Argo: A System for Distributed Collaboration

Hania Gajewska, Jay Kistler, Mark S. Manasse, and David D. Redell

Systems Research Center
Digital Equipment Corporation
130 Lytton Avenue
Palo Alto, CA 94301

ABSTRACT

The goal of the Argo system is to allow medium-sized groups of users to collaborate remotely from their desktops in a way that approaches as closely as possible the effectiveness of face-to-face meetings In support of this goal, Argo combines high quality multi-party digital video and full-duplex audio with telepointers, shared applications, and whiteboards in a uniform and familiar environment. The shared applications can be unmodified X programs shared via a proxy server, unmodified groupware applications, and applications written using our toolkit. Workers can contact each other as easily as making a phone call, and can easily bring into a conference any material they are working on. They do so by interacting with an object-oriented, client/server conference control system. The same conference control system is used to support teleporting, i.e. moving the desktop environment from one workstation's display to another (for example, from office to home). This paper describes the system we have built to test the hypothesis that the effectiveness of remote collaboration can be substantially impacted by the responsiveness of the interaction media.

KEYWORDS

Desktop videoconferencing, shared workspaces, collaboration, multi-media, groupware.

INTRODUCTION

There are many reasons why people need to collaborate remotely. Pollution and congestion of the highways dictate reducing commuter traffic[5,6]. Corporations are increasingly distributing their offices. The rising number of two-career families introduces added constraints on residence location, increasing the average distance from home to office. Recent government policies in issuing contracts encourage cooperation between potentially distant companies.

Personal computers, faxes, and conventional telecommunication help solve these problems for autonomous workers. However, they address neither the needs of the large number of workers for whom much of the workday is spent in collaborative efforts such as joint design or strategic meetings, nor the need to provide instruction and assistance to mostly autonomous workers.

The goal of the Argo system is to allow a group of users to collaborate from their desktops as effectively as if they were face-to-face. Our focus is on groups that work together, rather than on lectures, so our system is optimized for groups up to about eight people. Previous efforts to support remote collaboration, both research prototypes and products, have established beyond any doubt that while teleconferencing is extremely useful, exact simulation of a physical meeting using computers, audio, and video is not a reasonable goal. However, in most systems to date, the inherent limitations of the media have been entangled with more transient limitations of equipment cost and communication bandwidth. Argo exploits fast machines and networks, when available, to deliver low latency, high quality audio and full frame-rate video; it allows us to test the effectiveness of remote collaboration without impediments introduced by equipment or network limitations.

In pursuing this theme, we have intentionally declined to explore ways to use computers to control, restructure, or otherwise improve upon a traditional meeting. That idea is interesting, and has been investigated in systems like Colab[28] and Capture Lab[18], but it is largely independent of our work. If some notion of "augmented meeting" appears especially valuable for remote collaboration—e.g. to compensate for media limitations in some way—it should be possible to integrate it into an Argo-like system. We are also restricting our investigation to desktop systems, and are not investigating the coupling of desktop systems with meeting-room systems; although such interaction might present interesting challenges once desktop systems are better understood, it is outside the current scope of our project.

In support of remote collaboration, Argo provides three basic kinds of tools: real-time digital audio and video, general sharing of arbitrary single-user applications and groupware, and telepointing/telepainting tools for gesture and annotation in any shared window. These tools are integrated into a unifying conference control framework, providing a coherent notion of a conference and simple facilities for creating, joining, suspending, and leaving conferences using both phone call and conference room paradigms. Other tools we may integrate into future versions of Argo include scanned images and stored audio and video.

A central research hypothesis of our project is that the effectiveness of remote collaboration can be substantially impacted by the responsiveness of the interaction media. Many users of existing low bandwidth teleconferencing systems express the subjective view that, above a modest threshold, limiting audio and video quality doesn't matter very much, and is at worst a temporary annoyance "until you get used to it." We hypothesize that this is incorrect, based on studies like that of Whittaker et al.[31], who found that high quality video, while still inferior to direct face-to-face interaction, caused substantial changes in user interaction. The importance of audio quality is less controversial, but the impact of very high quality audio on group collaboration still remains to be measured in a real system. We believe responsiveness will also be shown to be important in shared application software and in gesture and annotation tools.

In this paper we describe the system that we have built to test the above hypothesis; we do not attempt to prove it here, because at our current level of deployment, our data is insufficient. We first present the user's perspective of Argo's component technologies: conference management, shared applications, telepointing and telepainting, and audio and video. In the following section we outline the implementation of the first three of these components; audio and video are discussed in a separate paper[4]. We then compare our system to other efforts. Finally, we outline our plans for future research.

COMPONENT TECHNOLOGIES

Conference Management

The Argo conference manager supports two paradigms for establishing and joining conferences. One model is that of a phone call. A user chooses who to call from a directory of users (or an abbreviated list of "favorite" users—the speed-dialing option). The call recipient is notified with a ring and a popup window identifying the caller and allowing the callee to accept or reject a call. Accepting the call can, at the callee's option, either establish a conference or add the caller to the callee's existing conference; rejecting a call can look like a no answer to the caller, or carry back a message indicating the reason for rejection ("I'm busy for the next 10 minutes").

The other model for establishing a conference is that of entering a meeting room. The user selects the conference to join from a list of existing conferences, or creates a new one by giving it a new name. Conference names can either be published or not; the names of unpublished conferences are supplemented with computer-generated capabilities to prevent intrusion. At present, the names of meeting-room conferences are published while the names of phone-call conferences are not; in the future, we intend to allow users to restrict the notification of conference existence and the corresponding capabilities to members of access control lists supplied at conference creation.

Regardless of how it was created, members of an existing conference can add other users to it using the phone call paradigm with the verb *add* instead of *call*. Whenever either the caller or the callee could be added to an existing conference (i.e., when the caller "adds" the callee while initiating the call, or the callee "adds" the caller when accepting the call), the callee is given the option of *conferring* with the caller. While conferring, the caller and the callee are placed in a private audio-only conversation, with the decision whether or not to join the existing conference

deferred. Conferring is meant to be a lightweight operation, in that no screen configurations are changed.

There are two ways of getting out of a conference: *leaving* the conference and *suspending* it. When a user leaves a conference, the user's state in the conference is discarded: positioning of video and application windows is lost, and the applications that the user brought into the conference are usually transferred with the user out of the conference. When the number of users remaining in the conference drops below some threshold, the conference is terminated and all applications that remain also terminate. This threshold is two for a call and one for a meeting; a user can set it to zero to create permanent meeting rooms.

Suspending a conference is like putting a phone call on hold or temporarily stepping out of a meeting. The conference continues without the suspended participant's audio or video, but all of her applications remain in the conference with the expectation that she will return. For the suspending participant, all video, audio, and applications belonging to that conference disappear.

A user may have several conferences suspended at any given time; she is active in exactly one conference, which may be the user's *personal conference*—the conference a user is in when not in any other conference. Applications run in the personal conference are sharable, although typically they are not shared; they can become shared as conference membership changes. To manage activation and deactivation, the user is presented with the name of the current conference and a list of the suspended conferences. This list is ordered: its first element is the conference in which the user will be active upon suspending or leaving the current conference. The user may click on any conference name in the list in order to activate in that conference.

We make the restriction of allowing just one active conference per user in order to preserve transitivity and symmetry of audio, video, and application interaction. Without this restriction, users might hear one side of a conversation or descriptions of interactions with applications they can't see. We decided that this would be too confusing.

The conferencing system provides an easy means of identifying ownership of objects on the screen via the use of text and heraldry. All applications in a given conference are identified by a distinctive pattern in the title bar, which matches the pattern behind the name of the conference in the conference control manager and the pattern surrounding all of the video windows in that conference. Similarly, all windows representing or belonging to a given user are identified with that user's color: the border around the user's video window matches the border of the user's shared applications and the user's telepointer. The system's choice of color and pattern is uniform across displays for all members of a conference, and uniform over time for any user or conference. Thus users can learn to associate Mary (and Frank) with the color blue, and their weekly project meeting with a brown title bar pattern.

Audio and Video

The most important aspect of collaboration is the ability to speak to and hear one another clearly. Argo supplies full duplex multi-party audio, connecting all active members of a conference. The primary goals of the audio system are that the sound be uninterrupted (so that users don't have to repeat utterances because of dropout) and transmitted with low latency (so that users don't end up interrupting one another or modifying their behavior to avoid such interruptions). In the current version of our

system, users use headsets to avoid echo problems; we plan to use echo cancellation systems as their functionality improves.

The audio stream adapts its latency characteristics to deliver uninterrupted speech. Bursty network traffic or intermittent access to the processor can force the use of more buffering to avoid dropout. The resulting increase in latency is tolerable only below about 200 ms. We are focusing on network technology that avoids imposing substantially more latency than this, since it is known to be disruptive to natural conversation[31]. In severely band-width-limited connections, our audio hardware can attach to a telephone line, and the software is configurable so that the audio mixing can be done at either end of the line.

The typical conference participant's workstation includes a video camera and hardware for simultaneous JPEG compression and decompression of video. Each user can see the other participants of her active conference in separate video windows, in color and at full frame rate (subject to hardware limitations discussed by Berc, et. al.[4].) There is no artificial limit imposed on the number of active conference participants. The sizes and layout of video windows are independently controlled on each conference member's display. This allows a user to arrange the video windows to fit her usual screen layout, and to single out currently important conference participants by making their video images larger (as when one participant is giving a presentation).

We aim to provide high quality full rate video, since it is best for facilitating natural, animated discussion. When processor or network resources are insufficient, the frame rate drops, but the image is otherwise undegraded. While inadequate for intensely visual communication, slow frame rate still provides the conferees with feedback from other participants which may be sufficient for many types of collaboration. For truly degraded networks or workstations without video-capture hardware, the video system presents canned still images in place of the video. This flexible video frame rate provides a testbed for our hypothesis about the importance of video quality for different types of collaborations.

Shared Applications

Our most important goal in integrating shared applications into the conferencing system is to preserve as much of the user environment as possible. Thus, any application can be shared, and can be moved in and out of conferences—including applications that were already running at conference startup. To move an application into the current conference, the user selects it, either explicitly or from a menu, and commands it to join the current conference; new applications automatically start up in the current conference. Our framework is flexible enough so that sharing can be provided by a proxy window server, a toolkit, or the application itself (as in explicit groupware). The sharing mechanism is transparent to the user. The conference-specific applications such as video and the conference control panel fit naturally into the environment and are controlled by the user's ordinary window manager.

To control input to applications, we have chosen a very light-weight model of floor control. We believe that, in the presence of full-duplex audio, groups of up to six or eight participants can achieve such control through verbal negotiation, and system-imposed floor control is obtrusive; therefore, we aim to provide the illusion that any user can provide input to a shared application at any time. However, since non-groupware applications don't know about multiple users, we restrict the input stream to be consistent with the actions of a single user's keyboard and mouse. At most one user holds the floor at any time, and only her keystrokes and mouse motion events are relayed to the application. To take the floor, a user clicks with the mouse in a shared window at a time when the floor holder's mouse has no buttons down. This floor control mechanism is designed to keep the input stream consistent; we provide no mechanism intended to restrict users from taking the floor. To help users decide whether their keystrokes will be accepted, the color of the cursor in a shared application reflects the color of the user holding the floor.

Given our infrastructure, we can move a user's environment from one workstation to another, as in office to home. We do this by adding the user at her new location to all the conferences that she is in, including her personal conference, and removing her old location. Since a user's personal applications run in her personal conference, this allows a user to migrate her entire windowing environment. We call this *teleportation*; we know of no other system that integrates teleportation with collaboration (cf.[13]).

When a user leaves a conference, applications started by that user can remain in the conference, follow the user to the next conference, or be deleted. Similar questions arise when a user suspends a conference and when a conference terminates. In our present system, different shared tools behave differently; we have not yet determined which behavior should be the default. We believe that no single policy is appropriate for all applications: it's less worrisome to leave behind static previewers of one's documents than writable editor buffers. In the future, we intend to allow users to dynamically set the desired behavior.

We now examine the three different mechanisms for sharing applications.

Window-system-based replication

Although group-aware applications offer multiple users greater flexibility, functionality, and transparency than single-user applications, few common applications are written as explicit groupware. Nonetheless, applications users might have seen on a single display while sitting in an office together must be capable of being shared between multiple remote displays. To this end, we use an off-the-shelf shared X server, shX[2], with an agent to support our models of conference and floor control. In the process, we removed shX's native user interface for controlling sharing. The sharing provided at the window system level is imperfect; we discuss this in more detail in the backstage section on sharing. Still, the X sharing engine that we've chosen is acceptable to our users. We are exploring window-system-level sharing for Windows NT[9].

Toolkit-based replication

The window applications that we write at SRC use Trestle[17], a Modula–3-based[20], window-system independent, object-oriented toolkit, currently implemented on top of X[26] and Windows NT. Trestle was designed to make it easy for applications to handle screen layout, and to facilitate teleportation. Because more of the program semantics are available at the toolkit level than at the window-system level, we can provide a more seamless illusion of window sharing. For example, if an application is being shared between a color display and a black and white display, Trestle will make it appear in color on the color display, and in the application designer's best black and white design on the black-and-white display. In addition, the toolkit approach is more malleable for our experiments with sharing behavior.

Among the many single-user Trestle tools that we share are our mail reader, an editor, and a simple drawing tool.

Groupware

Some applications can gain extra leverage from the knowledge that they are being shared; for example, an editor that is group-aware can support separate insertion points for each user, so that multiple users can edit different portions of the document simultaneously. We expect that more and more commercial applications will evolve towards group awareness. In support of this, Argo makes it easy for applications to handle sharing themselves. We have a few examples of explicit groupware in operation: our audio/video applications, our telepointers, and wb, a shared whiteboard program that is part of the Internet Mbone[16] application suite. The latter was integrated into Argo without change to the program itself, via the *agent* mechanism described later. We expect that other off-the-shelf groupware will be integrated similarly via agents or through simple source modifications to the membership-control portions of the applications.

Telepointers and telepainters

Since people working together at a computer or a whiteboard often communicate by pointing, we need to provide for this in our teleconferencing environment. To that end, each conference participant's cursor acts as a *telepointer* which can be used to point at shared applications. The telepointer is visible to other conference participants only when it's over a shared window and appears as an arrow, in the color of its owner and augmented with her name, moving in their copies of the shared window. We use color for its ability to convey information rapidly, and text to disambiguate similar colors and to assist color-blind users.

Argo provides *active telepointing*: no special action is required to initiate pointing, each participant's cursor behaves normally, and everyone can see everyone else's cursor position in shared windows. We know of no other system that provides active telepointing.

The fact that telepointers are visible whenever their owner's cursor is in a shared window might cause excessive clutter. However, we find that for our target group size of up to eight users this is not a problem. Additionally, since telepointers are visible *only* in shared windows, users can move their pointers out of such windows to reduce clutter.

Telepainting is like telepointing, except that it leaves behind a trail of user-identifying ink. We need telepainting because it's hard to point at large regions—or at more than one thing—with a single pointer, and because it is useful to make temporary annotations on running applications. For example, in discussing the layout of a page one might want to quickly mock-up the desired re-arrangement of blocks of text and figures. In our implementation, telepainting is provided by extending the functionality of the telepointer to draw and erase.

Note that Argo's scheme for annotation makes different tradeoffs than systems such as Sun Microsystems' ShowMe, which provides more powerful annotation tools but applies them only to captured snapshots of screen images; we feel that careful annotation is less important on live applications, since users can actually perform the actions their annotations would have described. For example, instead of highlighting a typo for later correction by the document owner, a user can simply correct it.

BACKSTAGE

We now present a brief overview of the architecture and implementation of Argo and its components. Figure 1 shows the major system components: conference control, shared applications, and audio and video. In the figure, arrows represent communication paths between clients and servers; the arrowheads represent the principal direction of data flow. The next two sections discuss the first two components; the third section discusses telepointers. Audio and video are addressed in a separate paper[4].
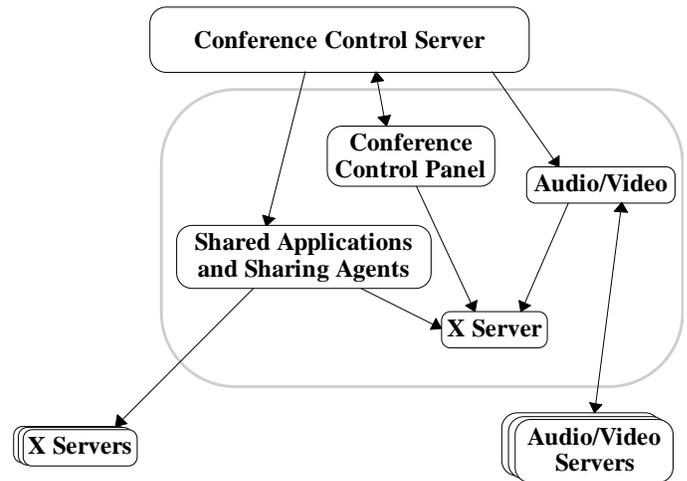

Figure 1. The top-level structure of Argo. The gray box represents a user's workstation.

Conference Control

Argo's conference control system is built using SRC's Network Objects[21], an object-oriented client/server model, for communication. We chose network objects because it provides a convenient mechanism for rapidly building network protocols. The conference control server maintains objects representing users, conferences, and members. A member object represents the pairing of a user and a conference in which the user is a member. Clients can associate private and public properties (arbitrary data structures) with any object; they can use the properties to record the state of relevant parameters such as window sizes or audio volume. Clients can register for event notification, e.g. users joining or leaving a conference. Client programs respond to events by, for example, changing which windows are visible or which audio signals are audible. One particular client is the conference control panel, which uses the object/event model to provide the user interface to the conference management functions described earlier.

To see how this works, consider what happens when a user joins a conference. First, a new member object is created from the user and conference objects. The member is made active, and all interested clients receive an activation event. The video clients of all users in the same conference react by adding the video image of the new user to their screens; the new user's video client reacts to the activation of a member containing its user by enumerating the other members of the conference and displaying video from each user. Shared applications in the conference react by displaying on the new user's screen.

Many teleconferencing systems use a more loosely-coupled approach, dispensing with the client/server architecture in favor of a pure peer-to-peer model. In some, the naming functionality is completely eliminated and connections are established via low-level addresses, such as ISDN phone numbers. In others, names of conferences are "advertised" and cached, forming, in effect, a virtual database held by the clients with only loose notion of any global consensus. We see our server as a precursor of functionality which will eventually augment the global naming databases already needed for widespread electronic mail, computer networking, and so on.
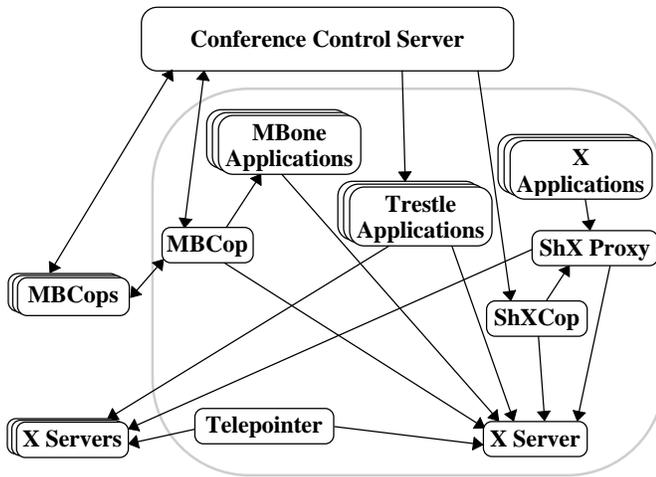
Application Sharing

Figure 2. Shared applications in Argo. The gray box represents a user's workstation.

Argo currently supports sharing of three classes of application: Mbone applications, applications written using the Trestle windowing toolkit, and generic X applications. In each case some kind of *sharing agent* or "cop" establishes callbacks with the conference control server to track conference membership. Applications available for sharing are always attached to some conference, which may be the user's personal conference. As a user enters or leaves a conference, or when she suspends or activates a conference, the agent creates, destroys, unmaps, or maps the appropriate application windows on that user's display. When a conference is deleted, the agent either terminates applications attached to that conference or reattaches them to different conferences. As described above, in our current implementation different agents behave differently; this allows us to experiment before settling on a default behavior.

The three application sharing mechanisms are illustrated in Figure 2 and are discussed in more detail below.

Shared X

Our most general sharing mechanism uses a shared X server to multiplex the windows of ordinary X applications onto multiple workstation displays. The shared X implementation we use, shX, is a *proxy X server*; that is, it looks like an X server to applications and like an application to the X server on each workstation. A crucial property of shX is that it supports late joiners, unlike other shared X implementations such as Jaffay's xtv from the University of North Caroline, or Bazik's xmx from Brown University..

The standard shX includes an external user interface for controlling window sharing and chalk-passing. In Argo, we replaced this with an Argo sharing agent called *shXcop*. An instance of shXcop runs on each workstation and watches for newly started X applications, for chalk-passing events, and for conference membership changes on behalf of its user. For new X applications, shXcop instructs shX to replicate the application to the displays of all members of its user's current conference. As conference membership changes occur, shXcop responds by mapping and unmapping shared windows, and by instructing shX to change the replication sites for affected applications.

Due to the nature of the X protocol and the way that shX uses it to implement replication, shared applications run uniformly slower for all participants. The slow-down is typically negligible, except at application start-up and for applications that perform complex animation.

Trestle

The Trestle toolkit supports window replication and floor control internally and independently of Argo and shX. We extended Trestle to communicate directly with the Argo conference control server to control sharing. Providing replication functionality within the toolkit rather than externally has two advantages. First, it makes it easier to coordinate resources such as fonts and colors across different screen types, since Trestle applications have long been required to support moving windows between screens. In contrast, under shX, if an application has requested and received a font which doesn't exist on a new display, there is no way to ask the application to choose again; shX has to make its best guess at a substitute. Second, richer semantics can be brought to bear on the multiplexing task. For example, in the shX world the pulling-down of a menu is difficult to distinguish from the appearance of a new top-level window. The best the shX proxy can do is to map the new window at the same coordinates on all remote displays; if the window is in fact a menu, this may cause it to appear at an apparently arbitrary location with respect to its parent. Trestle applications avoid this type of problem by presenting a higher level of interface, where the relationship between a menu and its parent is made explicit.

Mbone Applications

Mbone applications such as *wb* are explicit groupware, that is, they can present different views of the same data to different users and accept simultaneous modification from multiple users. Each is implemented as a collection of cooperating processes running at the various participating sites. The processes at each site manage input and output for their local windows and communicate, typically by using IP multicast, with their cohorts on other hosts to maintain a consistent view of the underlying data.

The main task of integrating Mbone applications with Argo was to replace their native group membership mechanism with Argo's. This was fairly simple, because Mbone applications determine conference membership by agreement on an IP multicast address and port. We add a new Mbone application to an Argo conference by selecting a unique multicast address and port and registering it with the Argo conference control server as a part of that conference. Each participating workstation runs an *mbcop* process which watches for registered Mbone applications and spawns them using the registered address and port. In addition, it monitors user activation and deactivation to map and unmap Mbone application windows. This approach allows us to run unmodified Mbone applications as a part of Argo conferences.

A Generic Sharing Agent

In each of our three approaches to sharing, much of the implementation effort involved tracking conference membership, responding to changes, and noticing new relevant windows or applications. Our three application classes each implement this functionality similarly, but not identically.

The Trestle sharing agent is somewhat different from the other two: since it's built into each application, it only needs to track conference membership for that application. In retrospect, building the sharing agent into Trestle was probably a mistake for a number of reasons; for example, the protocol used to communicate with the conference control server is synchronous and thus a Trestle application which hits a breakpoint can disable Argo. We plan to replace the Trestle sharing agent with an external one, like the cops we use for shared X and Mbone applications.

The similarity of the three existing mechanisms suggests that abstracting them into a *generic sharing agent* which can be parameterized by application class would be a useful step. This would not only eliminate duplication in the current code base, but would also simplify the integration of other classes of explicit groupware (e.g., programs whose native group membership mechanism is sending messages to well-known ports). We are currently designing such a generic shared agent and expect to begin an implementation soon.

Telepointers and telepainters

To implement telepointers we used X's shape extension—each pointer is an arrow-shaped window. We use X save-unders so that applications don't have to re-paint when the telepointer obscures and un-obscures various parts of their windows. We have found the performance of this implementation quite acceptable on contemporary workstations; it drops significantly if the X server on one of the displays doesn't support save-unders. Telepainting is also implemented using the shape extension to create windows whose shapes correspond to the ink. Save-unders are less important to telepainting, because the ink continues to obscure window content until explicitly removed.

Although we allow arbitrary placement of shared windows on different workstations, the current version of Argo restricts all copies of a shared window to be the same size. Telepointing and telepainting only make sense when "what you see is what I see;" since we share unmodified applications, the telepainter cannot determine the correspondence between sections of differently-sized windows.

COMPARISONS WITH OTHER WORK

Previous research in this area has included several systems supporting multi-party desktop conferencing. Among the earliest was BBN's Diamond/MMCONF[12, 29], one of the first to employ desktop digital video. Digital video was also used in MERMAID[30]. In both these systems, video quality was constrained by the network bandwidth available, but their use of digital video distinguishes them from other early systems like Rapport[1], CAVECAT[19], and Cruiser[25], which adopted analog video distribution under computer control. Many of these systems also supported collaboration software tools in various forms, ranging from specialized shared-drawing tools to general purpose application-sharing frameworks like the shared-X approach used by Argo. More generally, a number of shared-X systems have been built over the last several years; the same basic approach appears to be gaining popularity in the PC and Macintosh environments as well (for example, Intel's ProShare). In most cases, these systems have been designed to be usable without accompanying audio or video; this does not support the lightweight style of floor control (also available in some of the earlier systems) achieved by direct user discussion.

More recent projects, in various stages of development, include Sun's ShowMe[24], DiCE[8] and ABC[14]. These are intended to be integrated systems, as contrasted with frameworks like TouringMachine[3], ConversationBuilder[15], Suite[10], and Rendezvous[23], which attempt to generalize the lower levels of collaboration software to ease construction of new application-level systems.

Another major research thrust is the remote conferencing work of the Internet Engineering Task Force, which is attempting to establish a consensus on protocols for audio, video, and conference control[7, 27]. The internet currently provides capacities intermediate between narrowband channels like ISDN and the broadband channels available to Argo. As the internet evolves, the two approaches are likely to converge.

In addition to research systems, there are many commercial products available, both conference-room and desktop systems. Conference-room systems have been available for several years from Picturetel, Compression Labs, and others. These have been quite costly and tend to be allocated via advance scheduling, inhibiting quick and easy access. More recently, desktop products have become available, including AT&T's TeleMedia Connection, Compression Labs' Cameo, IBM's Person-to-Person, and Northern Telecom's VISIT.

Both types of commercial products are constrained by the assumption of limited bandwidth (typically a small multiple of 56 or 64kb/s), with corresponding limits on interaction speed and flexibility, especially for video. Both types are also limited in their support of multi-way collaborations, which is especially problematic for desktop systems, where each connected site represents only a single participant. Most products provide first class support for two-site connections, and fall back to a bridging or switching strategy for multi-way conferences. Bridging uses a central video compositor to divide the video image into quadrants, allowing up to four or five sites. Using a more complex switching arrangement, it is possible to accommodate more sites by selectively routing the video image of the current floorholder to all participants, but this inherently eliminates the rich connectivity and visual back-channels that are important in face-to-face meetings. These approaches lack the flexibility of our J–Video which allows the use of multiple video streams configurable by the recipient[4].

A few recent products in the LAN/Workstation market have begun moving toward the kind of flexible approach advocated here. One example is DEC's DECspin system[22]. In its current form, the DECspin product provides only audio and video and no facility for sharing applications, but as these limitations are removed, it will more closely resemble the kind of collaboration environment envisioned by the Argo project.

FUTURE RESEARCH

Our current system uses a capability-based security mechanism, which does not guarantee restricted access to

conferences. The current implementation trusts the conference agent not to disclose conference capabilities to users who do not have access to those conferences. This would be a problem in a more widely distributed conference control service, or in environments which require greater security. Even within a conference, there are privacy issues that we haven't fully investigated; for example, when a user is inactive in a conference, should her image as viewed by the other conference participants vanish, be static, or be live? The EuroPARC experiments[11] suggest that there are advantages to sporadically updated images over static ones; how do these results balance with the privacy concerns?

There are various promising ways to improve our use of audio. As mentioned above, we are investigating echo suppression; the earphones we use are small, but are still physically restrictive. We're also interested in synthesizing stereo soundscapes to match the layout of video windows on individual displays.

One aspect of face-to-face interaction that Argo doesn't address at present is that of sharing views of a printed page (as in a book). Camera resolution is too low for this purpose, so we intend to integrate Lectern, an anti-aliased document viewer for scanned images, into Argo.

Another aspect is the ability to whisper to a neighbor. In Argo, we plan to implement *side conversations,* a technique for redirecting your audio to only go to the person you're whispering to, and for simultaneously reducing the audio level of the other conference participants. To initiate a side conversation, you'd click on a person's video image ("tap her on the shoulder"); we plan to use heraldry to indicate to other conference participants that some subset is involved in a side conversation.

STATUS

We currently have about twenty J–Video-equipped workstations at SRC, and about half a dozen at CRL, our sister lab in Cambridge, MA. Our local workstations are connected via a 100 megabit packet-switched point-to-point network, and include MIPS R3000-based and Alpha/AXP-based DECstations. We regularly use Argo locally during the weekly meetings among the members of our project. Our connection to CRL consists of a leased 1.5 megabit T1 line which is used for all the general traffic between our two sites—netnews, mail, software distribution, etc.—in addition to teleconferencing. Argo is being used for group meetings for a project distributed between the two labs. Whereas our project's use of Argo is a matter of choice—we can just as easily meet face-to-face, and we choose to use Argo largely for testing—the distributed project's alternatives are limited to the telephone and extensive travel. Preliminary reports indicate that Argo has significantly contributed to the success of their collaboration.

We are in the process of deploying Argo to larger user communities. We hope that the installation of additional systems at SRC and CRL will produce more collaborative efforts between our labs. We are also installing Argo at three of Digital's semiconductor engineering labs in Palo Alto, Austin, TX, and Hudson, MA. These labs are engaged in joint design work, and are connected by T1 lines. In addition, we are actively seeking other groups who can help us test our research hypotheses by using Argo. We hope to report on the results of this wider deployment in a future paper.

A videotape demonstrating Argo in use is a part of the Multimedia '94 video program, and a brief video is part of the CD-ROM version of these proceedings.

REFERENCES

1. S. R. Ahuja, J. R. Ensor, and D. N. Horn. The Rapport Multimedia Conferencing System. In Proceedings of the 1988 Conference on Office Information Systems, ACM SIGOIS Bulletin (9, 23).

2. M. Altenhofen, B. Neidecker-Lutz, and P. Tallett. Upgrading a Window System for Tutoring Functions. In Proceedings of the ARGOSI Workshop on Distributed Window Systems, Abingdon, UK, December 1991.

3. M. Arango, et al. Touring Machine: A Software Infrastructure to Support Multimedia Communications. In Proceedings of the Fourth IEEE COMSOC International Workshop Multimedia '92, Monterey, CA, April 1992.

4. Lance Berc, Patrick Chan, and David Redell. J–Video: High-Performance Digital Video on Conventional Workstations. In preparation.

5. A. B. Boghani, E. W. Kimble, and E. E. Spencer. Can Telecommunications Help Solve America's Transportation Problems? A. D. Little Report #65740, Arthur D. Little, Cambridge, MA, February 1991.

6. California Engineering Foundation. Transportation Redefined: Moving People, Goods, and Information. CEF, Sacramento, CA, November 1991.

7. Stephen Casner and Stephen Deering. The First IETF Internet Audiocast. ACM Computer Communication Review, July 1992.

8. M. Chen, T. Barzilay, and H. Vin. Software Architecture of DiCE: A Distributed Collaboration Environment. In Proceedings of the Fourth IEEE COMSOC International Workshop Multimedia '92, Monterey, CA, April 1992.

9. Helen Custer. Inside Windows NT. Microsoft Press, Redmond, WA, 1992.

10. P. Dewan and R. Chowdary. Flexible Interface Coupling in a Collaborative System. In Proceedings of the CHI '91 Conference on Human Factors in Computing Systems, pages 41-48, New Orleans, Louisiana, April 1991.

11. Paul Dourish and Sara Bly. Portholes: Supporting Group Awareness in a Distributed Work Group. In Proceedings of the CHI '92 Conference on Human Factors in Computing Systems, pages 541-547, Monterey, CA, May 1992.

12. H. C. Forsdick. Explorations into Real-time Multimedia Conferencing. In Proceedings of the Second International Symposium on Computer Message Systems. 1985.

13. Christian P. Jacobi. Migrating Widgets. In Proceedings of the 6th Annual X Technical Conference, The X Resource, Issue 1, O'Reilly & Associates, St. Louis, MO, 1991.

14. K. Jeffay, et al. Architecture of the Artifact-Based Collaboration System Matrix. In Proceedings of the Conference on Computer-Supported Cooperative Work, 1992.

15. S. Kaplan, A. M. Carroll, and K. J. MacGregor. Supporting Collaborative Processes with ConversationBuilder. In Proceedings of ACM COCS, 1991.

16. Mike Macedonia and Don Bruzman. MBONE, the Multicast Backbone. IEEE Computer, April 1994.

17. Mark S. Manasse and Greg Nelson. Trestle Window System Tutorial. In Greg Nelson (Ed.), Systems Programming with Modula–3. Prentice Hall, Englewood Cliffs, NJ, 1991.

18. M. Mantei. Capturing the Capture Lab Concepts: A Case Study in the Design of Computer Supported Meeting Environments. In Proceedings of the Conference on Computer Supported Cooperative Work, Portland, Oregon, September 1988.

19. M.Mantei et al. Experiences in the Use of a Media Space. In Proceedings of the CHI '91 Conference on Human Factors in Computing Systems, pages 203-207, New Orleans, Louisiana, April 1991.

20. Greg Nelson (Ed.). Systems Programming with Modula–3. Prentice Hall, Englewood Cliffs, NJ, 1991.

21. Greg Nelson et al. Network Objects. In Proceedings of ACM of the Fourteenth ACM Symposium on Operating Systems, pages 217-230, Asheville, NC, December 1993.

22. L. Palmer, and R. Palmer. DECspin: Networked Multimedia Conferencing for the Desktop. Digital Technical Journal(5,2), 1993.

23. J. F. Patterson et al. "Rendezvous: An Architecture for Synchronous Multi-User Applications". In Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, CA, September 1990.

24. Amy Pearl. System Support for Integrated Desktop Video Conferencing. Technical Report TR-924, Sun Microsystems Laboratories, Mountainview, CA, 1992.

25. R.W. Root. Design of a Multi-Media Vehicle for Social Browsing. In Proceedings of the Conference on Computer-Supported Cooperative Work, pages 25-38, Portland, Oregon, September 1988.

26. Robert Scheifler and James Gettys. X Window System (Third Edition). Digital Press, Burlington, MA, 1992.

27. E. M. Schooler. Case Study: Multimedia Conference Control in a Packet-switched Teleconferencing System. Journal of Internetworking: Research and Experience, Vol 4., No. 2, pp. 99-120, June 1993.

28. M. Stefik et. al. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. Communications of the ACM, January 1987.

29. R. H. Thomas et al. Diamond: A Multimedia Message System Built on a Distributed Architecture. IEEE Computer, December 1985.

30. K. Watanabe et. al. Distributed Multiparty Desktop Conferencing System: MERMAID. In Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, CA, September 1990.

31. Steve Whittaker, Susan E. Brennan, and Herbert H. Clark. Co-ordinating activity: an analysis of interaction in Computer-Supported Co-operative Work. In Proceedings of the CHI '91 Conference on Human Factors in Computing Systems, pages 361-567, New Orleans, Louisiana, April 1991.