Shared Remote Control of a Videoconferencing Application: Motivation, Design, and Implementation

Todd Hodes, Mark Newman, Steven McCanne, Randy Katz, James Landay
Computer Science Division
University of California, Berkeley
{hodes,newman,mccanne,randy,landay}@cs.berkeley.edu

October 5, 1998

Abstract

Most conferencing systems are built to facilitate one of two types of meetings: those in a single room, consisting entirely of colocated participants, or fully distributed meetings, consisting entirely of isolated individuals at different physical locations. Our experiences are of a third style: hybrid meetings consisting of both colocated groups and isolated participants. We illustrate the inadequacy of using existing desktop-based tools in the shared meeting room portion of this hybrid meeting style, and propose adding a software control substrate matched to the specifics of the application to address the inadequacies. As a concrete example from the domain, we describe the design and implementation of an application for manipulating shared video display. Our design employs a user interface split across multiple physical devices paired with a control protocol managing communication between them. The client portion runs on wirelessly-connected portable devices (laptops and 3Com Palm Pilots) and supports per-user input; the server portion handles presentation of shared output on a video monitor. Our design is optimized for meeting room use in three ways: simplified operation to reduce demands on attention, support for remote control, and support for access by multiple simultaneous users.

1 Introduction

Many conferencing, collaboration, and meeting support tools focus either on meetings at a single site, with multiple colocated participants in face-to-face contact [27, 17, 28], or on meetings distributed across multiple sites, where there is exactly one participant per site [6, 4, 15]. In the former (completely colocated) case, sites outfit the participants with new artifacts — electronic whiteboards, computers embedded in tables, shared input devices, etc. — to assist tasks. In the latter (fully distributed users) case, each meeting participant interacts in isolation with software running on a dedicated computer system, and participates in the meeting through the tools.

It is our experience that many types of meetings defy categorization into either of the meeting types described above. We instead experience a *hybrid* environment in which colocated groups of participants meet in conference room, while at the same time other participants join from isolated desktops. Such an environment is illustrated in Figure 1.

The implication of having a hybrid meeting style is that a single collaboration now contains two simultaneous modalities of use; in other words, the needs of colocated groups of participants and individual desktop-bound participants are not the same. A tool that supports only a single usage model (by definition) cannot optimize for both modalities; though it might be amenable to use in both situations, it can be designed for only one or the other. Thus the challenge of the hybrid environment is supporting the two sets of user situations differently. In order to address this need, we espouse a toolkit-based approach utilizing multiple interoperable variants of tools, allowing each variant to be individually optimized for a particular usage style. That the two designs are only variants of the same application[1] provides for both interoperability and liberal code reuse, while at the same time allowing for specific tailoring when appropriate. Leveraging a toolkit framework allows for sharing of the likely significant overlap in the functionality of the variants. This approach can be used for an entire suite of collaborate tools — video tools, audio tools, or as one possible way to implement interaction with a shared electronic whiteboard. This paper specifically focuses on applying it to an existing videoconferencing tool, describing the design and implementation of a shared video display tool for use in the meeting room portion of a hybrid collaborative environment. We adapt a video tool originally designed for desktop use (vic [19]) and built from reusable components in the MASH toolkit [18] to implement the variant.

The rest of this paper is structured as follows. In Section 2, we describe the details of in-room usage and the specific manner in which we adapt the vic tool for use in the room. In Section 3, we detail the components of the new tool itself, called rvic. In Section 4, we describe the use of "standard layouts" in the tool. In Section 5 we detail the remote control protocol. In Section 6, we consider multi-user access and arbitration. In Section 7, we describe allocating and discov-

[1] Note that whether the variants are implemented as a single monolithic (but multi-modal) application or as separate ones is immaterial.

ering rvic servers. In Section 8, we discuss related work. In Section 9, we discuss future and continuing work. Finally, in Section 10, we conclude.

2 Adapting to Conference Room Use

The meeting room in which we performed our experiments is called the Berkeley CoLab (borrowing terminology from Xerox PARC), a room created by faculty, staff, and graduate students in the MASH research group[2] to act as a testbed for evolution of the MASH conferencing tool suite [18]. It supports meetings where a group of local colocated participants collaborate with remote participants via the room equipment; interactions between the participants (both local and remote) are handled by the MASH tools. Thus, it allows for (and practically demands) experimentation with what we have called a "hybrid" conferencing style. The room contains a conference table with chairs, a Xerox LiveBoard, mounted analog video monitors (TVs), video cameras, microphones, audio equipment, and workstations for dedicated processing. The A/V devices (computer and video monitors, cameras, etc.) are connected via an A/V switch controlled by one of the workstations. The intended use of the room is to hold meetings where a group of local colocated participants communicate and collaborate with independent remote participants via the room equipment. Participants have the option to attend meetings either by being physically present in the CoLab or via the Internet multicast backbone (MBone). Remote participants' real-time data is transmitted via RTP [26] and captured by workstations in the room, where it can then be flexibly routed so that video streams are displayed on various video monitors and audio streams mixed and broadcast through the room speakers.

Using the MASH tools with their original user interfaces (i.e., those inherited from their non-toolkit-based predecessors) inside the CoLab illustrates where there is room for improvement in their UIs when leveraged for this particular domain. Because the tools were originally designed for desktop use, the user interface was not well suited for group use in a conference room setting. An experienced user can interact with the tools during a meeting via a monitor, mouse, and keyboard, but when doing so, that user is distracted and thus has difficulty simultaneously participating in the meeting. For example, consider the steps a person has to perform in order to change the display on one of the video monitors to show a particular remote participant:

- Leave the conference table and walk over to the workstation in the corner of the room.
- Deduce which of the four rack-mounted computers display is being routed to the particular video monitor whose display he or she wishes to configure.
- Switch the workstation's monitor to the display of the selected computer. At this point the workstation's monitor will be displaying the same image as the monitor being configured.
- Using the workstation's mouse and keyboard with vic's user interface elements, work the display into the desired layout. This irrelevant activity is broadcast (via the attached monitor) to the room's occupants.
- If the user wishes to configure another monitor, repeat the last three steps.

Obviously, this takes its toll on attention span and can also distract other users. One user can park him or herself at the keyboard and take control of things, but that user then becomes a sort of "technician" and feels divorced from the conversation, a situation that isn't acceptable in certain conferencing situations (i.e., when no dedicated technician is available). To overcome the distance obstacle withot requiring a technician, it must be possible for users to control the applications from where they sit — in other words, remotely. Additionally, two other criteria must be considered: operation of the application must be simple enough to allow users' attention to remain focused on the content of the meeting, and it must support control by multiple simultaneous users. Applying these principles to the vic videoconferencing tool, we developed *rvic* (short for "remote-controlled vic"). rvic splits its user interface across two or more machines, where video display is managed by a server component running on one of the room computers, and user input is managed by a client component running on a portable device with communications ability. We decided to split the application in this way to match its intended use with some underlying technology trends: rapid growth in the number of people carrying personal communication/computing devices (such as PDAs, laptops, and the like), and availability of a ubiquitous computing [30] infrastructure providing local computational capability and device control.

We reiterate that conferencing tool variants such as rvic are not designed to replace the existing tools; instead, they are

[2] http://mash.cs.berkeley.edu/mash

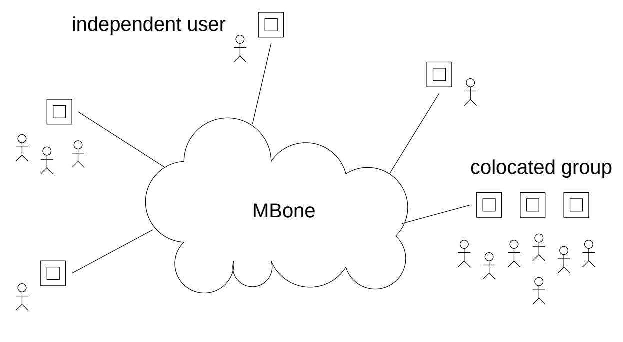


Figure 1: A hybrid meeting style: distributed collaboration between both colocated groups and independent individuals

optimized for a different modality of use. Distributed single users continue to employ the "standard" tools at their desktop, while groups of colocated users could employ variants with "remote-control."

There are alternatives to outfitting rooms with remote controllable software and using mobile clients to access them. One approach is to leverage the existing hardware substrate for remote control of devices: remote controls of the hand-held infrared variety. Why not leverage this substrate for device control, adding similar permanent custom hardware mechanisms for application control where necessary? Often meeting rooms are augmented in just such a way, and this contrasting approach avoids the need for personal devices. We contend that our approach is preferable because providing a general software service can be composable/component-based [12], can be ported to new rooms easily, and is likely to be less expensive, especially if client devices are assumed to be already available. Avoiding custom hardware promotes flexibility, and room/building-use flexibility is paramount in modern reconfigurable (a.k.a. "alternative") office environments [20]. Additionally, leveraging personal devices not only follows naturally from the assumption of their abundance; in certain kinds of meetings, it may be desirable to supply individual participants with personal computing devices to support group work anyway [28].

Another possible solution, perhaps the simplest of all, is to rearrange the room to make the existing interaction devices (mouse, keyboard, monitor) more accessible to meeting participants. This approach is insufficient for two reasons: first, interacting with "unsimplified" applications would tend to absorb the user and harm his or her ability to simultaneously participate in a meeting or conversation; second, only one user at a time could make use of such a setup without custom hardware modifications [17].

3 rvic Components

To implement our videoconferencing tool variant according to the preceding design criteria, rvic to departs from vic in its use of:

- "standard layouts" to group options and thereby reduce complexity of performing common tasks,
- a split client/server UI customized for interaction via a hand-held device and display on a remote monitor,
- an exposed remote control interface supporting multi-user control

The components of rvic include an *rvic server* and one or more *rvic clients*. A single room's rvic servers and clients are managed by a rudimentary *room manager*. (Details on the the room manager are not dealt with in this paper, though the issues are briefly discussed in Section 7.) The rvic server is a variant of vic that displays only a set of video windows rather than the usual UI, instead exposing its configurability through a control protocol. Removing the extraneous UI widgets from the display reduces visual clutter and saves screen real estate. Instantiations of rvic servers can be spawned on standard computers, utilizing only the monitor for video presentation if the monitor is situated appropriately for group viewing. In the Berkeley CoLab, we use scan converters connected through an A/V switch to route video from machines to room TV monitors, but this is simply a presentation optimization.

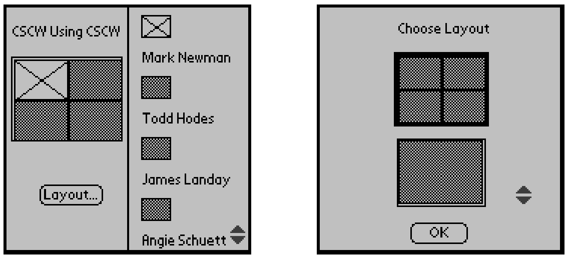
The room manager keeps an inventory of available resources, aggregates state relevant across multiple local instantiations of rvic servers, controls rvic server instantiation, and acts as a rendezvous for control address discovery.

We use the MASH toolkit [18] for the implementation of the rvic server, the laptop client, and the prototype room manager. MASH is a multimedia multicast networking toolkit that leverages a split OTcl/C++ object model to to provide both a low-overhead data-touching layer along with a highly-configurable scripting layer. Implementing rvic required only OTcl scripting of existing split C++/OTcl agents, which simplified the coding chore and validates the use of a script-configured component framework for such tasks.

The rvic clients are responsible for exposing a reconfiguration control UI to the user. We implemented rvic client applications on both the 3Com Palm Pilot and laptop running the MASH platform. The Pilot client communicates via an attached Metricom Ricochet modem, while the laptop client uses a wireless local area network. The clients present a miniature, interactive representation of the monitor screen being controlled and icons to represent each available video source. The user selects a video window on the monitor by selecting the corresponding window on the client interface, as seen in Figure 2(a). When a window is selected, the icon representing the video source currently being displayed in that window is highlighted. The user can switch the selected window's source by choosing a different source's icon from the sources list. The user can also select a different monitor layout by clicking the "Layout..." button and then selecting a different configuration in the form that is displayed, as illustrated in Figure 2(b). The laptop client operates similarly and is illustrated in Figure 3. Because the communication interface between the client and server is clearly defined and based solely on the transmission of text messages (as detailed below), it is straightforward to build alternative clients for other platforms.

4 Standard Layouts

For rvic to be useful in a conference room environment, it must be possible to perform common tasks easily. In order to reduce the cognitive overhead of interacting with rvic during a meeting, we present the user with *standard layouts*. Instead of providing the user with configuration options at the level of individual window geometries, video streams, and switching options (as in desktop vic), "standard layouts"



(a) rvic client layout manipulation

(b) rvic client layout selection

Figure 2: rvic client screenshots from the Pilot. In (a), the current session is "CSCW Using CSCW", and Mark Newman's video stream is displayed in the upper left window of a 2x2 layout. In (b), the currently selected layout is the 2x2 layout.

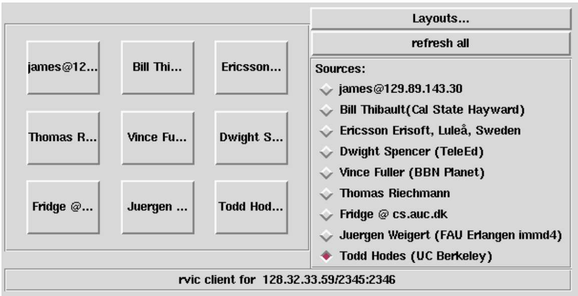


Figure 3: rvic client screenshot from a laptop with the 3x3 layout displayed.

assemble these elements into pre-arranged sets. The user is thus presented with a reduced set of options, and in particular with a choice of different screen arrangements appropriate for different types of meetings. This is intended to allow broad but common changes to be performed easily, and to minimize the time users spend fidgeting with configuration details. As an additional advantage, the use of standard layouts allows for simplified minimal client implementations: the UI need only present a text list of choices or the equivalent.

Not only does the use of standard layouts reduce the number of options with which the user is forced to contend, it also reduces the amount of work necessary to configure the display into a substantially different but similarly desirable configuration. Switching between layouts is accomplished by a single operation in rvic, whereas moving from one of the five supported standard layouts to another would require many operations in standard vic.

The five basic layouts are:

- Single Full Screen: one video window filling up the entire monitor
- Picture in Picture: one video window sized to fill the entire monitor and another smaller (1/16 screen size) superimposed over a corner of the large image
- Focus Plus Context: one 1/4 screen size window in the upper left quadrant of the screen and five 1/16 screen size windows arranged around the edge
- 3 x 3 (aka "Brady Bunch"): 9 equal sized 1/16 screen size windows arranged in a 3 x 3 grid across the monitor
- 2 x 2: 4 equal sized 1/4 screen size windows arranged in a 2 x 2 grid across the monitor

These layouts are illustrated in Figure 4. Each video window within a layout can be independently manipulated, either by explicitly selecting a video source from those in the current session, by cycling through the list of video sources, or by setting automatic switching options.

5 Remote Control Protocol

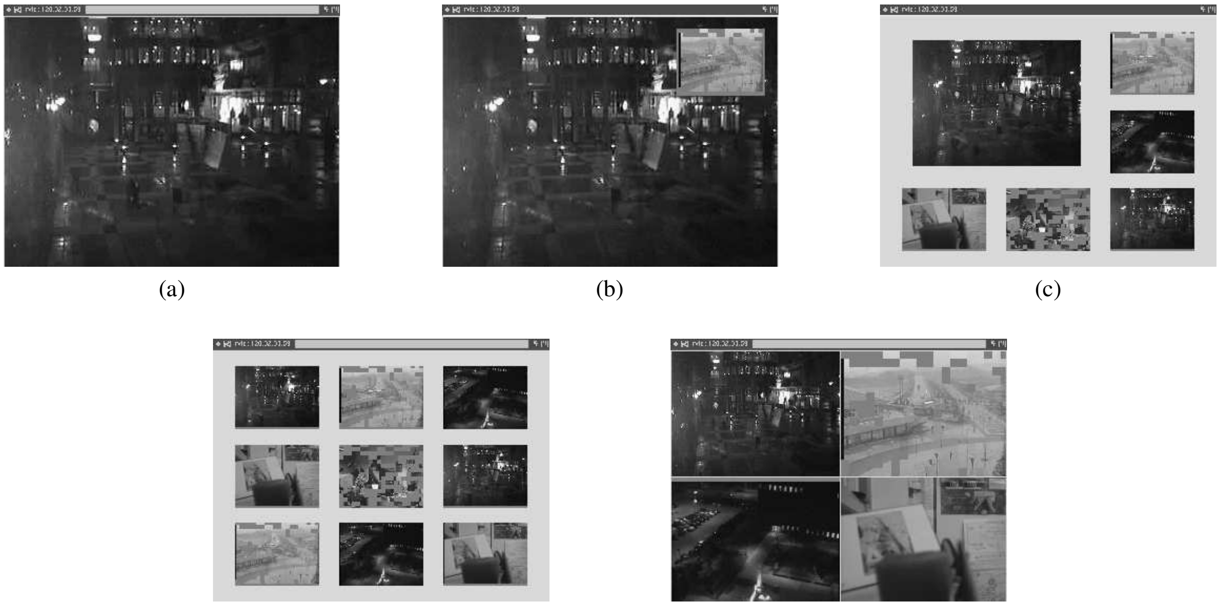
The functionality of the rvic servers is exposed to clients via a simple string-based messaging protocol. The basic format for messages (both request and replies) is:

```
<message type header> <parameters>
<type-specific data>
<type-specific data>
...
```

All messages are idempotent. Multiple messages may be bundled together in a single packet by concatenating them. Each client request is acknowledged via unicast to a list of recent requestors and also propagated via multicast. The current state of a server is periodically broadcast in an announce/listen style. These state updates from the server are not acknowledged. For all messages, sending "Set" commands with blank arguments tells the server to return the current state of those variables. We chose a string-based protocol for rvic instead of other potential choices, such as binary encodings, ASN.1, or RPC/RMI, because string-based protocols have the advantage that they are easily manipulated in scripting languages and can be transfered via application-level text-based transports [25].

The base set of rvic control messages are those for switching between standard layouts and manually switching the source of individual video windows. These messages are summarized in Figure 5. Additional capabilities are provided via a set of "advanced" messages that explicitly manage low-level window options (geometry and switching). The syntax and parameter semantics of these messages are listed in Figure 6. They provide support for:

- a more complete set of configuration options for use by advanced users,
- standard layout authoring, and



(a) (b) (c)

(d) (e)

Figure 4: The five "standard layouts" on the rvic server: (a) single, (b) picture-in-picture, (c) focus plus context, (d) 3 x 3, and (e) 2 x 2. Video images courtesy of the publicly-accessable "Places Around the World" MBone session.

- import and export of new standard layouts without requiring modifications to the rvic server.

The advanced messages also provide a framework on which to build multi-monitor room setups in addition to new single-monitor (i.e., per-rvic) layouts.

5.1 Window Switching

In addition to manual configuration of video windows, we provide control over automatic timer-driven or asynchronous switching similar to that provided by desktop vic. The design exposed a number of trade-offs. Clients can implement much of this functionality (e.g., timer-switching) through simple client-side scripting combined with the basic "Switch VideoWindow N to Source S" commands. But additional mechanism is necessary to allow for complete switching configurability. One possibility is to extend coordination bus[3] messages to client, allowing all the switching logic to remain there. This scheme has the property that asynchronous reactions can be arbitrarily variable, but at a cost of increased client complexity. Another possibility is to allow reactions to specific coordination bus messages to be *registered* with the rvic server. This scheme avoids a round-trip-time latency penalty and communication overhead, but at the cost of limiting the client to the rigid semantics exposed by the registration facility.

In rvic, we use the latter approach, and following our theme of using standard layouts to simplify clients and reduce overhead, we also provide support for standard "layouts" of *multiple-window switching.* Switching functionality includes the ability to:

- specify a list of sources and interval for timer-switching individual windows
- specify which coordination bus messages M cause an individual window to switch to source S when message M/S is received
- specify that the server dynamically reconfigure the layout to match the number of available streams
- specify a cycle of sources for a group of windows to share without duplication and simultaneously switch (a useful way to configure "thumbnail" windows that surround larger separately-switched windows)

Standard layouts are represented internally at both the client and server as named lists of "advanced" messages. In this way, one may think of the basic messages as the interface provided to the user, and the advanced messages as that provided to standard layouts. All rvic servers provide the basic set of layouts described in Section 4, but also may cache a number of "custom" *(name, message-list)* mappings that describe additional layouts. This provides a clean way

[3] The coordination bus, originally called the conference bus, provides many-to-many communication messaging between applications as described in the vic paper [19]. It communciates using tuples; we use the source ID and event type subfields.

Request	Reply
Get MonitorStyleList	MonitorStyleList <styleSpec> <numWindows> ...
Set MonitorStyle <styleSpec>	MonitorStyle <styleSpec>
Get SourceList <session>	SourceList <session> <CName> ...
Set VideoWindow <windowNum> <CName> ...	VideoWindow <windowNum> <CName> ...

Parameter	Description
<styleSpec>	a standard layout, basic or cached
<numWindows>	number of video windows in this style
<windowNum>	integer window number, with windows ordered from top to bottom then left to right, or keyword “all”
<CName>	a RTP [26] CName, or the keyword “next”

Figure 5: Basic messages and message parameters

Request	Reply
Set AutoswitchLayout <on|off>	AutoswitchLayout <on|off>
Set SwitchInterval <windowNum> <time in secs> ...	SwitchInterval <windowNum> <time in secs> ...
Set SwitchList <windowNum> [srcs] [cbMsgs] ...	SwitchList <windowNum> [srcs] [cbMsgs] ...
Set VideoWindowGeom <windowNum> <geom> ...	VideoWindowGeom <windowNum> <geom> ...

Parameter	Description
<on|off>	whether to have the server automatically match the number of video windows to the number of available video streams
<windowNum>	integer window number
[srcs]	list of CNames to switch between
[cbMsgs]	list of conference bus messages such that for each message M, receipt of message M/S should cause a switch to the source S
<geom>	X-windows-style geometry specifying size and placement; specifying “0” deletes the video window.

Figure 6: Advanced messages for automatic switching and specifying layouts

to describe and pass around arbitrary layouts, allowing for the “scripting” of single-monitor and multi-monitor setups. Authoring of such layouts can be performed off-line or via a ‘Save’ command that simply queries the rvic server for its current settings and then writes them to a file on the client.

5.2 Consistency

Having multiple users controlling a single application requires that updates from one client eventually reach all other clients. Changes can be propagated either proactively or lazily where appropriate.

Our expectation is that the rvic protocol will be used in local environments with small to moderate scale usage. Thus, it need not scale to huge numbers of users nor maintain transactional consistency. In fact, by the very (real-time) nature of video, switching between sources should not occur faster than on the order of a few seconds, otherwise the video will be unintelligible. But at the extreme of low-bandwidth connectivity and large client populations, the bandwidth/latency tradeoff must be addressed.

One possibility is to maintain perfect consistency at the expense of reactivity. In this extreme case, similar to the use of database transactions, both the consistency and bandwidth scalability problem can be solved simply by treating the rvic server as a centralized repository and requiring that all client state be updated before committing a change. By requiring acknowledgements from all clients, total bandwidth is limited (by worst-case round-trip time) and there are never client inconsistencies. The trade-off in the use of this particular solution is (likely greatly) increased latency and fragility. Such an approach is far too heavyweight for conferencing applications.

Another approach is to optimize for either bandwidth or consistency at the expense of the other using an eventual consistency model. In the case of latency optimization, updates can either be multicast and/or machine-gun unicasted upon *each* state change. This keeps convergence time low, but increases bandwidth utilization (clearly, the unicast case does not scale). In the case of optimizing for bandwidth utilization (possibly the most realistic example), updates can be rate-limited using timers to batch updates and individual request rates can be limited based on client population estimates (a la SAP [8]). This increases the delay in getting clients’ data consistent, but scales better in situations with highly dynamic updates, very low bandwidths, and/or large populations.

In the current implmentatino of rvic, we provide consistency updates via periodic announcements of the current state from the server. Announcements are both multicast and "machine-gun" unicast to clients that do not support multicast.

6 Multi-user Arbitration

A number of protocols that provide floor control between application instances in collaborative environments have been described in the literature, for example the protocol underlying the qb tool [16] or the CCCP [9]. Collaboration environments that support multiple simultaneous users of single applications (such as is the case when using rvic) inherent the need for participation in standard floor control, but additionally must manage *intra-application* access control. To clarify this difference, we call such intra-application conflict management *multi-user arbitration.* Multi-user arbitration is closely related to floor control. Both address the possibility that when users simultaneously attempt to do something — control the same objects or transmit data on a shared channel — their combined actions can produce conflicting behavior. Thus, both require mechanisms for managing permissions and for transport of control data. These issues have been investigated in terms of a general-purpose, extensible conference control protocol component, and therefore we focus on contrasting multi-user arbitration to floor control as described elsewhere.

Multi-user arbitration is a form of localized, many-to-one "floor control,"[4] and contrasts the global many-to-many nature of the broadcast/multicast moderation in conventional schemes. Thus, the floor control tool manages one or more session-wide "global floors," while a multi-user access tool manages one or more "local floors." The latter is constrained to a subset of conference members, in our case the colocated meeting room participants. Global floor control is insufficient for our hybrid meeting environment because group control of individual rvics must be localized so that each room is controlled separately. Thus, instead of having a single control channel, there are multiple. These cannot be allocated by the session advertiser for two reasons: the number of needed addresses isn't known at session initiation (and certainly not when generating the advertisement) in the light-weight sessions framework, and actual address allocation needs to be locally controlled to take advantage of administrative scoping. These concerns are analogous to those of layered data channel allocation as described in Swan et.al. [29].

Though with rvic there is only a single channel arbitrated, a multi-user arbitration agent should allow for multiple media or control channels to be combined into a single shared "floor." To provide for this flexibly, the multi-user arbitration protocol agent is separated from the rvic server, and a single multi-user arbitration proxy mediates between a set of clients and a 'raw' rvic control address. Conceived of in this way, the multi-user arbitration agent acts along with other agents as components rather than each as portions of monolithic applications, as in the MASH model, and arbitrary control protocols can leverage the framework. This separation is critical in allowing arbitrary groupings of multiple parallel floors without a priori knowledge of what either may be.

Some example possible policies for managing control contention include:

- *"free-for-all:"* no locking or message arbitration
- *moderated:* a single person or group is allowed full access to configure room monitors
- *rotating controller:* anyone may configure the monitors, but short time-scale conflicts (i.e., windows flipping back and forth as two users unknowingly fight) are avoided.
- *moderated rotating controllers:* group locking for moderation in combination with individual short-term locking for conflict avoidance
- *voting:* all participants assert "votes" as to how things should be configured (probably indirectly, based on the state of the client UI, as in SCUBA [1]). The multi-user arbitration agent aggregates the votes and configures controlled agents based on an application-specific combination of them, e.g., using a mean or median.

Additional arbitration policies may be appropriate for different meeting circumstances. Rather than embed the possible policy semantics into the protocol syntax, we instead attempt to map them to a small set of exposed protocol mechanisms and implement only these mechanisms. This separation of policy and mechanism allows for rich scripting of semantics by allowing application-specific mapping to the restricted control message syntax. The current implementation of the multi-user access agent implements a subset of these possible policies through the use of a single primitive — that of setting timed, named locks. Locks are associated with a list of lock holders and timeout for the lock. Long-term locks are maintained to allow for access control. Short-term locks are maintained to avoid conflicts from rotating controllers. The locks provide reliable and consistent control transfer since the information is maintained centrally, a good match to the inherently centralized rvic server. The locks are managed as follows: if multiple locks are set, access attempts are checked against them all. This allows for the moderated rotating controllers policy that uses locks on two time scales. A lock request from a current lock holder is acts as a refresh. Specifying a time interval longer than the current interval updates the lock timer. Use of this model guarantees locks will eventually free, even if lock-holders exit the session or

[4] Unfortunately, the term "floor" is overloaded. We use the term to generically describe the resources being arbitrated, whether access to a group of channels as in floor control or access to an object as in multi-user arbitration.

fail, while additionally allowing for a moderator to maintain control for an unbounded length of time.

Vote collection is not implemented. Difficulties in implementing this strategy include 1) voting only makes sense when vote composites make sense (i.e., it isn't clear how to vote for source to video window mappings beacuse they do not average), and 2) the delay in determining the outcome of voting must be minimized (c.f., SCUBA's analysis [1]) because video switching should be a low-latency operation.

Messages are "authenticated" based only on the sender address. Cryptographic methods must be added to make the mechanism secure.

From our experience with various class or group meetings, we have noted anecdotally that room-level multi-user arbitration may not need to be heavyweight (i.e. strictly moderated) in the common case. Because the simultaneous users are colocated, social conventions tend to prevent conflicts and promote productive use. This may be due to minimization of the "me" versus "them" mentality, perhaps resulting from the lack of (pseudo-)anonymity and distance.

7 Discovering and Allocating an rvic

The more general problem we are investigating is that of architecting mechanisms to launch and control tools in the infrastructure based on location information and user preferences. Such functionality should be accessible via simple extension of the session directory tool [7]. The session directory tool sdr has a facility for statically specifying and dynamically (but manually) choosing between multiple possible applications for a given data type. To use it with rvic, it must be able to spawn a remote rvic server, discover a control address, and spawn a local client application. Additionally, it would need to discover if a remote rvic server is already running so it can work with multiple meeting participants. One could imagine automatic mechanisms that make this process seamless and configurable, similar to the framework in Amir et.al. [2] paired with session directory extensions [29] for control address rendezvous and per-session specification of multi-user control policy.

We implement a prototype version of this functionality via a "helper" application that is spawned when "rvic" is selected as a media tool by a user of sdr. This helper program provides setup via the following steps:

- listen for location beacons [12] on the currently attached subnet; from them, extract the address and port number of the local room manager
- query the manager for the addresses of any locally running rvic servers and/or multi-user arbitration agents.
- spawn new servers if necessary
- spawn the rvic client with the proper address

8 Related Work

The *QuestionBoard (qb)* [16] is an application designed for floor control of collaborative sessions. Though rvic and qb are, in a sense, orthogonal, we might like for the rvic control protocol to participate in the unified floor control algorithm. Similarly, other media or control channels may wish to share the multiple-media unified floor.

The difficulty is that qb assumes there are exactly three media — audio, video, and text questions. Additionally, the control protocol behavior is unnaturally biased to one of the media — the questions — rather than cleanly separated. As described in the section on multi-user arbitration, we would like to generalize to arbitrary intra-session groupings of controlled applications/media and allow for multiple independent "floors." This can be achieved by disassociating the "floor control agent" functionality from the questionboard application and modifying the messages to account for multiple media groups. Disassociating the number and type of media in this way, the floor control agent acts as a component rather than a portion of monolithic qb application. This allows it to naturally extend for use with additional data and control channels, and in particular, the rvic control channel.

The confcntrlr application [21] provides functionality quite similar to our remote control interface for vic, and additionally provides configuration of video transmission options and control of the audio tool vat. The key difference in our approach and that of confcntrlr is that confcntrlr is intended for *remote desktop use* rather than shared remote control of a physically distant local application.

Our initial attempt to make the Berkeley CoLab more useful as a conferencing location focused on adding software-based control to room devices (i.e., for routing displays from one monitor to another) [11]. As illustrated by rvic, though, it must be possible to control the *software interacting with the devices* in addition to controlling the actual devices themselves in order to make the latter useful.

Our implementation leverages a number of ideas from the literature on splitting applications and interfaces across machines. The concept of splitting user interface functionality across multiple physical devices has been explored in several experimental systems [3, 24, 23]. The idea of partitioning application functionality to adjust the computational demands on impoverished devices and to address the challenges of intermittent or poor connectivity has also been described [13, 14]. This is influenced by existing work involving the notion of moving a user interface, e.g., statically via a remote display in X Windows [32], or by dynamically "transporting" displays based on location information [31].

The use of "standard layouts" is related to the notion of "interaction modes" in the ISABEL system [22], except Isabel interaction modes are controlled by a centralized administrator and account for multiple media tool settings.

9 Future and Continuing Work

Though rvic is logically consistent and usable as a stand-alone application, it is actually part of an ongoing effort to adapt all the MASH tools for use in shared meeting rooms such as the Berkeley CoLab. This larger effort attempts to deal with all the room devices and software components in an integrated fashion in order to best express logical consistency and match user expectations. As an example, rvic excludes the details of whether a camera is connected to the particular rvic under control, and the ability to toggle transmission from that camera. This pairing of cameras and video windows is a holdover from the desktop paradigm, and can be made logically separate. Similarly, the ability to remap rvics onto different monitors through the A/V switch is not presented to the user: modifying them *is* possible, but only through separate applications. Thus, though a "monitor" is really a computer running rvic, attached to a scan converter, sent through an A/V switch sending to a particular TV, the details of these mappings are independent of individual rvic servers. By composing and decomposing applications and device interfaces into logical sets regardless of their actual relationships, the hope is to provide a complete system for room interaction that can be matched to users' expectations.

10 Conclusions

As collaboration tools make it easier for distant users to participate in meetings that they could not otherwise attend, meetings that consist of distributed individual participants combined with clusters of colocated users will become more common. Nonetheless, the bulk of research on meeting support focuses on meetings consisting entirely of distributed users or entirely of colocated users. We have described the design and implementation of *rvic*, an example application providing non-intrusive control for colocated meeting participants collaborating with participants at other sites. The main differences between rvic and a desktop video application is our use of:

- standard layouts to group options and thereby reduce complexity of performing common tasks,
- a split client/server UI customized for interaction via a hand-held device and display on a remote monitor,
- an exposed remote control interface with multiple-user arbitration

We leverage the increasing ubiquity of personal mobile computing devices, relieving the need to provide custom hardware, and providing a starting point on which to develop "universal clients" [11], devices that leverage (and make more useful) items available in the environment around a user.

11 Acknowledgements

Special thanks to everyone involved with the MASH project, and many thanks to the members of the BARWAN and GUIR projects. This work was supported by DARPA contract DAAB07-95-C-D154, DARPA contract N66001-96-C-8508, NSF infrastructure grant CDA 94-01156, the California State MICRO Program, Hughes, Metricom, Daimler-Benz, PCSI, and GTE.

References

[1] AMIR, E., MCCANNE, S., AND KATZ, R. Receiver-driven Bandwidth Allocation for Lightweight Sessions. *Proc. ACM Multimedia '97* (November 1997).

[2] AMIR, E., MCCANNE, S., AND KATZ, R. An Active Services Framework and its Application to Real-Time Multimedia Transcoding. *Proceedings of ACM SIGCOMM 98* (1998).

[3] BOLT, R. A. Put-That-There: Voice and Gesture at the Graphics Interface. *ACM SIGGRAPH Computer Graphics 14*, 2 (1980), 262–270.

[4] CURTIS, P., DIXON, M., FREDERICK, R., AND NICHOLS, D. The Jupiter Audio/Video Architecture: Secure Multimedia in Network Places. *Proc. ACM Multimedia '95* (1995), 79–90.

[5] FLOYD, S., JACOBSON, V., LIU, C., MCCANNE, S., AND ZHANG, L. A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing. *ACM SIGCOMM 95* (August 1995), 342–356.

[6] GREENBERG, ROSEMAN, WEBSTER, AND BOHNET. Issues and Experiences Designing and Implementing Two Group Drawing Tools. *Proceedings of the 25th Annual Hawaii International Conference on the System Sciences 4* (January 1992), 139–150.

[7] HANDLEY, M. *Sdr Multicast Session Directory*. software on-line.

[8] HANDLEY, M. *SAP: Session Announcement Protocol, draft-ietf-mmusic-sap-00.ps*. IETF, 1996.

[9] HANDLEY, M., AND WAKEMAN, I. CCCP: Confernece Control Channel Protocol. *Proceedings of SIGCOMM 95* (Aug. 1995).

[10] HEWLETT-PACKARD. Hewlett-Packard JetSend White Paper. http://www.hp.jetsend.com/, 1997.

[11] HODES, T., AND KATZ, R. Composable Ad hoc Location-based Services for Heterogeneous Mobile Clients. *ACM Wireless Networks* (1998). Special issue on Mobile Computing, to appear.

[12] HODES, T., KATZ, R., SERVAN-SCHREIBER, E., AND ROWE, L. Composable Ad hoc Mobile Services for Universal Interaction. *Proceedings of the 3rd ACM International Conference on Mobile Computing and Networking* (1997), 1–12.

[13] JOSEPH, A., DELESPINASSE, A., TAUBER, J., GIFFORD, D., AND KAASHOEK, M. F. Rover: A Toolkit for Mobile Information Access. *Proceedings of the Fifteenth Symposium on Operating System Principles* (December 1995).

[14] LANDAY, J. A., AND KAUFMANN, T. R. User Interface Issues in Mobile Computing. *Proceedings of the Fourth Workshop on Workstation Operating Systems* (Oct. 1993).

[15] MACEDONIA, M., AND BRUTZMAN, D. MBone provides Audio and Video Across the Internet. *IEEE Computer Magazine* (April 1994), 30–36.

[16] MALPANI, R., AND ROWE, L. Floor Control for Large-Scale MBone Seminars. *Proceedings of The Fifth Annual ACM International Multimedia Conference* (Nov. 1997).

[17] MANTEI, M. Observation of Executives Using a Computer Supported Meeting Environment. *International Journal of Decision Support Systems 5* (June 1989), 153–166.

[18] MCCANNE, S., BREWER, E., KATZ, R., ROWE, L., AMIR, E., CHAWATHE, Y., COOPERSMITH, A., MAYER-PATEL, K., RAMAN, S., SCHUETT, A., SIMPSON, D., SWAN, A., TUNG, T.-L., WU, D., AND SMITH, B. Toward a Common Infrastructure for Multimedia-Networking Middleware. *Proc. 7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97)* (May 1997).

[19] MCCANNE, S., AND JACOBSON, V. vic: a Flexible Framework for Packet Video. *Proc. ACM Multimedia '95* (November 1995).

[20] MISCOVICH, P. The Next Generation Office: The Integrated Workplace. *Corporate Real Estate Executive* (Feb. 1996).

[21] PERRY, M. Confcntlr: A Videoconference Controller. Master's thesis, San Francisco State University, 1997.

[22] QUEMADA, J., ET AL. ISABEL: A CSCW Application for the Distribution of Events. *COST 237 Workshop on Multimedia Networks and Systems* (1996).

[23] REKIMOTO, J. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. *Proceedings of UIST '97, ACM Symposium on User Interface Software and Technology* (October 1997), 31–39.

[24] ROOBERTSON, S., WHARTON, C., ASHWORTH, C., AND FRANZKE, M. Dual Device User Interface Design: PDAs and Interactive Television. *Proceedings of CHI '96* (1996), 79–86.

[25] SCHULZRINNE, H. A comprehensive multimedia control architecture for the Internet. *Proc. 7th Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97)* (May 1997).

[26] SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. RTP: A Transport Protocol for Real-Time Applications. *IETF RFC 1889* (January 1996).

[27] STEFIK, M., BOBROW, D., FOSTER, G., LANNINGAND, S., AND TATAR, D. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. *ACM Transactions on Office Information Systems 5*, 2 (1987), 147–167.

[28] STREITZ, N. A., REXROTH, P., AND HOLMER, T. Does "Roomware" Matter? Investigating the Role of Personal and Public Information Devices and their Combination in Meeting Room Collaboration. *Proceedings of the Fifth European Conference on Computer Supported Cooperative Work (E-CSCW '97)* (1997).

[29] SWAN, A., MCCANNE, S., AND ROWE, L. Layered Transmission and Caching for the Multicast Session Directory Service. *Proceedings of ACM Multimedia '98* (1998).

[30] WEISER, M. The Computer for the 21st Century. *Scientific American 265*, 3 (September 1991), 94–104.

[31] WOOD, K. R., RICHARDSON, T., BENNETT, F., HARTER, A., AND HOPPER, A. Global Teleporting with Java: Toward Ubiquitous Personalized Computing. *IEEE Computer 30*, 2 (1997), 53–59.

[32] X CONSORTIUM. *The X Window System.* software online.