LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-282

COMPUTER-BASED REAL-TIME CONFERENCES

Sunil Sarin

Irene Greif

July 1985

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Computer-Based Real-Time Conferences

Sunil Sarin, Computer Corporation of America
Irene Greif, M.I.T. Laboratory for Computer Science

To appear in *IEEE Computer 18*, 10 (October 1985), special issue on Computer-Based Multimedia Communication.

ABSTRACT:

A *real-time conferencing* system allows a group of users to conduct a problem-solving meeting from their workstations. Participants in such a conference use the computer to jointly view, edit, and process relevant information, and use voice communication to discuss the information they are sharing. General principles are presented in this paper for selecting a set of user functions in a real-time conferencing system. The available implementation strategies are reviewed and compared, with emphasis on the tradeoffs between reusing existing single-user interactive programs and writing new distributed multi-user programs. Network communication requirements for real-time conferences, and their potential impact on communication protocol standards, are discussed. Real-time conferencing is contrasted with asynchronous communication support such as electronic message systems and shared databases, and the need for the two to work together within the total system environment is emphasized.

Key Words: Conferencing, Network Protocols

1. Introduction

Computer support for group work, including electronic mail[1], computer conferencing[2], form management[3], and coordination support[4, 5], primarily addresses *asynchronous* interaction among users. These systems are most useful when each user can work at times of his own choosing. However, for certain group tasks such as crisis handling, simultaneous or *real-time* interaction is essential. To date, relatively little work has been done on computer support for people working together simultaneously. In a *real-time conference*, each participant can be seated in his own office at a workstation that might include a high-resolution screen for computer output, a keyboard and a pointing device, a microphone and a speaker, and possibly a camera and video monitor. Parts of each participant's screen will be dedicated to displaying a *shared space* in which everyone sees the same information. The voice communication equipment can be used by the conference participants for discussion and negotiation; video communication can add an illusion of physical presence, simulating a face-to-face meeting; and conversational references ("this number" or "that sentence") can be clarified by pointing at displayed information. The displayed information can be dynamically edited and processed, permanent records can be saved, and new information that is relevant to the discussion can be retrieved for display at any time. Participants may, in addition, have *private* spaces on their screens, allowing them to view relevant private data or to compose and review information before submitting it to the shared space.

Systems that provide some of the above features already exist. As early as 1968, the NLS system[6] provided a *shared-screen* mode for simultaneous collaborative authoring of structured documents. This facility, which can be used to access any interactive program from multiple terminals, is now available in many time-shared operating systems in the form of *terminal linking*. Terminal linking on most systems does not work correctly unless all linked terminals are of the same type. A notable exception to this is Tymshare's Augment[7] system (the commercial successor to NLS), which supports "virtual" terminal linking across dissimilar terminal types. Augment also includes commands for passing the "floor", i.e., permission to enter program input, from one linked terminal to the next.

Real-time conferencing can be used to support joint work in many different applications. For example, a fairly common use of terminal linking is joint debugging of programs from remote terminals, often accompanied by a telephone conversation. The Balsa algorithm animation system at Brown University[8] supports programming tutorials and demonstrations. An instructor can interactively invoke and control a program whose execution and results are displayed graphically on the workstations of the students. A student viewing the demonstration on his workstation can pause at selected points, even as the instructor's program continues to run. Bell Laboratories' TOPES system, which supports the design and engineering analysis of complex building plans, has a

graphics teleconferencing feature[9] that allows a group of engineers to hold a design meeting on-line while conducting a voice conversation over the phone. The *chairperson's* screen shows an image that can also be displayed on other participants' screens, and participants may take turns entering commands to be processed by the chairperson's design editing and analysis program.

Real-time conferencing is also beginning to appear on the personal computer (PC) market. At the American Society of Civil Engineers Annual Conference in October 1982, Structural Programming Inc. demonstrated a teleconferencing enhancement to their Palette computer-aided drafting system. This feature allows two users to jointly edit the same design drawing, using independent cursors that are visible to both users, from personal computers connected by a telephone line. At the Federal Computer Conference in September 1984, Teneron Corp. demonstrated the Tango-writer Interactive Word Processor which allows a telephone link between two PCs to be alternately switched between voice communication and data communication, the latter allowing either user to edit a document while the other user watches.

Each of the above systems was built on an *ad hoc* basis to meet a particular need. Our research into real-time conferencing[10] identified general principles underlying the design and implementation of real-time conferences. Based on our own prototype implementations, described in Section 3, we identified several potentially useful functions for real-time conferences. These functions and related user interface issues are discussed in Section 4. Section 5 describes and contrasts the different approaches to organizing the conferencing and application software, while Section 6 discusses the protocols needed to implement real-time conferences on interconnected workstations. Before we describe our prototype systems and our design and implementation guidelines, Section 2 briefly reviews and contrasts other kinds of communication and informational support that could be used for real-time group meetings.

2. Other Conferencing Media

The emphasis in the kind of real-time conference we describe is somewhat different from *teleconferencing* that uses video communication to provide some simulation of face-to-face contact among participants[11]. Video teleconferencing is more valuable when nonverbal communication, in the form of gestures and facial expressions, is an important part of the discussion and negotiation that takes place. On the other hand, among people who have already established a working relationship and use shared on-line information as an integral component of their work, a real-time conference will be an ideal way of holding a problem-solving meeting. Video communication may be a useful enhancement to the kinds of real-time conference we are describing, but will be less critical than voice communication and the ability to share information and software.

Computer-based information sharing tools have an advantage over other informational aids typically included in teleconferencing systems, such as facsimile transmission and *telewriting*[12] (which transmits hand-drawn sketches in highly compressed form). These facilities do not allow information that was originated from one location to be modified by participants at other locations. *Electronic blackboards* do allow joint manipulation of a shared image, but are subject to the same limitations as a physical blackboard. That is, the only manipulation possible on a real or simulated blackboard is writing, drawing, and erasing; it is not possible to easily move and rearrange text or the components of a drawing, or to change an entry in a spreadsheet and have the totals automatically recomputed, or to run a complex simulation model in order to evaluate a proposed plan. Furthermore, the only information that can be shared in a face-to-face meeting or teleconference is that which was brought to the meeting or composed during the meeting. Participants cannot access the increasing multitude of private, corporate, and public databases to satisfy unanticipated information needs. The ability to dynamically access and manipulate information using powerful computer-based tools is the distinguishing characteristic of the real-time conferences we discuss in this paper.

The ability to access unanticipated information, and to process and analyze information, also makes the computer useful for meetings held within a single room. Brown University's Balsa system, described above, can be used in a special lecture hall where each student has a workstation. At the Xerox Palo Alto Research Center, Colab[13] is a specially-equipped room that has a large display on the wall, and a smaller recessed screen (with keyboard and mouse) in front of each participant.* The wall display shows the shared space of the meeting, while the individual screens can be used for viewing and manipulating private spaces. The participants can discuss and negotiate, verbally and non-verbally, as in any face-to-face meeting, but also have access to new meeting support tools such as "Argnoter" which allows participants to record the structure of their discussion in the form of a network of nodes (text statements or graphic objects) connected by various kinds of semantic links. Computer-based tools are thus likely to be useful in both conventional meetings and in real-time conferences held at a distance.

3. Two Prototype Systems

This section describes two prototype real-time conferencing systems, developed at the M.I.T. Laboratory for Computer Science, that illustrate two ends of the spectrum of possible implementation strategies. The first system supports a specific activity in a conference, and is tailored to that activity

*Xerox PARC also saw the development of a few experimental systems supporting joint document editing and electronic circuit editing over the Ethernet; these systems are, unfortunately, undocumented.

alone, while the second combines a simple conference support package with preexisting application code to provide a more limited meeting facility. Voice communication was not included as an integral part of these systems, but must be set up by outside means in order to conduct a multimedia conference. We used the telephone system's "conference call" feature when more than two people were in a conference. To keep the participants' hands free for interacting with the system, and to avoid cramped shoulders, we equipped our telephones with speaker-phones.

3.1. Real-Time Meeting Scheduling

The first prototype, RTCAL, supports meeting scheduling by building a shared workspace based on participants' on-line calendars. RTCAL provides users with information and tools for decision support; it does not attempt to automate the selection of a meeting time. We chose meeting scheduling as an application area partly because many users on our system maintained on-line calendars using the personal calendar system PCAL[14]. This gave RTCAL a realistic database from which to read relevant calendar information, and in which to record decisions made during a conference.

The salient features of RTCAL, which are illustrated in Figures 1 through 3, are:

- *Shared and private spaces*. The former present information (the conference status, and the group calendar information) that is visible to all participants, while the latter present information (details of private appointments) visible only to the individual participants.

- *Alignment of related information*. Whenever the shared calendar window is scrolled, the private window is also scrolled so that the two always show the same date and time range side by side. The participant can thus always see relevant private details about his own appointments for the date and time range that is visible in the shared space.

- *Voting*. When a meeting time is proposed (by the current controller), each participant is asked to vote "yes" or "no". Unlike a voice-only conference, where each participant must be polled in turn, voting on-line is more efficient because participants can vote in parallel (and are not influenced by each others' votes). Votes are collected and tabulated on the shared display (as shown in Figure 2) until all participants have voted. If some participants take a very long time to vote, the chairperson can timeout the voting procedure and allow the conference to continue. The vote status of proposals (alternative proposals are allowed) can be examined later, in which case a participant's vote that arrived late will be reflected in the display.

- *Participant autonomy*. A participant is free to leave a conference at any time, and to return when he pleases; when he does so, his display of the shared space is automatically brought up-to-date.

A participant can choose to ignore the decision of the group (i.e., the committed meeting time) and not have the committed time entered into his private calendar. Further autonomy in manipulating the private space could have been provided (but was not implemented), by giving participants the option to scroll the private window to any date and time range independent of the shared window, and to return to automatic alignment of the windows at any time.

- *Separation of application and conference control commands.* Calendar-specific commands (scrolling the shared window, proposing or committing a meeting time) are echoed in the shared space as typed, and only one participant at a time (the *controller*) can enter such commands. A separate set of control commands, invoked by typing an "escape" character, is used for conference-specific actions such as requesting and passing control, leaving and terminating the conference, and other functions listed in Figure 3. This distinction between two types of commands was easy for users to grasp, and we believe will be important for other conferencing applications as well.

- *Conference roles.* The *chairperson*, who creates the conference and invites participants, oversees all activity in the conference. He determines who has control (permission to enter calendar commands) at any given time, and is the only participant who can terminate the conference. The other participants have no power except when they are given control of the shared space by the chairperson (who may take it away).

- *Presentation of status information.* A small summary window indicates what the conference is about, which participants are present (and which were invited and are unavailable or have not yet responded), who the chairperson is, and who currently has control. Important changes to status information (e.g., a participant leaving or joining, or passing of control) not only update the summary window but also cause a one-line notification to be displayed in reverse-video. This was found to be important because there are many kinds of information simultaneously displayed on the screen, and a participant will often not notice a small change to summary information especially if he is looking elsewhere, e.g., at the calendar information.

RTCAL was implemented on a time-shared host, but each participant was represented by a separate process and the processes communicated by message-passing only. This allowed us to treat RTCAL as a logically distributed system, and to explore implementation techniques that would carry over to to a real distributed environment.

RTCAL supports a specific application activity in a real-time conference. The next prototype system is an example of a generic facility, albeit limited, that can be used by multiple applications.

3.2. Shared Bitmap System

Our shared bitmap system, MBlink, supports raster graphics output on workstations over a network, and graphical input from each workstation's pointing device (a mouse). MBlink extends an existing experimental protocol "Blink"* that provides remote bitmapped graphics between a single workstation and a host. MBlink allows a given bitmap to be displayed identically on multiple workstations rather than just one. In addition, MBlink can track every workstation's mouse and show its position on the shared bitmap, thus allowing participants in a real-time conference to point at information under discussion. MBlink is a generic utility that can be used by Blink applications with little or no modification to existing code.

MBlink, like Blink, is implemented with the application program running on a DEC Vax™ mainframe and with the participants viewing a shared bitmap from Xerox Alto workstations; the machines are connected by local area networks and gateways using the DoD Internet Protocol. The shared bitmap is implemented on the mainframe as an abstract data type module that is linked into an application program that wishes to use it. The application program manipulates the bitmap as if it were the bitmap of a memory-mapped screen, i.e., it can read and write arbitrary pixels or regions, perform "RasterOps" (bitwise boolean operations) on rectangles, and so on. However, invisible to the application, the shared bitmap module transmits bitmap changes to the remote workstations on which the bitmap is actually displayed.

Each workstation in turn reports the position of its mouse, and the state (up or down) of the buttons on the mouse, to the bitmap module on the mainframe. Each workstation is assigned a 16x16-pixel pointer "shape" (bit pattern) that is superimposed on the bitmap at the reported mouse position. The bitmap module assigns default pointer shapes, shown in Figure 4, but the application can specify other shapes if it desires. Each participant can see on his workstation screen the position of every participant's mouse. In addition, the hardware at each workstation tracks its own mouse and displays a distinguished pattern on the screen; each participant can therefore easily see the location of his own mouse. A participant actually sees two cursors showing his own mouse position, one locally tracked by the workstation hardware and the other remotely tracked or "echoed" by the mainframe. When the mouse is moving, the remote echo lags behind the locally tracked position by about half a second; when the mouse stops moving, the participant can judge approximately when the other workstations have caught up and can see his pointer at the correct position.

The application can read the position and button states of any workstation's mouse at any time, and can use this to implement any commands it wishes. Keyboard input is not handled by the shared

---

*Designed by David Reed at M.I.T.

bitmap module; this was a limitation of Blink that would have required considerable software modification to rectify. Rather, the application reads keyboard input from the terminal that was used to start the application program. Terminal input is invisible to the bitmap module, and the terminal from which the application reads keyboard input cannot be changed during a conference.

The shared bitmap module provides procedure calls for adding and removing workstations on which the bitmap is displayed. In addition, the application program can ask the module to synchronize with the workstations, only returning control when all workstations are known to have received all bitmap updates or when a specified timeout period expires. This can be useful if, for example, the application wishes to display an important prompt before reading input. The application can at any time check the status of any workstation, in order to determine whether it is displaying the bitmap at all (a workstation that was added may never have responded) and if so whether the workstation's bitmap display is up-to-date.

To demonstrate MBlink, we took an existing single-user application* and linked it with the MBlink version of the bitmap module in place of the Blink version. This application uses the bitmap to present a graphical view of an internetwork being simulated together with dynamically updated information describing the state (e.g., packet queue lengths) of the hosts, networks, and gateways. About a dozen lines of code were added to the application's command input loop to allow workstations to be added to and removed from the conference; this was the only change to the existing application code.

4. Conference Design Issues

We assume that the participants in a real-time conference are interested in manipulating some collection of abstract *objects* of various types (e.g., electronic circuit designs, or documents, or spreadsheets) that are relevant to the problem they wish to solve. Some kind of *editor* may be provided for displaying and modifying the state of an object, and various application tools (e.g., a circuit simulator or spelling corrector) may be invoked to analyze and process the objects in some useful way. This abstract model readily extends to multimedia conferencing systems, in that a shared voice channel, or video image, can be treated as an abstract object that supports an interface (e.g., speaking into a microphone) peculiar to its object type.

---

*The application was written by Lixia Zhang of M.I.T., for experimenting with different internetwork congestion control algorithms.

4.1. User Interface

Existing techniques for designing pleasant user interfaces are certainly applicable to real-time conferences. These include windows, pop-up menus, and the use of pointing devices for selecting objects and commands or for sketching.

Somewhat independent of the exact user interface style is the issue of what users need to learn in order to conduct a real-time conference. Many contemporary interactive systems emphasize the principle of *uniformity* of interface across multiple applications: similar commands do similar things, so that the number of commands a user must learn for a new application is minimized. For real-time conferencing, this principle can be extended in two important ways:

1. Real-time conferences developed for different applications should provide consistent commands for conference control, e.g., for conference initiation, floor-passing, and joining and leaving the conference.
2. When participants wish to access an existing application tool from a real-time conference, they should be presented with an interface that is consistent with the interface that an individual user sees when he interacts with the program by himself (i.e., when not in a conference).

Achieving the above objectives will depend largely on having reusable software, both for applications and for conference control. Modern trends in software engineering, such as "object-oriented" programming (which allows new information types to be defined as incremental extensions of existing types) and user interface "toolkits" (which allow construction of uniform interfaces to different applications) are likely to be useful in developing such software.

4.2. Shared versus Individual Views

If all participants in a real-time conference have identical views of shared information, they can make conversational references to the data with the assurance of being understood, just as if they were viewing a common blackboard in a face-to-face meeting. There are occasions, however, when participants with different interests and viewpoints (or with different display capabilities on their workstations) may wish to see the same data presented in different ways. Conversational references can be more difficult to interpret in this case, but need not be if there is sufficient similarity between different participants' views (e.g., the same text formatted differently) and if the references are made in logical terms (e.g., "the second sentence of the first paragraph") rather than in terms of the displayed view. The interaction between the disparity among views and the kinds of references that can be understood is the subject of current linguistics and human factors research, e.g., at Xerox PARC.*

---

*This was reported by John Seely Brown at the CHI'85 Conference on Human Factors in Computing Systems.

When participants do wish to see identical views of the shared space, it is still important to allow some variation in the individual views so that each participant knows which displayed objects have special meaning to him and not to the others. In MBlink, for example, each participant can not only distinguish the different participants' cursors but can also tell which one is his own. Similarly, messages about important changes of status should be specially tailored for the participant that they pertain to. When participant Smith is given the floor in RTCAL, the other participants see the message "Smith has received control", but Smith himself sees "YOU have received control".

In some situations, conference participants may want to view different parts of a large shared space, e.g., different documents or drawings, or different sections of a large document. While conversational references are ineffective in these situations, it is useful for participants to know who is looking at what part of the shared space. Participants can then choose whether to align their views together or to allow them to diverge; they can also tell when others are looking at the same data and will understand conversational references to the data. Many contemporary systems provide some means for an individual user to orient himself, e.g., a horizontal or vertical *scroll bar*, representing the length of the document, in which the document region currently displayed is highlighted, or a graphical *world view* with a superimposed rectangle indicating which region is visible in the main display window. These tools can be extended to highlight multiple regions rather than just one, so that participants can determine who is looking at what.

4.3. Access and Concurrency Control

In a system (e.g., file system or database) where many users operate on shared data over a period of time, it is common to set access controls specifying who can read and update which objects or files. This allows a given user's access to a given object to be tailored to match his "role" with respect to that object. A real-time conference, on the other hand, usually involves a very small group of users who are often peers and whose roles in the conference (e.g., moderator, or note-taker) may change rapidly. We therefore believe that complex access controls are not necessary within a real-time conference. It will usually be best to give all participants equal power to read and update shared data. At most, it may be useful to designate one participant as chairperson, as in RTCAL. This will often be based solely on convenience rather than on one participant actually having some authority over the others. There may also be situations in which update access is restricted to one or a few participants, with the other participants acting only as observers. Except possibly in non-discretionary security environments (such as the military), we do not see the need for access controls more complicated than these.

If more than one participant has update access to the shared space, problems may arise if

participants concurrently enter update commands. This has been observed with terminal linking, where characters typed concurrently by the participants will be interleaved arbitrarily, probably in order of arrival at some process in the system. Even worse, if a participant hits the "rubout" key to erase a character that he typed by mistake, or to correct for the interleaving, he may erase a different character than the one he last typed because somebody else inserted another character (or erased a character) in the meantime.

The problem with interleaving of commands is that the effect of a given command* entered by a participant may be different from what he expected at the time he entered the command, because some other participant's concurrent command may have been processed in between. This can be avoided by using a floor-passing strategy, such as RTCAL's, which only allows one participant at a time to update the shared space. Floor-passing can be generalized by having participants set *reservations* on different parts of the shared space. This will allow participants to work concurrently on different documents or on different sections of a document; in a multimedia conference, one participant may be speaking while another is working on shared data. Two key issues that must be addressed are: the granularity at which reservations are set on the shared space; and the policy (e.g., automatic or manual) for passing a reservation from one participant to another.

Participants may find it a burden to have to explicitly request and release reservations, especially if reservations must be set on many small subparts of the shared space. Participants might instead wish to dispense with reservations altogether, and informally negotiate (e.g., using voice communication) who should work in the shared space and when. Such negotiation will not always work perfectly, and we can expect that interference among concurrent conflicting commands will occasionally happen. But, if all participants are looking at the same displayed information, they are likely to notice when interference does happen and take action to recover from it. The extent to which this approach is workable is yet to be determined; it may be quite reasonable with small groups of two or three participants. With a large number of participants, the voice channel itself becomes an object of contention, making it less likely that voice negotiation will successfully prevent concurrent interference.

4.4. Before and After the Conference

Although we do not exclude the possibility of two (or more) participants happening to be on-line at the same time and wanting to hold a real-time conference, most real-time conferences will take place only after some planning and scheduling. As with any other kind of meeting, participants will

---

*Note that each character typed is a "command" to a text editor or a command line editor.

generally negotiate outside the system (or even use an on-line calendar system) and agree to hold a conference at a specific time. This will give them the opportunity to prepare for the conference. Participants may also distribute relevant on-line materials in advance, to avoid long bulk transmission delays at the time of the conference.

It is convenient to think of a real-time conference as an abstract object, which contains other objects that the conference participants will work on. Existing access control techniques can then be applied to specify which users will be allowed to participate in a conference. This will typically be a short "access control list" of user names, but may in certain cases include the names of user "groups", or even allow any user ("the public") to join. A possibly useful extension, not available with most object access control schemes, is to allow users to find out about a conference but to be given permission only to submit requests to join. Such requests will be approved or denied by a participant who is authorized to do so. A conference should permit new participants to join at any time (and as many times as) they wish, although in certain circumstances (e.g., the number of participants is the maximum that can be supported with acceptable performance) a conference may be permanently or temporarily closed to new participants.

A participant can leave a conference at any time. A conference may be terminated by a command from an authorized participant, such as the chairperson in RTCAL. In real life, meetings do not end so abruptly and it is desirable to permit a smoother phasing out. Thus, a few participants may wish to linger and hold smaller follow-up discussions, or some participant may wish to review and possibly edit the "minutes" of the conference before committing them to permanent storage. In general, unless revocation of access to the conference is called for, a conference should not terminate until all participants leave of their own accord.

4.5. Getting Data In and Out

A real-time conference does not exist in isolation. It will typically access information that has an independent existence prior to the conference, and will generate information that will continue to exist after the conference. The data being read in or written out may have significance to users other than the actual participants of the conference. Thus, some other user may have prepared a draft design or position statement but be unable to attend, or a follow-up task assignment may need to be communicated to a user who was not present at the conference. Such transfer of information into and out of the conference raises some new access control questions that do not arise when an individual user is reading and updating shared information:

- When an object, e.g., a file, is read into the shared space of the conference, whose access rights are checked before allowing the file to be read? Is it sufficient for the participant invoking the read operation to have read access, even if some of the others do

not?

- When data from the shared space is written to a file, is it sufficient for the invoking participant to have write access to the file or must every participant have write access? What if the file write operation was not explicitly invoked by any participant but was automatically generated by the system, e.g., "auto-saving" of a file version; whose access rights must be checked in order for such write operations to be approved by the file system?
- When a new file, or new version of a file, is successfully written from the conference, who gets permission to access the file in the future? Who "owns" the file, i.e., who has the ability to grant and possibly revoke access permissions in the future? Should the file somehow be "jointly" owned by the participants who contributed to its content, rather than individually owned by the participant (if there was one) who happened to invoke the file write command?

The answers to these questions will vary from one user community to the next, and will also depend on the ability of the underlying operating system to perform sophisticated access checking for groups of users.

4.6. Constraints on Real-Time Conference Design

The extent to which the desired functionality can be achieved in a real-time conference depends on the available hardware and software (the workstations, communication network, and operating systems). Thus, it will not be feasible to display high-resolution bitmapped images if participants only have alphanumeric terminals. Even when participants can display bitmapped images, transmission may take too long if the available communication channel is an ordinary telephone line (as opposed to a local area network or a satellite). Communication and processing delays may influence the choice of concurrency control policy, e.g., floor-passing might be more critical when communicating over a long-distance network, with longer delays, than over a local area network. Existing operating system facilities may also constrain the available access control policies.

We expect, at least for the immediate future, that a given real-time conferencing system will be tailored to a particular communication environment and class of workstations. If a flexible system is desired that can adapt itself to different kinds of equipment, the designer will have to consider the possibility of participants in the same conference having workstations with widely differing capabilities (e.g., some participants may have alphanumeric screens while others have bitmapped displays) or connected by communication lines of different delay and bandwidth. In such a situation, the following choices are available:

- All participants' interfaces could be degraded to the "least common denominator" supported by all workstations.
- A minimum requirement could be established and only workstations that meet the requirement may be allowed to participate.

- Individual workstations could provide the best interface they can, at the expense of no longer having a common shared image on all participants' displays.

How these decisions should be made (by a chairperson, or by voting, or automatically by the system), and how they affect the quality of a conference, remain to be seen.

The size of a conference, i.e., the number of participants, influences the nature of the interaction among participants and also constrains the achievable functionality because of the communication and processing load that is imposed. Two-person conferences are quite different from "small" conferences (say three to six participants) which are in turn quite different from "large" conferences of about ten participants. A given real-time conferencing system will probably have to be tailored to a given range of conference sizes; within this range, it might provide special support based on the actual conference size, e.g., changing its "mode" when a third participant is added to a two-person conference.

5. Software Organization

We next turn to the practical problem of realizing some selection of the above conferencing functions in a real system. We assume a *logically distributed* system consisting of a collection of interconnected *nodes*. Nodes do not share memory; they communicate by message-passing only. A node will typically be a host on a network, but could be a process on a time-shared host. Each participant in a real-time conference is represented by a node called a *workstation*. Other nodes might be involved in a conference, to provide services such as user name lookup or permanent storage.

5.1. Sharing Existing Programs

If participants already have a particular application program (say a document editor or circuit simulator) that they use as part of their work, they will be most comfortable using the same program during real-time conferences. Then the only learning overhead is that of mastering a few simple commands for conference control. Given that existing application programs typically interact with a single user's terminal via an input and output character stream, such a program can be shared in a real-time conference, without modification, as follows:

- Instead of interacting directly with a user's terminal, the program interacts via a *virtual terminal* channel with a *controller* node (or process) that reads the output of the program and sends input to the program.
- The controller node in turn communicates with the participants' workstations and multiplexes the program's input and output in such a way as to realize the functionality desired in a conference. That is, output from the program can be sent by the controller to every workstation, and the controller can selectively feed input from the workstations to

the program based on which workstation has the floor.

This *virtual terminal* approach, illustrated in Figure 5(a), has several advantages: constructing the controller program is not difficult or time-consuming, and once the controller code is available, arbitrary programs can be connected up in a real-time conference. It is also possible to share more than one program, using multiple windows, and individual participants may run private programs in separate windows on their workstation screens. Other convenience features, provided by some time-shared executive programs to individual users, may also be provided, such as copying of output from one window to be fed as input into any window, and recording of input and output "scripts" which can be reviewed or replayed.

Using the virtual terminal approach, it is necessary to choose a particular virtual terminal protocol that the controller will support and that the application program and workstations must adhere to. Many kinds of virtual terminal protocol are available, such as:

- Alphanumeric screen protocols that allow positioning of the cursor at arbitrary locations and selective erasing of parts of the screen. Augment's virtual terminal protocol[7] is of this kind. Recent extensions to screen protocols include highlighting options (reverse-video, underlined, bold, and blinking) and multiple colors.
- Graphics terminal protocols that allow drawing lines and arcs, filling regions with specified patterns, and various operations (and, exclusive-or, etc.) on rectangular subsets of the screen bitmap. The protocol may support pointing device as well as keyboard input. Reed's Blink protocol, which we used in MBlink, is a limited kind of graphics terminal protocol that supports bitmap operations and pointing device input. Stanford's VGTS[15] is a more advanced structured graphics protocol.

Because application programs that do not follow the chosen protocol will not be accessible from a real-time conference, the protocol used by the conference controller should be chosen based on the terminal functions used by existing programs. Workstations that can interact directly with an application program using the given protocol will then be able to access the same program via the conference controller as well.

While sharing existing programs has the virtue of simplicity and low cost (of implementation, and of user training), this is offset by limitations in functionality that arise because the application program believes it is interacting with a single fixed user:

- Multiple concurrent contexts (e.g., editing cursors) in the same shared space cannot be supported.
- Private views of shared information cannot be supported. Private windows into separate application programs can be supported, but that is not the same thing.
- Transfer of application information between shared and private windows can be done only at a very low level. The shared space consists of arrays of characters or bits, which a receiving program may not be able to parse.

- Participants cannot be given different access privileges or naming environments. All operations executed by the shared program will be based on the privileges and naming environment of one fixed user (the chairperson, say), which will in general be different from those of the other participants. Thus, a participant may attempt to read or write a file that he normally has access to, only to fail because the chairperson does not have the required access. Or, a participant may inadvertently write a file into the chairperson's directory rather than his own.

A variant of the virtual terminal approach, which has slightly different characteristics but shares many of the same limitations, is the sharing of terminal input rather than output. Each participant's workstation runs an identical instance of the program, and identical input (from whichever participant has the floor) is sent to all workstations for processing by the program. Sending program input, rather than the output generated by a single program instance, can save considerable transmission bandwidth. This approach, which is used in Brown University's Balsa system, is useful when the workstations can run the program in parallel; it is wasteful and slow if the workstations share a processor on a host machine. Some care is needed to ensure that the program instances on the workstations actually do the same thing when presented with identical input. This will entail some modifications to the application code to trap commands that interact with the program's environment, e.g., to write a file. Such commands should usually be executed by one workstation (the one that issued the command) rather than all in order to avoid anomalous results.

5.2. Multi-User Applications

The alternative to using existing single-user programs is to write new application programs that can interact with multiple users simultaneously. What distinguishes such a multi-user application from a single-user one is that it explicitly takes into account the identity of different participants in a conference. Multiple editing cursors into shared data can be supported, and shared application data (e.g., a structured document with sections and paragraphs) may have associated ownership and access information that determines who can perform what operations. This access information can be derived from the access controls on external data objects that were brought into the conference.

With this approach, a participant's workstation (which may be a process on a host machine, if the participant has an ordinary terminal) is supplied with high-level application information rather than a screen image so that it can provide the participant with whatever view of the shared space he desires; this is illustrated in Figure 5(b). This approach requires defining a "high-level protocol"[16] for application-level communication between the workstations in a conference. Different applications will require different high-level protocols, so this approach has some cost (relative to the virtual terminal approach) in that each new conferencing application may require significant programming effort to construct. This was one of the limitations of the RTCAL implementation; while it did the job

well for its particular application, the code could not be easily adapted to a different application. As more experience with different applications is gained, it should be possible to factor out software for common functions (such as conference initiation, floor-passing, and voting) into a conferencing "toolkit" that can be used by multiple applications. The cost of implementing the toolkit will be incurred only once; each new conferencing application will only have to implement application-specific data types and operations and interface them with the toolkit functions.

When workstations can store and process application information, they may keep copies of more data than just what is currently displayed, e.g., data that was transmitted for display earlier in the conference. This avoids transmission delays and provides faster response whenever a participant scrolls or browses over parts of the shared space that are already available in his workstation's local store. Data that is not immediately needed for display can sometimes be transmitted in advance, e.g., in small pieces whenever the conference participants are momentarily idle and spare communication capacity is available. This can be taken to the extreme by having participants' workstations acquire copies of needed data in advance of the conference. For large documents or high-resolution images, this will avoid long bulk transmission delays at the time of the conference.

A simple approach to implementing a multi-user application, with or without a toolkit, is to use *centralized control*. Some node (which may be a participant's workstation, or a server selected for the conference) acts as the *controller* and processes all commands from participants. The results of commands (or the commands themselves) are sent by the controller to all workstations. This is a similar architecture to that used when sharing existing programs using virtual terminals, except that a wider range of functions can be supported because communication between the controller and workstations is no longer at the level of terminal input and output streams.

5.3. Improving Response Time

A multi-user application with a centralized architecture still suffers from the same performance limitation as using virtual terminals: A command issued by a participant must undergo two message transmission delays (from the originating workstation to the controller, and then from the controller to the workstations) before its effects can be displayed on participant's screens. This delay can sometimes be reduced to a single message transmission time by permitting direct communication of application commands between workstations. Some care must be taken to ensure that if commands issued concurrently by two or more workstations arrive at different workstations in different orders, the workstations' copies of shared data do not become inconsistent. We have identified two main approaches to ensuring this:

1. At any given time, one workstation is the sole source of update commands to a given object or

collection of objects in the shared space. This workstation, which we call the *controller* for the given object(s), can transmit commands directly to the other workstations, and the workstations (including the controller) can process commands immediately with no danger of inconsistency. Control of a given part of the shared space can be passed dynamically from one workstation to another by a resynchronization protocol which ensures that all workstations have processed all commands from the first workstation before they begin receiving commands from the second workstation. A few special objects, such as a participant's pointer position, can only be updated by one fixed workstation; no passing of control and resynchronization is needed for these.

2. Any workstation can issue an update command to any object at any time by transmitting the command directly to all workstations. A workstation receiving a command (including one issued by itself) executes it immediately in a *reversible* way that allows its effects to be undone. Some synchronization mechanism, either a central sequencer or timestamps, is used to define the correct order in which commands from different workstations should be processed. A workstation that received and processed conflicting commands in other than the defined order will undo their effects and reexecute the commands in the correct order. Thus, while workstations may temporarily hold (and display) inconsistent copies of data, such inconsistencies are quickly rectified.

While passing control of an object among workstations is very similar to passing reservations among participants, the idea can be applied independently at the system level and the user interface level. That is, either of the above methods can be used internal to the system, whether or not participants use reservations for concurrency control.

The first approach above is conceptually simpler to implement, especially if one workstation at a time controls the entire shared space. However, each change of control involves the exchange of a large number of messages and a delay of at least two message transmission times. If the activity in the conference is such that the same participant performs many commands in a row, then this overhead will have minimal impact because changes of control will be infrequent and once a given workstation gains control each subsequent command will be processed after one message transmission delay. However, if control must be passed frequently, the overhead wil be significant and will probably cause response time to degrade.

The second approach above never requires any resynchronization, and every command is processed after undergoing one message transmission delay. While undoing and reexecuting commands may seem complicated, many modern interactive programs already have this capability. This technique does affect the user interface in that a participant may see inconsistent states that no

other participant sees and that could not have occurred if command execution were delayed until the correct order is known. The application designers must consider whether this is an acceptable price to pay for the improved response time. For simple editing and display commands, users may well find it acceptable to see inconsistent states if the inconsistency is quickly corrected (which it is) and if this happens infrequently. The frequency with which this occurs, i..e, participants issue concurrent conflicting commands, can be minimized if participants negotiate externally by voice or use reservations on objects in the shared space.

6. Implementation Protocols

6.1. Initiation and Negotiation

Assume for simplicity that a given real-time conference is managed by a *controller* node through which all communication related to the conference passes. The controller node may be changed during the course of a conference, with some resynchronization. In order for users to participate in the conference, their workstations must be able to establish appropriate network connections with the conference controller. Establishing a network connection requires that either party, or both, be able to determine the network address of the other. While it is possible for network addresses to be communicated verbally outside the system (in advance of the conference, or when voice communication is established), this is cumbersome and error-prone and is best avoided except as a last resort. More typically, one or both of the following kinds of server may be used to determine the desired network addresses:

1. Users who wish to be available for a real-time conference register their workstation addresses at a *name server*, which the conference controller queries in order to determine the addresses of the participants to be invited. This mechanism was used in RTCAL, which only allows specifically invited users to participate in a conference.
2. The controller's network address is registered at a server that responds to queries about ongoing conferences. A participant's workstation can thus determine the address of the conference controller and establish communication if the participant wishes to join the conference. This approach is useful for "public" or group conferences, where it is inconvenient to explicitly enumerate the set of all possible participants and send each one a message.

Conference architectures in which all workstations communicate directly with each other will require that more network connections be set up. A workstation need only determine the address of any one other workstation (by any of the above means), from which it can then obtain the addresses of the other workstations in order to establish the remaining connections.

For a given real-time conference, some collection of *parameters* that govern the communication and user interface will need to be determined, e.g., size of virtual terminal screen or windows, or

maximum data transmission rate, or whether particular response time optimizations (described in Section 5.3) will be used. This can be done by *negotiation* among the controller and workstations: the controller proposes a set of parameter values, and then makes a selection based on the replies received from the workstations. Such negotiations can be performed at any time in a conference, including but not limited to the following:

- Network conditions change significantly enough to make the current parameter selection unworkable.
- A new participant joins the conference, and the parameters need to be adjusted to accommodate his workstation.
- A participant's workstation leaves the conference, or is removed because it crashed or was operating too slowly, allowing a better choice of parameter values that are acceptable to the remaining workstations.
- *Termination* of a conference is a special form of negotiation in which the controller issues a message requesting every workstation to leave the conference, and then awaits their replies for some reasonable amount of time.

6.2. Message Transport

We assume that the abstract operations and arguments that workstations need to communicate can be encoded into linear messages suitable for communication over a network, and can be properly decoded at the receiving end. (In RTCAL, we used Herlihy's algorithm for transmitting abstract data values,[17] which saved considerable programming effort by relieving us of the need to deal with low-level encodings.) The messages themselves can be transmitted over a *virtual circuit* that masks communication errors in order to provide reliable sequenced delivery. For certain types of messages, such reliable sequencing is not really needed, and the application might prefer to use *datagram* communication that is less reliable but gets new messages delivered faster because they are not held up by earlier messages awaiting retransmission. This is used in packet voice communication, and may be useful for certain special kinds of on-line information such as the position of a participant's pointer. Updates to such information can be sent in "absolute" form (i.e., the actual coordinates of the pointer rather than an offset from the previous value) so that they do not depend on previous updates. There is no need for a position report to be acknowledged or retransmitted because a new one will be issued very quickly. And, sequence numbers or timestamps can be used to ensure that delayed or duplicated old messages are correctly discarded by the receiver. This method was used in MBlink, based on Reed's original Blink protocol.

The number of connections that must be established in a conference with N workstations will depend on the communication architecture being used. If all communication goes through a central controller, the number of virtual circuits needed is linear (N-1, assuming the controller is itself a

workstation) in the number of workstations. If workstations need to communicate directly, however, the number of virtual circuits, N*(N-1)/2, grows as the square of the number of nodes. While we have argued that direct workstation communication can be useful in improving response time, it may instead have a detrimental effect on response if the overhead of a large number of virtual circuits causes the message transmission delay to increase; this may well happen for large values of N, say greater than 4 or 5.

For large conferences, an alternative to having a large number of virtual circuits is to exploit *multicast* communication. Some networks (e.g., Ethernet, token-ring, token-bus, and satellite) allow a single transmitted packet to be received by all or a specified subset of the machines on a given physical network. This can be exploited when all or many of the workstations in a conference are on the same network, but will not be useful when the workstations are scattered among different networks in an internetwork. This may be remedied in the future, as evidenced by recent proposals[18], and at least one operational protocol[19], for multicast across multiple interconnected networks.

Because of the unreliability of transmission, multicast datagrams are useful mainly for special kinds of data such as voice or pointers. For other kinds of data, real-time conferences need reliable sequenced delivery, which should be provided by a transport-level protocol that builds on any network-level multicast datagram service that is available. Currently, no general-purpose reliable multicast transport protocols are available that applications can use with the same ease as virtual circuits.* This is partly explained by the difference in complexity between two-party and multi-party communication; the latter introduces some new issues:

- Dynamically adding and removing nodes.
- Reducing the overhead of processing acknowledgments, which may all arrive at the same time, from multiple receivers.
- Performing flow control with respect to receivers who may be processing messages at different rates.
- Ensuring relative sequencing, where required, of multicast messages with respect to non-multicast messages sent to individual receivers.**

Until reliable multicast transport protocol standards which address the above are defined, real-time conferencing applications will either have to use multiple virtual circuits, or use unreliable multicast

---

*A step in this direction is the group communication facility of Stanford's V-system[20], with which the sender of a multicast message is informed when one receiver replies. The application must do more work if it wishes to ensure receipt by all processes in a group.

**Some applications also require that messages from different senders be delivered to all receivers in the same order. This is expensive to implement and is not needed for real-time conferences because some higher-level mechanism will ensure either that different senders update different objects or that out-of-order conflicting updates are resolved by undoing and reexecuting.

datagrams and implement an acknowledgment and retransmission scheme that is tailored to their needs.

6.3. Integrating Data and Voice

The real-time conferencing systems we have mentioned are meant to be used in multimedia conferences that include voice communication as well as on-line data communication. Since these systems (including ours) actually implement only the on-line data component of such a multimedia conference, voice communication must be established by a separate mechanism.*

We found in our experience that having separate communication channels for voice and data has some unforeseen benefits. Because a telephone connection tends to be more reliable and has lower delay than most computer communication, the voice channel was used to check that everybody could see the right thing on their screens and even to obtain a rough estimate of the elapsed time between a participant typing a character and the echo appearing on another participant's screen. If the voice channel is established (using on-line phone directories and automatic dialers) by the same computer system that establishes the data connection, it will be possible to provide an integrated user interface so that participants need issue only one set of commands for conference setup.

Given that packet voice conferencing can be implemented on a computer network (it was demonstrated some years ago on the Arpanet[21], and more recently on the DoD Internet[22]), it may well be feasible to transmit both voice and data over a single communication channel or network. Not only will the participants be relieved of multiple connection setup (as with automatic phone number lookup and dialing), but the system can also provide additional feedback such as indicating which participant is speaking. It will also be possible to address a subgroup of participants on the voice channel, should that be desired.

The transmission characteristics and requirements of voice and data are quite different, and not all computer networks will necessarily be able to provide the desired level of service for both. Voice transmission requires short bounded transmission delay and low variation in delay, and can tolerate small amounts of data loss or corruption, while on-line data typically requires reliable delivery which can only be achieved by accepting longer and more variable delays. Nonetheless, to whatever extent it is feasible, using a single communication channel will be more economical. In some situations (e.g., in remote field locations, or with personal computers communicating over a single telephone line) there may be only one communication channel available. The design guidelines and implementation

---

*It is possible to use these systems without voice communication; participants may type text messages to each other, but this is an extremely low-bandwidth and ineffective means of communication that does not allow spirited discussion and argument.

techniques we have described can be adapted to integrated systems that perform both voice and data conferencing over a single network.

7. Conclusion

This paper examined some prototype real-time conferencing systems and generalized from them to guidelines for the design and implementation of future real-time conferencing systems. Access to and manipulation of on-line information can enhance the effectiveness of meetings, both remote and face-to-face. However, further research is needed in evaluating which of the available functions are useful and when, and what the long term impact of this new mode of communication will be on the effectiveness and morale of groups of people working together.

We have identified two constrasting approaches to constructing real-time conferencing systems: sharing existing single-user interactive programs, and writing new application programs that interact with multiple users and can treat each one differently (in terms of command context or access control) if necessary. The former approach is easy to implement but limited in functionality, while the latter can provide better functionality and performance but requires more work to implement. Some systems may be able to use a mixture of the two approaches.

For the immediate future, it is probably more important to build simple real-time conferencing systems quickly, even if the functionality is limited, so that users will have an opportunity to experiment with this new technology. Rapid development and deployment will require reusing existing application software, with little or no modification, and using simple communication architectures that can be trusted to work correctly rather than extremely efficiently. As experience is gained with the use of such systems, it will become clear which kinds of more advanced functions users feel the need for, and where the implementation bottlenecks are to which the performance enhancement techniques we have outlined can be fruitfully applied. It will then be appropriate to construct more complex but more powerful and more efficient systems that remove the observed limitations of existing systems. In the meantime, we expect to see continued development of software engineering "toolkits" and communication protocol standards. We hope that real-time conferencing requirements will be recognized early and incorporated into these tools so that they can be used for building the next generation of conferencing systems.

Acknowledgments

We would like to thank: Michael Greenwald, Maurice Herlihy, David Reed, Larry Rosenstein, and Lixia Zhang for their voluntary and involuntary contributions to the prototype systems described; Harry Forsdick for providing information about existing conferencing systems; and the editor and reviewers for their constructive suggestions for improvement.

This research was sponsored by the Defense Advanced Research Projects Agency, and was monitored by the Office of Naval Research under contract number N00014-83-K-0125.

REFERENCES

1. D.A. Henderson and T.H. Myer, "Issues in Message Technology", *Proc. Fifth Data Communications Symposium*, September 1977, pp. 6-1 to 6-9.

2. S.R. Hiltz and M. Turoff, "The Evolution of User Behavior in a Computerized Conferencing System", *Comm. ACM*, Vol. 24, No. 11, November 1981, pp. 739-751.

3. D. Tsichritzis, "Form Management", *Comm. ACM*, Vol. 25, No. 7, July 1982, pp. 453-478.

4. S. Sluizer and P.M. Cashman, "XCP: An Experimental Tool for Supporting Office Procedures", *Proc. First IEEE Intl. Conf. Office Automation*, December 1984.

5. B.I. Kedziersky, "Knowledge-Based Project Management and Communication Support in a System Development Environment", *Proc. 4th Jerusalem Conference on Information Technology*, May 1984.

6. D.C. Engelbart and W.K. English, "A Research Center for Augmenting Human Intellect", *Proc. Fall Joint Computing Conference*, AFIPS Conference Proceedings Vol. 33, December 1968, pp. 395-410.

7. D.C. Engelbart, "Toward High-Performance Knowledge Workers", *Office Automation Conference Digest*, AFIPS, April 1982, pp. 279-290.

8. M.H. Brown and R. Sedgewick, "Techniques for Algorithm Animation", *IEEE Software*, Vol. 2, No. 1, January 1985, pp. 28-39.

9. W. Pferd, L.A. Peralta, and F.X. Prendergast, "Interactive Graphics Teleconferencing", *IEEE Computer*, Vol. 12, No. 11, November 1979, pp. 62-72.

10. S.K. Sarin, "Interactive On-Line Conferences", technical report TR-330, M.I.T. Laboratory for Computer Science, December 1984, (Contact LCS Publications, 545 Technology Square, Cambridge, MA 02139.)

11. G. Heffron, "Teleconferencing Comes of Age", *IEEE Spectrum*, Vol. 21, No. 10, October 1984, pp. 61-66.

12. T. Kamae, S. Ohtsuka, and Y. Sato, "Sketchfax - A Terminal Having Telewriting and Facsimile Capabilities", *Proc. International Conference on Communications*, IEEE, June 1983, pp. D3.4.1-D3.4.5.

13. G. Foster, "CoLab, Tools for Computer-Based Cooperation: A Proposed Research Program",

technical report UCB/CSD-84/215, Univ. California, Berkeley, Computer Science Division, December 1984.

14. I. Greif, "PCAL: A Personal Calendar", Technical Memo TM-213, M.I.T. Laboratory for Computer Science, January 1982.

15. K.A. Lantz and W.I. Nowicki, "Structured Graphics for Distributed Systems", *ACM Transactions on Graphics*, Vol. 3, No. 1, January 1984, pp. 23-51.

16. R.F. Sproull and D. Cohen, "High-Level Protocols", *Proceedings of the IEEE*, Vol. 66, No. 11, November 1978, pp. 1371-1386.

17. M. Herlihy and B. Liskov, "A Value Transmission Method for Abstract Data Types", *ACM Trans. Programming Lang. and Systems*, Vol. 4, No. 4, October 1982, pp. 527-551.

18. L. Aguilar, "Datagram Routing for Internet Multicasting", *Proc. Symposium on Communications Architectures and Protocols*, ACM SIGCOMM, June 1984, pp. 58-63.

19. J.W. Forgie, "ST - A Proposed Internet Stream Protocol", Internet Experimental Note IEN-119, M.I.T. Lincoln Laboratory, September 1979.

20. D.R. Cheriton and W. Zwaenepoel, "Distributed Process Groups in the V Kernel", *ACM Transactions on Computer Systems*, Vol. 3, No. 2, May 1985, pp. 77-107.

21. D. Cohen, "A Protocol for Packet-Switching Voice Communication", *Computer Networks*, Vol. 2, No. 4/5, September/October 1978, pp. 320-331.

22. C.J. Weinstein and J.W. Forgie, "Experience with Speech Communication in Packet Networks", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-1, No. 6, December 1983, pp. 963-980.

Figure Captions

Figure 1. Display screen of participant in an RTCAL real-time conference (or *session*), showing the following windows (from top to bottom): the *informational* window, which lists the program version, current date and time and system load, and this participant's name, and indicates the character ('CTRL-↑') to be typed for entering conference control commands; the *summary* window describing the meeting to be scheduled, and indicating which of the desired participants are present in this session and who the chairperson and current controller (holder of the floor) are; the *events* window which presents notifications of important events (participants joining and leaving, control passing, calling a vote, and text messages between participants); the main *calendar* windows, shared and private, displayed side-by-side; and the *command* window, showing a command prompt, the current contents of the command buffer, and error messages or other feedback in response to completed calendar commands.

Figure 2. Voting on a proposal in RTCAL. Each participant is prompted for his vote. A default vote is suggested based on whether or not this participant has a conflicting appointment in his private calendar. The participant can accept the default by typing a carriage-return, or can override it by erasing and typing over. Participants vote in parallel, and their votes are tabulated in the shared window as they arrive. The voting terminates when votes have arrived from all participants, or when the chairperson decides not to wait any longer for participants who are slow to respond.

Figure 3. Menu of RTCAL control commands, obtained by typing 'CTRL-↑' followed by '?'. Each single-character code indicates a different control command, which may be available to the chairperson only (preempt control, terminate), to other participants only (request control, quit), or to any participant ("escape" temporarily, send text message to all or a specified participant). Any participant, whether or not he holds the floor, may enter a control command at any time. The control command prompt and menu overlay the participant's private calendar window, and are not seen by other participants.

Figure 4. Sample MBlink screen display. The shared bitmap graphically depicts the state of a hypothetical network being simulated (not the network on which this conference itself is running). The position of each participant's mouse is indicated by overlaying a distinctive pattern on the shared bitmap. Each participant in addition sees a pattern (an arrow in this case) showing the locally tracked position of his own mouse. The mouse of the participant on the left is currently stationary. The mouse of the participant on the right is currently moving (or has just moved) and the remotely echoed pattern has not yet caught up.

Figure 5. Contrasting software architectures for real-time conferences. (a) When sharing a single-user application, each participant's workstation holds only a screen image of the application data; it does not have access to the application objects. (b) With a multi-user application, a participant's workstation holds copies of application objects and derives the screen image locally. A multi-user application may also allow direct communication between workstations, not shown in the figure.

```
RTCAL 3.2 ctrl-↑ for control cmds  12-4-82 11:52:07 Load=8.7 SARIN

scheduling "thesis defense"  uncommitted (2hrs, 12-25 to 12-31-82)
-session-   GREIF         LICKLIDER     SARIN         HAMMER
-Running-   Chairperson   IN-Session    Controller    Waiting

YOU have received control

Monday 27 December 1982                 Private calendar
Merge of GREIF LICKLIDER SARIN           Joe's birthday
 9:30   XXX                              9:30
10:00   XXX                             10:00
10:30                                   10:30
11:00                                   11:00
11:30                                   11:30
12:00                                   12:00
12:30   XXX                             12:30    lunch
13:00                                   13:00
13:30                                   13:30
14:00   XXX                             14:00    Darpa meeting
14:30   XXX                             14:30     xx
15:00   XXX                             15:00

 COMMAND> propose 10:30_
```

Figure 1

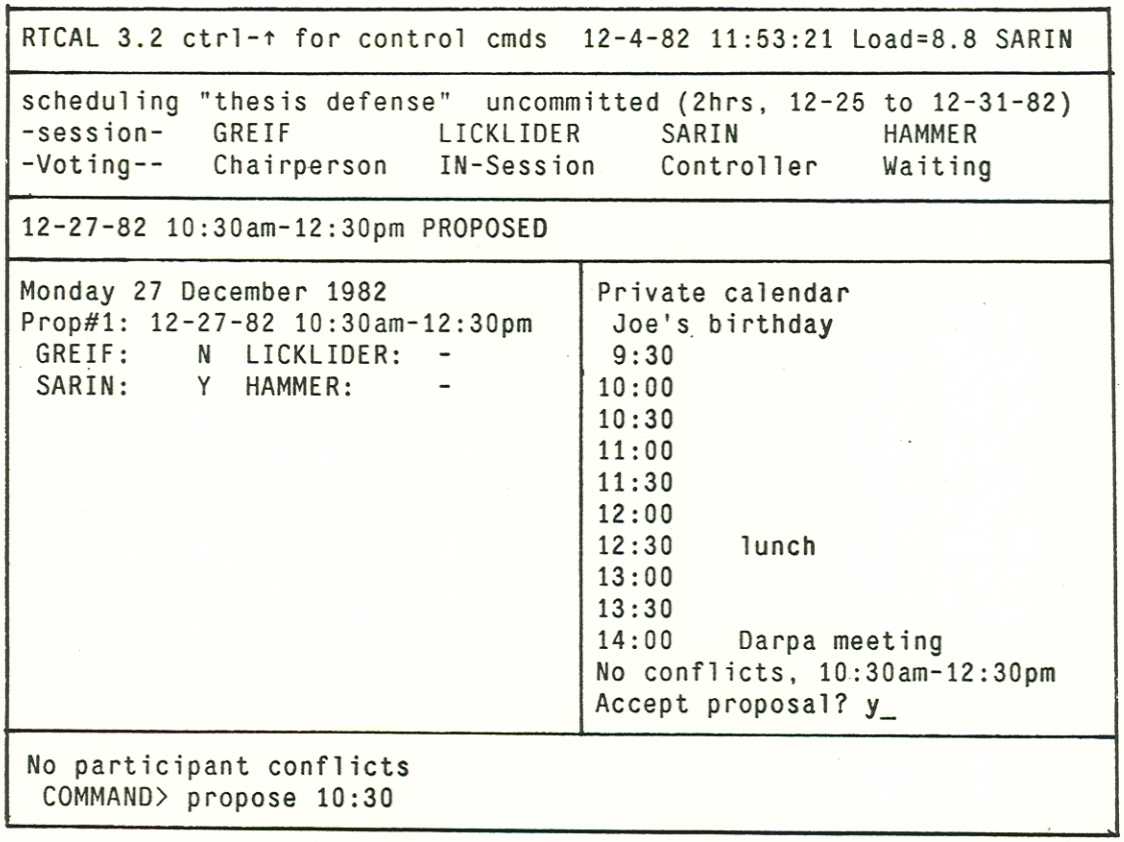
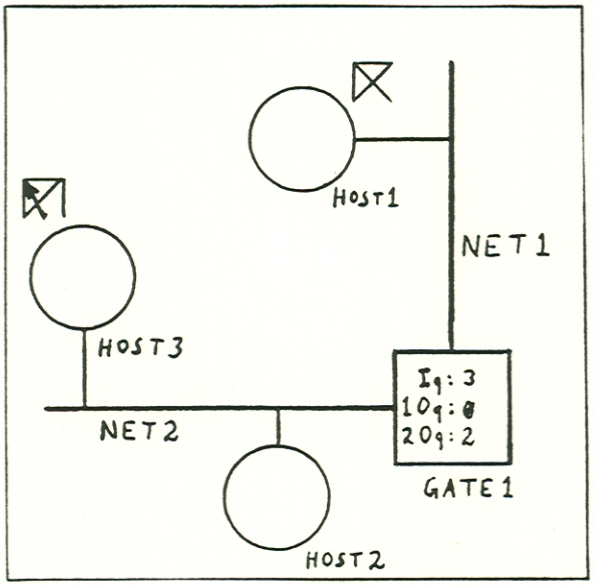

RTCAL 3.2 ctrl-↑ for control cmds 12-4-82 11:53:21 Load=8.8 SARIN
scheduling "thesis defense" uncommitted (2hrs, 12-25 to 12-31-82)
-session- GREIF LICKLIDER SARIN HAMMER
-Voting-- Chairperson IN-Session Controller Waiting
12-27-82 10:30am-12:30pm PROPOSED
Monday 27 December 1982
Prop#1: 12-27-82 10:30am-12:30pm
GREIF: N LICKLIDER: -
SARIN: Y HAMMER: -
Private calendar
Joe's birthday
9:30
10:00
10:30
11:00
11:30
12:00
12:30 lunch
13:00
13:30
14:00 Darpa meeting
No conflicts, 10:30am-12:30pm
Accept proposal? y_
No participant conflicts
COMMAND> propose 10:30

Figure 2

```
RTCAL 3.2 ctrl-↑ for control cmds  12-4-82 11:54:51 Load=8.2 SARIN

scheduling "thesis defense"  uncommitted (2hrs, 12-25 to 12-31-82)
-session-   GREIF          LICKLIDER     SARIN          HAMMER
-Running-   Chairperson    Controller    IN-Session     Waiting

LICKLIDER has received control

Tuesday 28 December 1982                 E: Escape (leave temporarily)
Merge of GREIF LICKLIDER SARIN           A: Terminate session (chp only)
 9:30                                    Q: Quit session (partc only)
10:00   XXX                              R: Request control
10:30   XXX                              G: Give control
11:00   XXX                              P: Preempt control (chp only)
11:30                                    L: List requests (chp only)
12:00                                    M: send text Message
12:30   XXX                              X: eXclude partc (chp only)
13:00   XXX                              C: Catch up (partc only)
13:30                                    B: report Bug, or remarks
14:00                                    H: read Help file
14:30
15:00   XXX                              Control command: _

licklider> next
```

Figure 3


HOST1
NET 1
HOST3
NET2
Iq: 3
10q: 0
20q: 2
GATE 1
HOST2

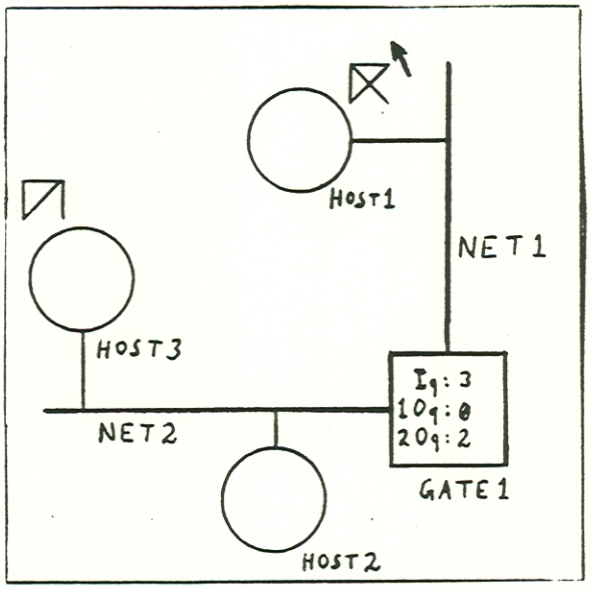

HOST1
NET 1
HOST3
NET2
Iq: 3
10q: 0
20q: 2
GATE 1
HOST2

Figure 4

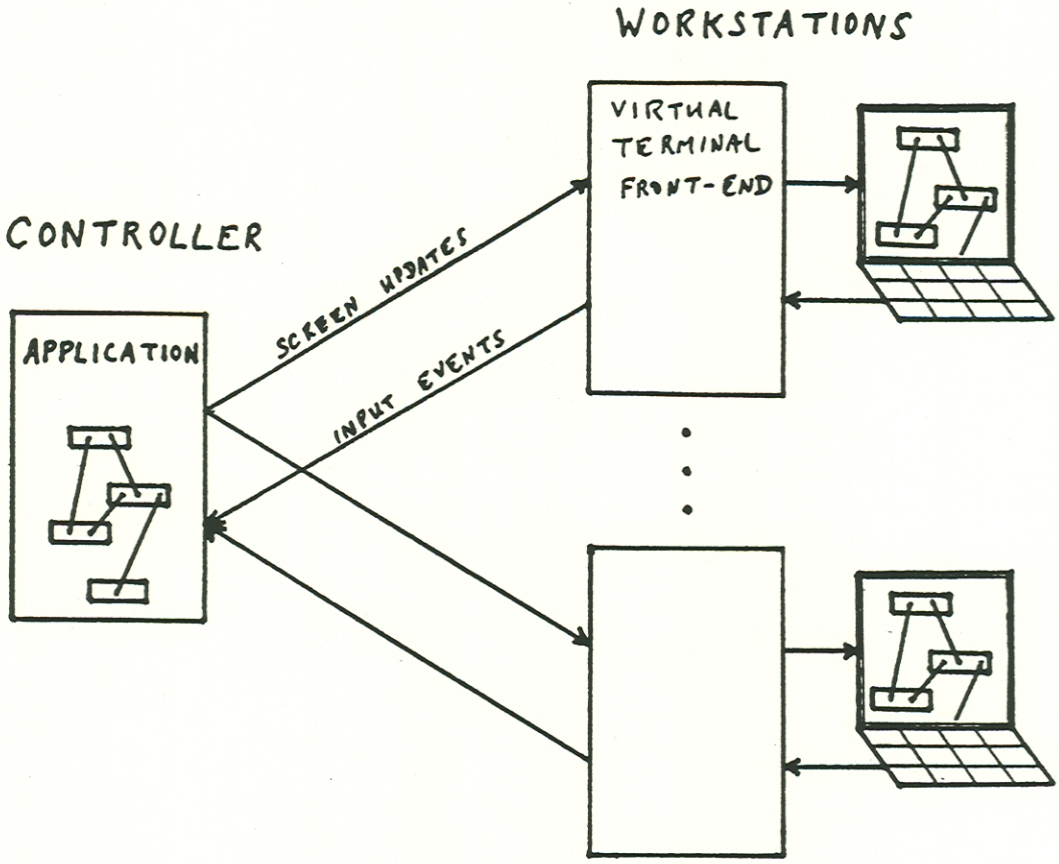

WORKSTATIONS
VIRTUAL TERMINAL FRONT-END
CONTROLLER
APPLICATION
SCREEN UPDATES
INPUT EVENTS

Figure 5(a)

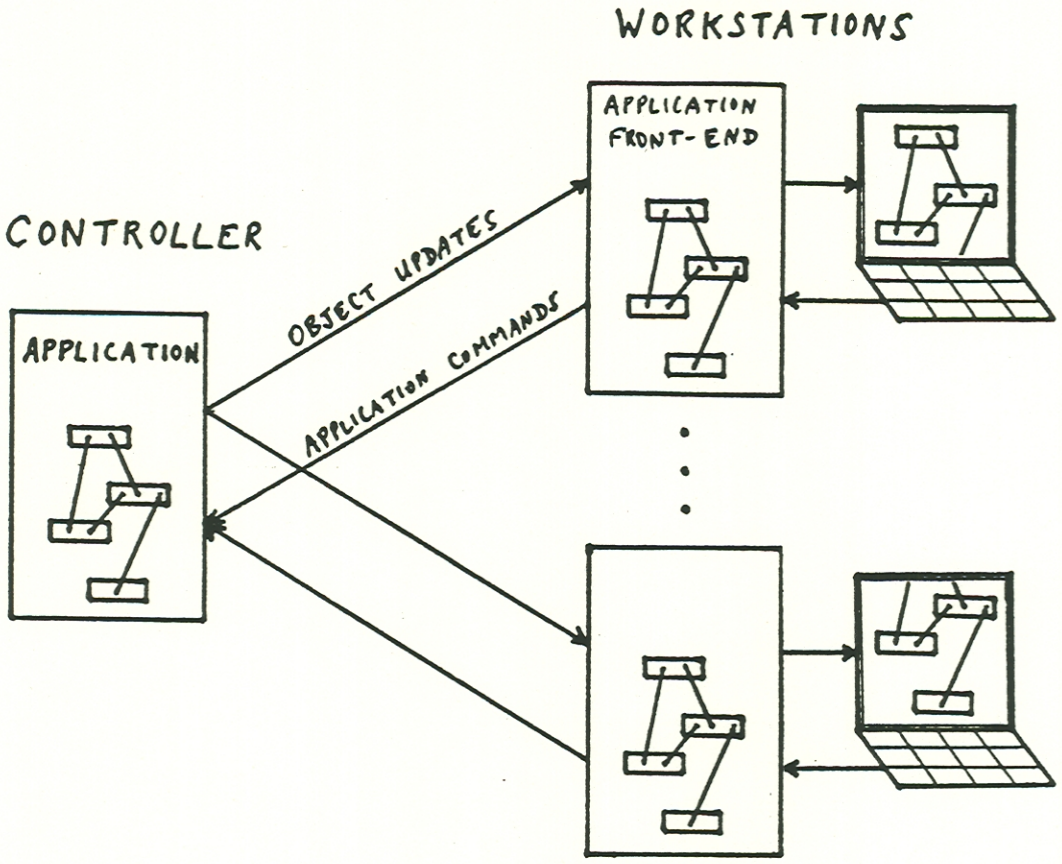

WORKSTATIONS
APPLICATION FRONT-END
CONTROLLER
OBJECT UPDATES
APPLICATION COMMANDS
APPLICATION

Figure 5(b)