then the other while moving one relative to the other. [Note: actual grayscale screen resolution is much better than shown in this screen dump—dithering was used when dumping the screen onto a bitonal laser printer.]

Conference Modes

There are three conferencing modes: conversation-only, flicker, and movie. A conference may be switched back and forth between modes during a conference. Conversation-only mode does not show or require any images. Flicker mode allows alignment of images while movie mode permits interactive creation and showing of "film clips" of many images.

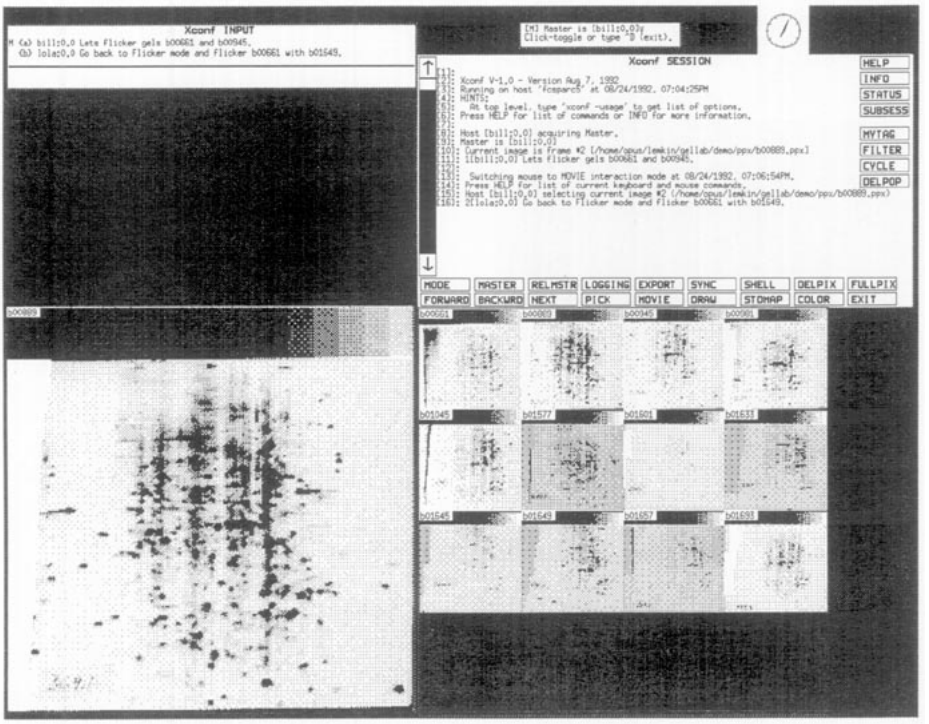

FIG. 3. Screen dump of a multiuser conference in movie mode. This illustrates a navigation mosaic icon map consisting of 12 image frames of different lymphocyte sample gels from a human leukemia gel database (*32*). The mosaic map helps users navigate through the set of images by seeing what images are currently available to the conference for flickering, making movies, etc. The images in the database are represented by sampling images into small panels. The 512 × 512-pixel 2D gel images could be sampled by 2 resulting in the 256 × 256 size images when network height bandwidth is not available. [Note: actual grayscale screen resolution is much better than shown in this screen dump—dithering was used when dumping the screen onto a bitonal laser printer.]

All of these windows except for the pop-up window are considered global to the conference and are displayed simultaneously on all remote displays. This helps preserve a conference's cohesiveness. Each user may control their own local scrollable pop-up window. The latter may be used for additional conference displays specific to that user *not* visible to other users.

Control of the Conference

Any user may control the conference by typing, moving, or pressing the mouse buttons from their display. The default control is a "free-for-all" with no one in charge. This conference control may temporarily be restricted to one user by their becoming the *Master* user. Other users may request access to become the Master and are put into a queue. When the current Master relin-

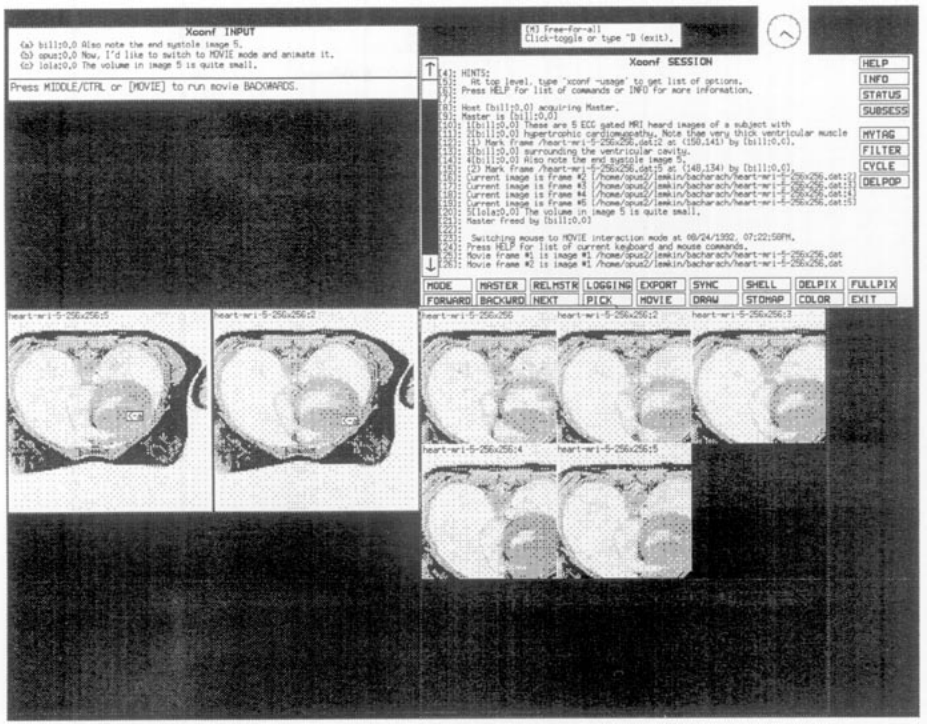

FIG. 4. Screen dump of a three-user conference using five MRI heart images graciously supplied by Steve Bacharach of NCI. These are transaxial, ECG-gated images of the heart in a subject with hypertrophic cardiomyopathy, i.e., a very thick heart muscle. They were made with a GE 1.5-T nuclear magnetic resonance scanner. You can see the very thick ventricular muscle surrounding the ventricular cavity. The images span the time from end diastole (image 1) when the ventricular chamber is largest and most filled with blood, to end systole (image 5) when the blood has been pumped from the ventricular chamber and it is very small. When viewed as a movie sequence it is possible to observe the thickening of the myocardial muscle with time, and to visualize the contraction of the ventricular chamber (causing expulsion of the blood from the cavity into the aorta). The standard map image on the right shows the five images in the database. Image frames 1 and 5 are shown on the left with the *pin* mark labels inserted as "<-a" for user **bill.** The images were 256 × 256 × 8-bit pixels. [Note: actual grayscale screen resolution is much better than shown in this screen dump—dithering was used when dumping the screen onto a bitonal laser printer.]

quishes it, the next user in the queue gets it or if there are none it reverts to a free-for-all. Having a Master user is advantageous when teaching new users a computer conferencing system since new users may only "listen" instead of randomly pressing keys or buttons. If further control is desired, a specific user may be declared the *Moderator* for the duration of the conference and may take control away from the current Master at any time. The Master may be thought of as a baton that may be passed from user to user to take turns "having the floor."

The paradigm is that any user may converse or manipulate text or images at

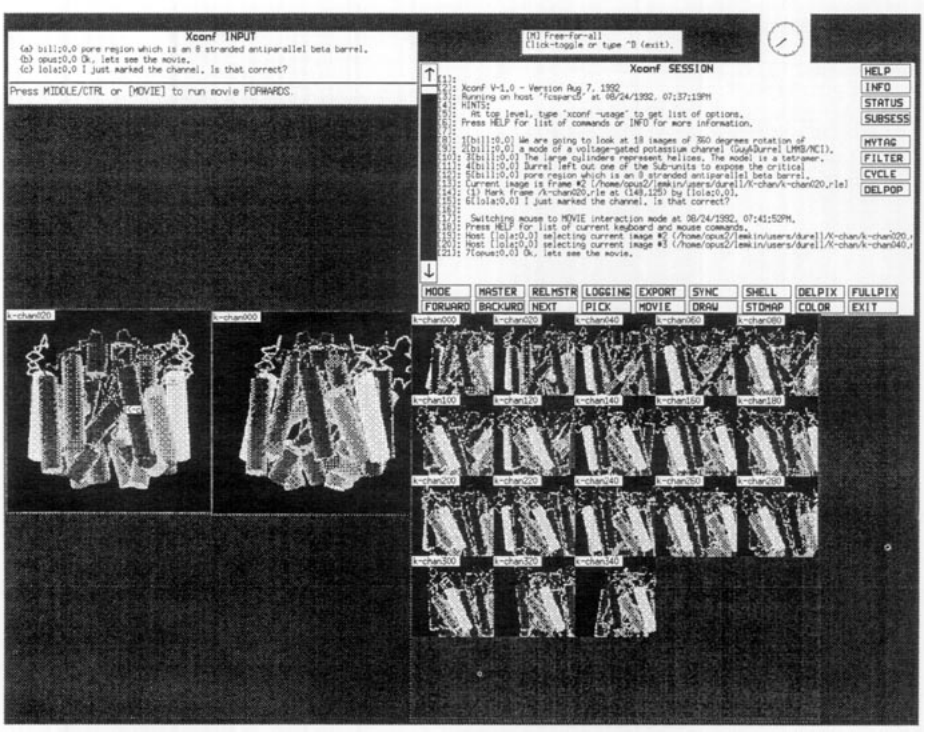

FIG. 5. Screen dump of a three-user conference using 18 view images computed for a 360° rotation of a protein crystalographic model of votage-gated potassium channel graciously supplied by Bob Guy and Stewart Durell of LMMB/NCI. The large cylinders represent helices. The model is a tetramer. Stewart left out one of the subunits to expose the critical pore region which is an eight-stranded antiparallel beta barrel. This is also the closed conformation of the channel. Note that **lola** pin marked with a "<-c" in image **k-chan020** where she thinks the opening is and can query the other users to immediately verify her question. The standard map image on the right shows the 18 images in the database. The images are 512 × 512 pixels but were sampled by 2 in order to show two views adjacent view **K-chan020** and **K-chan140.** [Note: actual grayscale screen resolution is much better than shown in this screen dump—dithering was used when dumping the screen onto a bitonal laser printer.]

any time. However, more restrictive session control may be invoked when needed to limit this free-for-all dialogue using Master control. Since all users always see the same global windows, it is necessary to show which user is the Master and which users have caused which actions. All user actions are tagged in the various control-icon, input, session, and image windows with "name tags." These are host names or, alternatively, names defined by the users (see Fig. 2).

Xconf Benchmarks with Small Groups

We conducted a number of tests on the usability of Xconf on wide area networks. In one test, Xconf was used over the Internet to conference 12 users at a time in conference-only mode with satisfactory (but not spectacular)

response times of about 1 sec. Conferees were located both in North America and in Europe. In other tests, we have used up to 18 images at varying resolutions (512, 256, 128, and 64). Depending on the detail of the structures in the different types of image material (2D gels, MRI heart images, molecular protein models) and what is going to be discussed, more or less resolution is required for the conferencing to be usable. We have concentrated on testing the system on 2D gel image data, generally in 2-person conferences, between Frederick, MD and Atlanta, GA using an Internet connection restricted by a shared 56-Kb line. It was found usable—although not blindingly fast. It has been used over the last year from time to time to augument E-mail, and when we need to quickly converse about text or image data.

Before making users aware that they are in the conference, Xconf first downloads the images to all user host system displays. This is the major time delay in running an intercontinental conference, with data rates of 1 to 3 Kbytes/sec. However, once the images are in place on all the remote display hosts, interactive conferencing may then proceed with response times on the order of 1 sec. The response is rapid because display server resident images and other windows may be moved, mapped, and unmapped without the client system having to redraw them over the network—the remote X servers do most of the work. Conferences on local area networks would of course have much better initialization and response times.

4. Description of Xconf Algorithm

Although we cannot describe the Xconf algorithms in detail here, we will try to sketch out the basic data and control structures needed to understand how it works. A knowledge of C, UNIX, and X-Windows is assumed. This section may be skipped by those interested in the modality of image conferencing but not in the details.

Key parts of the main data structures and algorithms are presented that are necessary to understand the issues involved. First, let us examine portions of the four primary data structure *typedef* definitions (many details have been omitted). Space limits our discussion, so comments on other variables indicate additional functionality.

host_t host display instance database
xconf_t global database for the entire conference
scroll_t host specific instance of scroll buffer database
import_t instance of imported window from specific host

```
typedef struct __HOST_T__
  { /* HOST_T */
    STRING displayName;          /* full display name */
    STRING hostName;             /* host TAG name: "e.g. joe from joe@pickle.edu */

    /* X-server specific stuff */
    Display *dpy;                /* Display struct of the host */
    int fdHost;                  /* file descriptor of host connection */
    /* Major modes */
    int requestForAccess;        /* if waiting for Master access */
    int masterIcon;              /* if this host is iconisized */
```

```
    /* Replicated frame windows; and pixmaps if used (i.e if no backing store). */
    Window *imageWindow;          /* [MAXFRAMES+1], images */
    Pixmap *imagePixmap;          /* [MAXFRAMES+1], pixmaps of images */
    int *ignoreImageExposeCnt;    /* [MAXFRAMES+1], ignore expose events if >0 */

    XImage **image;               /* [MAXFRAME+1] data for imageWindow[] */
    XRectangle *imageLoc;         /* [MAXFRAME+1] size of imageWindow[] */

    Window markWindow[MAXHOSTS+1][MAXFRAMES+1]; /* imageWindow[frames] is parent*/
    Pixmap markPixmap[MAXHOSTS+1][MAXFRAMES+1];
    Window inputWindow;           /* INPUT window for all keyboard echos */
    Pixmap inputPixmap;
    Window masterWindow;          /* MASTER ICON window */
    Pixmap masterPixmap;
    Window sessionWindow;         /* SESSION conf-wide output window */
    Pixmap sessionPixmap;
    Window sessionScrollBarWindow; /* SESSION scrollbars */
    Pixmap sessionScrollBarPixmap;

    /* Popup windows which are only local to THIS host instance */
    Window scrollPopupWindow;     /* optional - updated only on this host */
    Pixmap scrollPopupPixmap;
    Window popupScrollBarWindow;  /* HELP, INFO, STATUS, SUBSESSION scrollbars*/
    Pixmap popupScrollBarPixmap;

    /* Command buttons which are children to sessionWindow; */
    Window buttonWindow[MAXRADIOBUTTONS+1];
    Pixmap buttonPixmap[MAXRADIOBUTTONS+1];

    /* Color stuff */
    int useVirtualColormapFlag;   /* if THIS host uses Virtual colormap */
    Colormap color_map;
    X_PIXEL lookup[257];          /* 8-bit pixel to 7-bit image map table */
    X_PIXEL image_forground, mark_forground, session_forground, input_forground,
            master_forground;
    X_PIXEL hostColor[MAXHOSTS+1]; /* modulo: RED, YELLOW, GREEN, CYAN, BLUE*/
    GC *host_gc;                  /* [MAXHOSTS+1], host specific color pixel */

    /* Scroll data structures */
    scroll_t scrollPopup[MAXSCROLL_POPUPS+1]; /* popup scroll DB's */
    int curScrollNbr;             /* exists if > 0 */

    /* Input window  data for THIS host */
    int newLine;                  /* TRUE if last char of iline was '\n' */
    STRING iLine;                 /* keyboard input for this host */

    . . .
  } host_t;
```

The main struct containing global variables is **xconf_t** which has a single global instance called **xconf.** All access to host specific objects is through the **xconf.hList[host#]** data structure.

```
typedef struct __XCONF__
  {
   int glbArgc;                 /* Shell cmd line args */
   STRING *glbArgv;
   STRING *glbEnv;              /* host environment variables */

   char clientHostName[256];    /* host name of Xconf client (me) */
   STRING helpMsg;              /* changes with mode: Conf, Flicker, Movie */

   /* Host specific DB structure */

   host_t *hList[MAXHOSTS+1];   /* host specific DB instance */
```

```
/* Numbers of host structures and current # active */
int nHosts;                 /* current # of hosts ever in conference */
int firstHost;              /* set to LAST host which was select()'ed */
int nDisplaysActive;        /* current # of active hosts, <= nHosts */

/* Movie state of 'all possible' movie frames */
STRING movieList[MAXFRAMES+1]; /* all images available for making a movie */
int maxFrame;               /* # of all image frames */
int currentFrame;           /* current frame of interest (i.e. 'NEXT') */
int syncFlag;               /* use XSync() instead of XFlush() when flush*/

/* Frames used in the 'actual' movie */
int lastWasFullMovie;       /* TRUE if use all frames in movie */
int maxMarkedFrames;        /* # of frames in the movie */
int markedFrames[MAXSUBMOVIEFRAMES+1]; /* frames subset for PICKed movie*/
int mosaicMapSize;          /* size*size ~< maxFrame */

import_t *importExport[MAXFRAMES+1]; /* setup if EXPORT image is used */

/* Conferencing 'modes' */
STRING currentMode;         /* either: "FLICKER", "MOVIE", "CONF-ONLY" */
int hostDoingGrab;          /* if not 0, then this host is doing a grab. */
int activeWindows;          /* TRUE when windows are usable for updating */

/* SHELL command 'csh' data shared by the conference */
subShell_t *sh;             /* for Unix csh SHELL operation */
STRING lastShellCmd;        /* last SHELL command string if any */

/* Input-line/Output-scroll buffers */
scroll_t *sessionScroll;    /* scroll buffer for session window*/
STRING scrDBname[MAXSCROLL_POPUPS+1]; /* Scroll database titles */

/* Master icon window messages */
STRING iconMsg;             /* "[mode]CONF-Request from <client-host>" */
STRING clickMsg;            /* "Click-toggle or type ^D (exit)" */
STRING masterMsg;           /* "Master is <current-master-hostname>" */

/* Master state variables */
Display *masterDpy;         /* is a master if not NULL */
int lastAccessGranted;      /* 0=none, else the index of Master host */
int masterFIFOqueue[MAXHOSTS+1]; /* hosts in Master queue requesting access */
int masterFIFOfirst;        /* name of next host in queue */
int masterFIFOlast;         /* name of last host in queue */
int masterInactivityTimer;  /* release master if not active > MAXINACTIVITYTIME seconds */

/* Last coordinates for host "Pin-Mark" window for each image frame if active. */
int isVisibleMarkWindow[MAXHOSTS+1][MAXFRAMES+1]; /* if mark window is visible */
int xMark[MAXHOSTS+1][MAXFRAMES+1];
int yMark[MAXHOSTS+1][MAXFRAMES+1];

/* Radio and command style buttons */
int maxRadioButtons;        /* # of radio style push buttons */
STRING radioButtonName[MAXRADIOBUTTONS+1]; /* label of buttons */

/* Timers*/

long totalConfTimeMsec;     /* real-time timer for whole conference */
long localConfTimeMsec;     /* real-time local timer for Master timeout*/

/* X Protocol Error handler state if an error occurs */
int xProtoErrCode;          /* Success if ok */
int xProtoErrHost;          /* 0 if none else host # */

  . . .
} xconf_t;
```

Since Xconf was written in Xlib and no scrollable Xt-based widget was used, we needed to implement a scrollable text window for the SESSION and other pop-up scrollable text windows. Text may be independently scrolled on each host's pop-up scrollable windows. Each host has its own set of data structures defined in **host__t** as **scroll__t scrollPopup[MAXSCROLL__POPUP+1].**

```
typedef struct __SCROLL_
  {
    int isVisible;          /* TRUE if the scroll buffer was created */
    STRING title;           /* Big font title of this scroll DB */

    /* Viewable window */
    Window wind;            /* Scroll window */
    Pixmap pixmap;
    Window scrollBarWind;   /* Scroll bar window for scroll window */
    Pixmap scrollBarPixmap;

    int lastCharWasNotNL;   /* last line of buffer NOT a '\n' */
    int maxScrollLines;     /* maximum scroll buffer size */
    int nbrScrollLines;     /* top line # in scrollNbr[1:maxScrollLines] */
    STRING *scrollLines;    /* [0:maxScrollLines], dynamically allocated */
    int *scrollLen;         /* [0:maxScrollLines], dynamically allocated */
    int *scrollNbr;         /* [0:maxScrollLines], dynamically allocated */
    int scrollPosition;     /* where the viewable scroll display is now */
    int scrollNbrReq;       /* last viewable scroll number requested */
    int pixelsPerChar;      /* of a character in pixels */
    int pixelsPerLine;
    int scroll_width;       /* of viewable scroll window */
    int scroll_height;
    int maxViewableLines;   /* maximum viewable buffer size */

    /* Scroll filters: pass input from hosts to scroll window through filter tag list */
    int nFilters;                   /* maximum # of active filters if NEQ 0 */
    STRING filterTag[MAXHOSTS+1];   /* [1:nFilters] are active */
    int filterLen[MAXHOSTS+1];      /* [1:nFilters] are active */

    . . .
  } scroll_t;
```

When a user exports an image window from their display into the conference, Xconf needs to get and analyze information from that host on that window and its colormap. This analysis is saved in an **import__t** instance in the **xconf__t** struct. Xconf does an **XGetImage()** call as well as getting and analyzing the colormap associated with that window. Then, a copy of the image is mapped to each host's newly created image instance using a pixel mapping between import and export (i.e., **xconf.hList[host]->color__map**) colormaps.

```
typedef struct __IMPORT__
  {
    /* Information on window used for import-export */
    int iHost;                  /* host where import window came from */
    Window importWindow;        /* XID of window imported at iHost */
    Colormap importColormap;    /* imported colormap XID */
    int iWidth,                 /* size of original import window */
        iHeight,
        iFormat;                /* XYpixmap, Zpixmap */
    XImage *iImage;             /* data from original import window which
                                 * may be reanalyzed if remap colors */

    . . .
  } import_t;
```

As noted, each host display's state is saved in a **host_t** data structure instance. Therefore we need a list of such hosts and this is found in the global data structure list **xconf.hLists[1 : nHosts].** When started, a list of displays to be used in the conference is checked using the **XOpenDisplay()** call to check accessibility. Those hosts which are accessible from the client system have **xconf.hLists[+ + xconf.nHosts]** entries created while those which are not are dropped from the conference. Then, each type of window (master icon, input, session, images, marks) is *created but left unmapped* for each accessible host. However, the first display in the list of displays—generally the person starting the conference—does have these windows mapped as they are created to track the progress of the conference broadcast initialization during conference setup. Only after all windows/pixmaps are in place on all hosts are the master-icon windows mapped on all of the hosts. Then as conferees click on their master icons, the other windows are correspondingly mapped as the users join the conference.

The key process in conferencing is to asynchronously service X events from any host as well as from timers and other asynchronous I/O devices. We now describe the basic conferencing multiplexor algorithm suggested in (*28*) and outlined here with our extensions required for image conferencing.

The way to think about these data structures is: (1) information which is visible on all displays is global information (e.g., text data for shared SESSION windows); (2) information which is host specific (e.g., particular colormaps used for a particular host or positions of a local pop-up scrollable window), will be in the specific **host_t** data structure instance.

When a particular window needs to be updated on all hosts, it is broadcast using code like the following example. Note that only active hosts are updated. This is necessary in case a host leaves the conference or the network connection is broken. Since a host may leave the conference at any time, we need to check this each time any attempt is made to update all hosts.

```
int host;
for (host=1;host<=xconf.nHosts;host++)
  if (xconf.hList[host]->dpy!=NULL)
      updateLocalMsgWindow(host,
            "Press MIDDLE/CTRL or press MOVIE to run movie FORWARDS.\n");
```

Since, the conference, is event driven, the event loop must figure out which host has caused an event and then dispatch to service it appropriately. We use the Unix **select()** system call to handle asynchronous I/O. This must be initialized to include not only the X display hosts but also a timeout for background processing and possible UNIX subshell pseudo teletype I/O.

```
int setupXSelects()      /* RETURN: nfds - highest # file descriptor */
{/*SETUPXSELECTS*/
  int h, rb,
      nfds= 0;
  long buttonOnlyMask= (ButtonReleaseMask | ExposureMask | SubstructureNotifyMask);
  long fullMask= (ButtonPressMask | ButtonReleaseMask | PointerMotionMask |
                  KeyPressMask | ExposureMask | SubstructureNotifyMask);
```

```
    /* Get all file descriptors for working display connections. */
    for (h=xconf.firstHost;h<=xconf.nHosts;h++)
      if (xconf.hList[h]->dpy!=NULL)
        xconf.hList[h]->fdHost= ConnectionNumber(xconf.hList[h]->dpy);

    nfds= xconf.hList[1]->fdHost;        /* find maximum fd value */
    nfds= MAX(nfds, xconf.sh->iCmdChan); /* add sub SHELL pty if exists */
    for (h=2;h<=xconf.nHosts;h++)
      nfds= MAX(nfds, xconf.hList[h]->fdHost);

    /* Enable events for event-loop FOR EACH HOST. */
    for (h=xconf.firstHost;h<=xconf.nHosts;h++)
      if (xconf.hList[h]->dpy!=NULL)
        {/* set up selects for host[h] */
          host_t *ht= xconf.hList[h];
          int hh, fNbr;

          if (!xconf.sw.conferenceOnlySwitch)
            {/* Doing images and mark windows as well */
              for (fNbr=0;fNbr<=xconf.maxFrame;fNbr++)
                if (ht->imageWindow[fNbr]!=0)
                  XSelectInput(ht->dpy, ht->imageWindow[fNbr], fullMask);
              for (fNbr=1;fNbr<=xconf.maxFrame;fNbr++)
                for (hh=1;hh<=xconf.nHosts;hh++)
                    if (ht->markWindow[hh][fNbr]!=0)
                      XSelectInput(ht->dpy, ht->markWindow[hh][fNbr], fullMask);
            }/* Doing images and mark windows as well */

          XSelectInput(ht->dpy, ht->inputWindow, fullMask);
          XSelectInput(ht->dpy, ht->sessionWindow, fullMask);
          XSelectInput(ht->dpy, ht->masterWindow, fullMask);
          XSelectInput(ht->dpy, ht->scrollPopupWindow, fullMask);
          for (rb=1;rb<=xconf.maxRadioButtons;rb++)
            XSelectInput(ht->dpy, ht->buttonWindow[rb], buttonOnlyMask);
          XSelectInput(ht->dpy, ht->sessionScrollBarWindow, buttonOnlyMask);
          XSelectInput(ht->dpy, ht->popupScrollBarWindow, buttonOnlyMask);
        }/* set up selects for host[h] */

     return(nfds);     /* This value will be used in the UNIX select() call. */
    }/*SETUPXSELECTS*/
```

The X events, timeouts, and possible subshell pseudo teletype I/O are detected asynchronously using the **select()** UNIX system call shown in the following code.

```
int waitForSelectOrXevent(dpy,nDisps,sh,readfds,writefds,execfds,timeout,host)
                          /* RETURN: the value returned by select() call */
    Display *dpy[];       /* [1:nDisps]. Inactive entries are NULL */
    int nDisps;           /* # of X-window displays. */
    subShell_t *sh;       /* Subshell struct - NULL if not active */
    fd_set *readfds,      /* input FD bits set before and by select call */
           *writefds, *execfds; /* not used */
    struct timeval *timeout; /* timeout variables set here */
    int *host;            /* RETURN: X-Window dpy host index if event waiting if not 0 */
{ /* WAITFORSELECTORXEVENT */
  int
    nFound= 0,
    nfds= 0,              /* highest # fd in select call */
    i;

  /* [1] Check if Xevent is pending and return immediately if it is. */
  *host= 0;               /* init it to no X-event */
  for (i=1; i<=nDisps; i++)
    if (dpy[i]!=NULL)
      {/* check for traffic on this display */
```

```
      if (XEventsQueued(dpy[i], QueuedAfterReading))
        { /* found X event for in this display's queue */
          *host= i;          /* NOTE: tells us if XEvent waiting! */
          goto finished;     /* RETURN NOW with host to process. */
        } /* found X event for in this display's queue */
      else XFlush(dpy[i]);  /* just flush display if it needs it */
    }/* check for traffic on this display */

  /* [2] Otherwise, wait for select call to unblock with input I/O ready. */
  timeout->tv_sec= 5;          /* 5 sec. timeout if no display or pty events*/
  timeout->tv_usec= 0;
  FD_ZERO(readfds);            /* Clear the select masks */
  FD_ZERO(writefds);
  FD_ZERO(execfds);

  if (sh!=NULL && sh->iCmdChan!=-1)
    {/* Add pty file descriptor since SHELL command is active */
      FD_SET(sh->iCmdChan,readfds); /* pty for subshell */
      nfds= MAX(nfds,sh->iCmdChan);
    }/* Add pty file descriptor since SHELL command is active */

  for (i=1; i<=nDisps; i++)
    if (dpy[i]!=NULL)
      { /* Add X display file descriptor */
        FD_SET(ConnectionNumber(dpy[i]),readfds);     /* X display */
        nfds= MAX(nfds,ConnectionNumber(dpy[i]));
      } /* Add X display file descriptor */

  nFound= select(nfds+1, readfds, writefds, execfds, timeout);

  /* [3] Check for errors. */
  if (nFound==-1)
    {/* TROUBLE */
      fprintf(stderr, "select() ERROR. nFound=%d errno=%d readfds[0]=0%o\n",
              nFound, errno,*((int *)readfds));
      nFound= 0;               /* try to keep going */
      goto finished;
    }/* TROUBLE */
  else if (nFound==0)
    goto finished;             /* timed out - continue */
  else if (nFound>0)
    goto finished;             /* Process async. I/O back in the event loop */

 finished:
  return(nFound);
} /* WAITFORSELECTORXEVENT */
```

Finally, we show key parts of the event loop. Once the displays are initialized, this loop is entered until Xconf is exited. It polls the displays, in a timely fashion, using **select()** and then services the conferees in a round-robin order. Processing is divided into two pairs: (1) timeouts and pty input and (2) X events. We will not show the details of the low level event service functions, but their functionality can be inferred from the comments, function names, and discussion in the rest of this paper.

```
    void doEventLoop()
    {/*DOEVENTLOOP*/
      char sMsg[256];
      int flag= TRUE;
      XEvent event;
      fd_set readfds,              /* READ fd mask for select call */
             writefds, execfds;    /* WRITE, EXEC fd masks for select call - unused */
      struct timeval timeout;
```

```
int allDone= FALSE,
    nFound,                    /* # of fds found */
    host, h;

/* [1] Setup select and event loop. */
grabMasterIfFreeElseQueueIt(xconf.sw.moderator); /* if moderator is active*/


/* [1.1] Setup all display connection file descriptors. See Oliver Jones,
 * "Introduction to X-Windows", pp 404-408, Prentice Hall, 1989.
 */
setupXSelects();                /* set up Event enables for all windows*/
xconf.firstHost= xconf.nHosts;  /* so next time we start from here */


/* [2] X-Event-Loop. Loop here forever until exit Xconf. */
while(!allDone)
  {/* here until the end of time */
    Display *dpyList[MAXHOSTS+1]; /* will contain active hosts displays*/
    Window wind= (Window) 0,    /* e.g. INPUT, SESSION window, etc. */
           subwind= (Window) 0; /* e.g. buttons or scroll bars etc. */
    int doneFlag= FALSE,
        got_one= FALSE;
      . . .                     /* more variables */

    /* [2.1] Check if any there are any events from any display to process. */
    while(!got_one)
      {/* See if any events pending */

        /* [2.1.1] Nothing found from any display connection - do the
         * select. This keeps the CPU usage down as it waits until there
         * is a display connection event, timeout or pty input.
         */
        for (h=1;h<=xconf.nHosts;h++)
          dpyList[h]= (xconf.hList[h]->dpy!=NULL)
                        ? xconf.hList[h]->dpy
                        : NULL;    /* list of current active hosts */

        /* Block waiting and checking for queued X events, timeouts, pty I/O */
        host= 0;
        nFound= waitForSelectOrXevent(dpyList, xconf.nHosts, xconf.sh, &readfds,
                                      &writefds, &execfds, &timeout, &host);

        /* Analysis of 'select' return values:
         *        | XEvent | pty | timeout
         *        ------------------------
         * host   |  > 0   |  0  |   0
         * nFound |    0   |  1  |   0
        */
        if (host > 0)
          {/* X event found for 'host' */
            got_one= TRUE;          /* so we can get out */
            goto foundActiveDisplay;  /* Process X events at [2.3] */
          }/* X event found for 'host' */

else if (nFound==-1)
    continue;                   /* TROUBLE with select call*/

else if (nFound==0)
  {/* Timeout - do some extra processing */
    /* (a) Test if Master timed out. If so, free the Master.
     * NOTE: only count down if Moderator is NOT the Master!
     */
    if (xconf.masterDpy!=NULL &&
       (xconf.lastAccessGranted!=xconf.sw.moderator))
      {/* See if time to release the master */
        xconf.masterInactivityTimer += timeout.tv_sec;
```

```
            if (xconf.masterInactivityTimer > xconf.sw.maxInactivityTime)
              { /* Ok, release master from inactive host */
                xconf.masterInactivityTimer= 0; /* reset the timer*/
                sprintf(sMsg, "Master timed out after %d seconds.\n",
                        xconf.sw.maxInactivityTime);
                updateSessionWindow(sMsg,0); /* on all hosts */
                freeMasterAndGetNextReq(xconf.lastAccessGranted); /* adv. queue */
              } /* Ok, release master from inactive host */
            }/* See if time to release the master */

          /* (b) If subshell died, turn off pushbutton & notify everyone. */
          if (xconf.sh!=NULL && xconf.sh->iCmdChan==-1)
            {/* Failed*/
              updateSessionWindow("Sub shell exited.\n",0);
              redrawAllHostsRadioButton(SHELL_RB, xconf.radioButtonName[SHELL_RB], FALSE);
              free(xconf.sh);  /* GC the struct so can realloc later*/
              xconf.sh= NULL;  /* flag as inactive */
            }/* Failed*/
          continue;
         }/* Timeout - do some extra processing */

      /* [2.1.2] Check if subshell input for us. If so post in SESSION */
      else if (nFound>0 && xconf.sh!=NULL && xconf.sh->iCmdChan!=-1)
        {/* Check for SUBSHELL input */
          if (FD_ISSET(xconf.sh->iCmdChan,&readfds))
            {/* Process pty: we have input from cmd subshell - read it */
              STRING sInputX= evalCmdIn(xconf.sh);

              if (sInputX!=NULL)
                updateSessionWindow(sInputX,0); /* copy PTY data to SESSION window */
            }/* Process pty: we have input from cmd subshell - read it */
        }/* Check for SUBSHELL input */
    }/* See if any events pending */

  /* [2.2] End of select analysis. DON'T process events by  falling
   * through here.
   */
  continue;
```

The second part of the event loop deals with servicing the X events. Since the analysis of the **select's** return values told us which host was active, we just get the next event for that display and process it. Again, we only show highlights of the code.

```
    /* [2.3] Process event. We got here from step [2.1] by a GOTO! */
  foundActiveDisplay:
    XNextEvent(xconf.hList[host]->dpy,&event); /* Get the event for 'host' */

    if (xconf.hList[host]->dpy==xconf.masterDpy)
      xconf.masterInactivityTimer= 0; /* reset timer since there IS activity */

    wind= event.xany.window;    /* window we are working on*/
    subwind= (event.type==ButtonPress || event.type==ButtonRelease)
             ? event.xbutton.subwindow : 0;  /* subwindow working on if any*/
    switch(event.type)
      { /* Event eval */
      case KeyPress:
        lth= XLookupString((XKeyEvent *)(&event), keyBuf,keyBufMaxLen, &keySym, &compStat);
        /* Put into INPUT buffer. If '\n', copy to SESSION windows & process  keystrokes. */
        processKey(host,wind,keyBuffer,event.xkey.state);
        if (xconf.nDisplaysActive<=0)
          goto exitProgram;   /* in case typed ^D in Master-icon window */
        break;
```

```
case ButtonPress:
  if (xconf.hList[host]->dpy!=NULL && wind==xconf.hList[host]->masterWindow)
    toggleMasterIcon(host); /* i.e. iconisize this host */

  if (subwind!=0)
    {/* It must be a scrollBar or radio button - check for it */
      int rb, scrDBnbr,
          foundScrollButton= FALSE;
      host_t *ht= xconf.hList[host];

      /* Note: scroll #s go from SESSION_POPUP, HELP_POPUP, to MAXSCROLL_POPUPS */
      for (scrDBnbr=SESSION_POPUP; scrDBnbr<=MAXSCROLL_POPUPS; scrDBnbr++)
        if (wind==ht->scrollPopup[scrDBnbr].wind &&
            subwind==ht->scrollPopup[scrDBnbr].scrollBarWind &&
            (scrDBnbr>=SESSION_POPUP ||
             (scrDBnbr==SESSION_POPUP &&
              (xconf.masterDpy==ht->dpy || host==xconf.sw.moderator))))
          {/* Pressed scroll bar - scroll data in active windows */
            moveByScrollBar(host, scrDBnbr, event.xbutton.x, event.xbutton.y);
            foundScrollButton= TRUE;
            break;
          }/* Pressed scroll bar - scroll data in active windows */

      if (!foundScrollButton)
        for (rb=1; rb<=xconf.maxRadioButtons;rb++)
          if (ht->buttonWindow[rb]==subwind)
            {/* Process radio/command button press */
              doProcessRadioButtons(host,&event,wind,subwind);
              break;
            }/* Process radio/command button press */
    }/* It must be a scrollBar or radio button - check for it */

  else
    {/* Process continuous FLICKER/MOVIE mode key-button commands */

      int buttons= event.xbutton.button,
          state= event.xbutton.state;

      if (xconf.masterDpy!=NULL && xconf.hList[host]->dpy!=xconf.masterDpy)
        break;            /* ignore non-master host */

      if (xconf.sw.movieSwitch)
        showMovieMouseCmds(&event, host, wind, buttons, state)
      else markAndFlickerImagesMouseCmds(&event, host, wind, buttons, state);
    }/* Process continuous FLICKER/MOVIE mode key-button commands */
  break;

case Expose:
  doExpose(host,(XExposeEvent *)&event); /* process Expose event if no backing store */
  break;

case DestroyNotify:
  sprintf(sMsg,"Host [%s] is leaving the conference.\n",
          xconf.hList[host]->hostName);
  updateSessionWindow(sMsg,0);
  deleteHost(host);       /* disable display struct for this host */
  break;

  /* Intercept window manager delete window messages. */
case ClientMessage:
  if (event.xclient.format==32)
    {/* check for messages */
      host_t *ht= xconf.hList[host];
      Atom delWind= XInternAtom(ht->dpy,"WM_DELETE_WINDOW",FALSE);

      if (event.xclient.message_type==delWind)
        {/* OK the server wants us to kill the connection! */
          XCloseDisplay(ht->dpy);
```

```
                deleteHost(host);     /* free storage and clear ht->dpy*/
                if (--xconf.nDisplaysActive==0)
                  goto exitProgram;   /* No hosts left in conf. THE END */
                sprintf(sMsg,"Host %s is leaving the conferance at %s.\n",
                        ht->hostName, todays_date());
                updateSessionWindow(sMsg,0);
              }/* OK the server wants us to kill the connection! */
            }/* check for messages */
          break;

        . . .

        default:
          break;
        } /* Event eval */

    doneThisEvent:
      ;
      . . .
  }/* here until the end of time */


exitProgram:
 /* [3] CLOSEDOWN Stragegy: kill any remaining opened displays then die. */
 for (host=1;host<=xconf.nHosts;host++)
   if (xconf.hList[host]->dpy!=NULL)
     {/* cleanup */
       XCloseDisplay(xconf.hList[host]->dpy);
       deleteHost(host);
     }/* cleanup */

   . . .
}/*DOEVENTLOOP*/
```

5. SUMMARY

Multimedia computer conferencing allows people to share image and text data in near real-time even when located far apart. Xconf is a multimedia computer conferencing system which uses existing hardware technology and does not require any special conferencing software on the user's end. Users take turns controlling a common conferencing database and may use the conference to display text and image data windows on users' displays. In addition, they may run other non-Xconf software that generates "well behaved" windows (in terms of colormap usage) and have these windows exported by Xconf for all other conference users to see.

Xconf is an example of groupware for small conferences which need to discuss a moderate amount of visual data. It was designed for existing networks and workstations to reach the widest set of users. This was done at the cost of response time because of current limitations on network bandwidth and display resolution. Therefore, it is adequate for many problem domains, but not for very high resolution diagnostic medical imagery.

Xconf requires minimal user display resources: TCP/IP or DECnet network-connected X-Window display systems. Color X displays are best, but it does work with black and white. No other special hardware is required and no special conference-server programs are needed on any hosts which join a conference, so setting up a conference is very easy.

The key to maximizing utility in a conference is to transparently share and manipulate information important to the group. Simple and nondistracting control of the conference must be a primary design goal. For this we made a transferable "Master" user, a Master access queue, and a Master control of global commands to help maintain a single theme in a conference. Alternatively, if conferees may cooperate without central control, then a free-for-all conference mode of operation allows more rapid interactive manipulation of the images. Using a telephone conference call in parallel with Xconf helps provide enough control so that a free-for-all image conference is usable.

Another main conferencing goal is simultaneity so everyone knows what everyone else is doing at all times and responses to ideas and suggestions may be given immediately. This immediacy results in higher rates of idea communication because of minimizing "turn-around" response times during brainstorming.

Specifically, everyone should see any text other users are entering or actions involving images they have taken such as pointing to objects in images. This paradigm replicates needed information on all participants displays presenting a WYSIWIS environment ["What You See Is What I See" first coined by Stefik (*31*)]. Information is divided into that which is global (seen by all users) and that which is local (optically called up by individual users). Users are aware of who is actively participating and who is inactive (iconisized). Common windows for user conversational input, session output, and images allow all users to see this global status of the conference. Individual users may invoke local pushbutton commands to create scrollable pop-up information windows such as the subsession windows to view subsets of conference information. This lets them direct their attention to relevant subgroup activity in the conference.

Some image operations used for analyzing our 2D protein gels, were added to Xconf (for example, pointing to objects in images, flickering, and making movies of sets of image data frames). We generalized these direct manipulation concepts in Xconf for use with other problem domains. Although Xconf could have been intimately tied into our GELLAB-II system, the choice was made to generalize it so that conferencing could be used with *any* X-Window application using the subshell and the import–export window interface.

Information may be shared from several sources. Image and text files residing on the Xconf client host system may be read by the conference server. Image or window data from conferees's displays may be added to the conference at any time by exporting selected windows. The conference may also share a Unix shell running on the Xconf client host system. With the latter, it is possible for the Master user to run any application program with the results shared by all users. Because any user may become the Master, different users may take turns running the shared application to try out different data analysis ideas. An important implication is that the conference may run database program analyses that result in images or windows. These visual results may then be exported into the conference for all to see and comment on.

Social Issues of Simultaneous Conferencing

Simultaneous conferencing depends on a cooperating group being able to schedule a common block of time when the conference may be held. Because of the impromptu nature of short conferencing sessions, this social scheduling problem may be a major impediment to using this type of tool. This problem becomes more acute as the number of participants grows. Everyone is involved in other activities which are beyond the control of the conference moderator. The difficulty of conference scheduling may be alleviated to some extent by giving the participants the ability to enter a conference after it has started or to leave it before it is over.

For remotely located users, combining a Xconf session with a telephone conference call may help minimize "insecurity" in initially using computer conferencing. Verbal discussion may preceed each Xconf user image interaction to help the flow of the conference. This is especially useful for conferences involving occasional users, for larger groups of people, or for conferences with complicated image data.

In summary, Xconf is a tool for sharing image and conversational text data in real time among a geographically distant group of people. It does this by providing the medium for transferring information and coordinating user interactions.

Future Directions

As we gain experience in the routine use of this tool, the Xconf user interface will evolve more toward helping people communicate without the "tools" getting in the way. Its direct manipulation interface should be transparent and obvious, requiring minimal concentration to use. There will also be increased integration with various types of databases.

Other issues that would improve conferencing have to do with overcoming problems in the underlying X-Window System protocol X11R4/R5. We need to supply information about movie frame synchronization rates to the remote X display servers or window managers. This is important because of variability in network bandwidth and X display server speeds. Fine-grain control would allow better reproduction of movies at each remote display by specifying image frame-change rates in real time. Currently, Xconf may optionally synchronize each frame. However, this still does not give the type of fine-grain synchronization control desired since it may slow the movie too much if there are many users or slow network connections.

It would also be useful if image zoom and compression were available as part of the X-Window System protocol. Images could be sent compressed, saving time in this major bottleneck rather than computing the zoom window and then transmitting the uncompressed pixmaps. There are several different solutions: (1) changing the X-Window protocol standard to include new synchronization, compression, and zoom primitives—difficult because of inertia but rumored to be available in version X11R6; (2) using a modified window man-

ager—also difficult since it must reside on all user's systems; and (3) using a Xconf demon program "Xconfd" running on remote users' workstations.

This last method will probably be used in the future. We also envision rewriting Xconf using the X-Windows Xt tool kit rather than with Xlib, possibly with a conferencing widget. Jones has used Xt in a multiuser X application (29), so the technology is there. Because it adds complexity, and since a major goal of Xconf is simplicity, it was not used in the initial design. However, it might be added in the future using Sun remote procedure call methodology similar or X Window "Atom" properties to pass encoded synchronization or compressed image data and requests for complex server operations. The use of a remote demon could be added in a robust way so that if the Xconfd exists on a particular workstation, Xconf would use it. If not, Xconf would degrade gracefully and do the same thing but more slowly than if using a demon.

Xconf is undergoing beta-testing with some features such as zoom and draw not fully implemented.

ACKNOWLEDGMENTS

Thanks for constructive feedback on the user interface are due Rob Ashmore, Steve Bailey, Ann Barber, Stewart Durell, Adam Feigin, Jeff Garlough, Peter Greif, Andrew Grimshaw, Voitek Kasprzak, Gerald Latter, Eric Lester, Richard Levenson, Jake Maizel, George McGreggor, Mark Miller, Art Olson, Geof Orr, John Powell, Don Preuss, Pete Rogan, Tom Schneider, Mike Scott, Bruce Shapiro, Randy Smith, John Taylor, Kyle Upton, Jack Waters, and others who helped test the early versions of Xconf under a variety of X Servers, window managers, and systems. Thanks to Steve Bacharach for a series of MRI heart image data which were used in testing the multiple images/file option for movies, and to Stewart Durell for preparing a multiframe 360° degree movie sequence data of a protein molecular model enabling us to "walk" around the molecule. Many of their suggestions and critiques are reflected in the current design and in this paper.

REFERENCES

1. LEEBAERT, D. "Technology 2001, the Future." MIT Press, Cambridge, MA, 1991.
2. LEMKIN, P. F. Xconf—A multimedia conferencing system as a collaborative tool. Talk presented at International Electrophoresis Society Conference, Washington, DC, March 19–21, 1991.
3. COMER, D. E., AND NARTEN, T. TCP/IP. *In* "UNIX Networking" (P. H. Wood and S. G. Kochen, Eds.), pp. 49–91. Hayden Books, Indianapolis, IN, 1989.
4. CERF, V. Networks. *Sci. Am.* **269**(3), 72–81, 1991.
5. SCHEIFLER, R. W., GETTYS, J., AND NEWMAN, R. "The X Window System." Digital Press, Bedford, MA, 1988.
6. LIPKIN, L. E., AND LEMKIN, P. F., Database techniques for multiple PAGE (2D gel) analysis. *Clin. Chem.* **26,** 1403–1413, 1980.
7. LEMKIN, P. F., AND LESTER, E. P. Database and search techniques for 2-D gel protein data: A comparison of paradigms for exploratory data analysis and prospects for biological modeling. *Electrophoresis* **10**(2), 122–140, 1989.
8. LEMKIN, P. F. GELLAB-II, A workstation based 2-D electrophoresis gel analysis system. *In* "Two-Dimensional Electrophoresis," (S. Hanash and T. Endler, Eds.), pp. 53–57. VCH Press, Germany, 1989.
9. LEMKIN, P., MERRIL, C., LIPKIN, L., VAN KEUREN, M., OERTEL, W., SHAPIRO, B., WADE, M., SCHULTZ, M., AND SMITH, E. Software aids for the analysis of 2-D gel electrophoresis images. *Comput. Biomed. Res.* **12,** 517–544, 1979.

10. BARINAGA, M. Biology goes to the movies. *Science* **250,** 1204–1206, 1990.
11. ELLIS, C. A., GIBBS, S. J., AND REIN, G. L. Groupware—Some issues and experiences. *CACM* **34**(1), 38–58, 1991.
12. KLING, R. Reading "all about" computerization: Five common genres of social analysis. *In* "Directions in Advanced Computer Systems" (D. Schuler, Ed.). Ablex, Norwood, NJ, 1990. [Draft available from comp.groupware.]
13. RAPAPORT, M. "Computer Mediated Communications: Bulletin Boards." Wiley, New York, 1991.
14. SCHRAGE, M. Computer tools for thinking in tandem. *Science* **253,** 505–507, 1991.
15. MCGARTY, T. P., AND SUNUNU, S. B. Applications of multimedia communications systems to health care transaction management. *In* "HIMS Conference Proceedings, February 1991."
16. PIZER, S. M., AND BEARD, D. V. Medical image work stations: Functions and implementation. *J. Digital Imaging* **2**(4), 185–193, 1989.
17. DREW, P. G. Pac systems: Medical image management systems are the technology of the future and will be for quite a while. *Pixel* Nov/Dec, 22–23, 1990.
18. TESLER, L. G. Networked computing in the 1990s. *Sci. Am.* **269**(3), 86–93, 1991.
19. SPROULL, L., AND KIESLER, S. Computers, networks and work. *Sci. Am.* **269**(3), 116–123, 1991.
20. HILTZ, S. R., AND TUROFF, M. Structured compute mediated communication systems to avoid information overload. *CACM* **28**(7), 680–689, 1985.
21. HILTZ, S. R., JOHNSON, K., AND TUROFF, M. Experiments in group decision making. *Human Commun. Res.* **13**(2), 225–252, 1986.
22. HILTZ, S. R. The "virtual classroom": Using computer-mediated communication for university teaching. *J. Commun.* **36**(2), 96–104, 1986.
23. MCGARTY, T. P. Image processing in full multimedia communications. *Advanced Imaging,* Nov., 28–33, 1990.
24. MCGARTY, T. P. Multimedia communications in diagnostic imaging. *Invest. Radiol.* 1991.
25. ROGER, E., GOLDBERG, M., AND DILLON, R. F. Image organization and navigational strategies for a radiological work station. *J. Digital Imaging* **2**(4), 229–244, 1989.
26. LITTLE, D. C., AND GHAFOOR, A. Synchronization and storage models for multimedia objects. *IEEE J. Selected Areas Commun.* **8**(13), 413–427, 1990.
27. GRUDIN, J. Why CSCW applications fail: problems in the design and evaluation of organized interfaces. *In* "Proceedings, 2nd Conference on Computer Supported Cooperative Work," pp. 85–93. ACM, New York, 1988.
28. JONES, O. "Introduction to the X-Window System." Prentice–Hall, Englewood Cliffs, NJ, 1989. Pages 404–408 discuss multiple display connections.
29. JONES, O. Multi-user application software using Xt. *The X Resource* **3,** 55–76, 1992.
30. BBN. "BBN/Slate—Multimedia communication software for work-groups." BBN, Cambridge, MA, 1991.
31. STEIFIK, M., BOBROW, D. G., FOSTER, G., LANNING, S., AND TARTAR, D. WYSI-WIS revised: Early experiences with multiuser interfaces. *ACM Trans. Off. Inf. Sys.* **5**(2), 147–186, 1987.
32. LESTER, E. P., LEMKIN, P. F., LOWERY, J. F., AND LIPKIN, L. E. Human leukemias: A preliminary 2d electrophoretic analysis. *Electrophoresis* **3,** 364–375, 1982.