Message-Based API for the Dynamic Configuration of Multimedia Conferencing Applications

Henning Schulzrinne
GMD Fokus
Berlin, Germany
hgs@fokus.gmd.de

February 19, 1995

1 Introduction

The control of multimedia conferencing, covering, among others, the distribution of state information, floor control, voting, membership, have been discussed in a number of papers; a range of proposals have been implemented, ranging from highly structured "traditional" multimedia systems such as MMC and JVTOS [1] to "loosely coupled" systems such as the Internet multicast tools (MBONE tools).

Except for the MBONE [2, 3] tools, most of the existing conferencing systems are monolithic: They either consist of a single program or a tightly integrated suite of programs that can only interoperate within the set. The MBONE tools show a different approach, in that different media are handled by distinct *media agents* (`nv`, `ivs` and others for video, `vat`, `nevot` or `maven` for audio, ...). These MBONE tools can be operated manually or combined into multimedia conferences by *conference controllers* such as `sd` or `mmcc` [4–6]. The former implements a participant-driven ("active", implicit) conference, where participants search for interesting conferences in a directory distributed via multicast. `mmcc` presents an example of a initiator-driven ("passive", explicit) conference, where potential participants are personally invited, telephone-style. Other explicit mechanisms include invitation by active mail; WWW and X.500 have also been used as directories. This architecture has been described in more detail in [7].

While the MBONE conference controllers allow the combination of different media agents, these controllers are of the "fire-and-forget" variety. The conference controller simply starts the media agent processes, passing parameters such as multicast address, port number or media encoding as command line arguments. Once a conference has been started, applications are on their own, unaware of other local media agents within the same or other conferences and beyond reach of the conference controller. Within this approach, it would be possible to add additional applications to an ongoing application or kill running applications, but this is not currently implemented in either `sd` or `mmcc`.

For a number of applications, more elaborate control is desirable:

Controller(s) to media agent: Note that a single conference may have multiple controllers. For example, a floor controller may coexist with a membership controller or invitation agent and directory applications. The conference control protocol may be used to change media parameters, for example, to accomodate a new participant or changing network conditions. A floor controller enables and disables individual media applications on the sending side or disables receiving from all but one session member. A radio broadcast controller automatically enables local rebroadcast of hourly news.

Applications within the same conference: Media and other applications within a conference may also communicate. Examples include variations of video-follows-audio or highlighting-follows-audio as well as recording and playback applications. These recorders and playback devices could make use of speaker detection, so that only certain sources are recorded or that video is recorded only when a particular speaker is talking. Other, more farfetched applications, include automatically generating local, unicast or multicast announcements when somebody enters a conference. Another applet that might be handy and easy to implement is a talk timer, possibly sending commands to the floor controller. Another useful example is an SNMP management proxy implemented without modifying the media agent itself.

Applications across conferences: In some cases, auxiliary applications might wish to communicate across session. Recording tools and priority mechanisms come to mind. The latter might automatically lower the volume or display size of a less important conference when activity on another is detected or adjust playback volume depending on the active conference or even speaker.

Many of these control functionalities could be explicitly coded into the media agents. However, many of these are also specialized and may only be useful in very limited circumstances. Rather than bloating already complicated media agents (even assuming that access to source code or a cooperative author exists), these tasks are often ideally suited for implementation by users or local administrators as small scripts in an interpreted language, such as Tcl [8].

The Tcl tool command language and the Unix shell share some of the same philosophy. While Tcl uses the `send` command to implement a type of one-to-one remote procedure call, the Unix shell uses pipes to connect a linear sequence of independent tools. A related approach for constructing digital signal processing simulators was presented in [9]. The approach presented here is geared towards exchanging control information rather than data; it is point-to-multipoint rather than one-to-one and is based on an asynchronous communication model rather than the reply-response model.

2 Local Conference Control Architecture

Based on the descriptions and requirements presented above, we present a local application architecture particularly suited for "composing" multimedia conferencing applications from multiple, independently written components. These components generate and are controlled by event-driven messages that contain no specific destination, but are rather picked up by any other application that cares about that specific class of events.

The architecture is depicted in Fig. 1. We distinguish controllers (abbreviated below as C) and media agents (denoted by M). The former are conference control applications of various sorts, including floor controllers, while examples of the latter are audio engines, video software, shared applications, media recorders, etc. The replicator, denoted by R, simply forwards control messages to all interested parties, without regard for content. In the figure, gray elements have been implemented.

These control messages could be delivered through any Unix IPC mechanism (Unix sockets, TCP connections, RPC, pipes, stdin, ...). Currently, a single message replicator process is used, listening for connections at TCP port 5010. This message replicator could be replaced by a host-only (ttl value zero) multicast address, with the advantage of fewer processes and the disadvantage that every message wakes up every process that subscribes to the local multicast address. The replicator also allows spreading a single conference over several hosts, typically within a local network. This may be useful, for example, if a user employs two workstations, one for audio and applications and another for video. All conferences and all users share a single replicator; there may be several media agents of the same type per conference.

For the purposes of this discussion, we define a *conference* as an entity consisting of several *(media) sessions*. Each session has a fluid membership of zero or more *session members* distributed throughout the

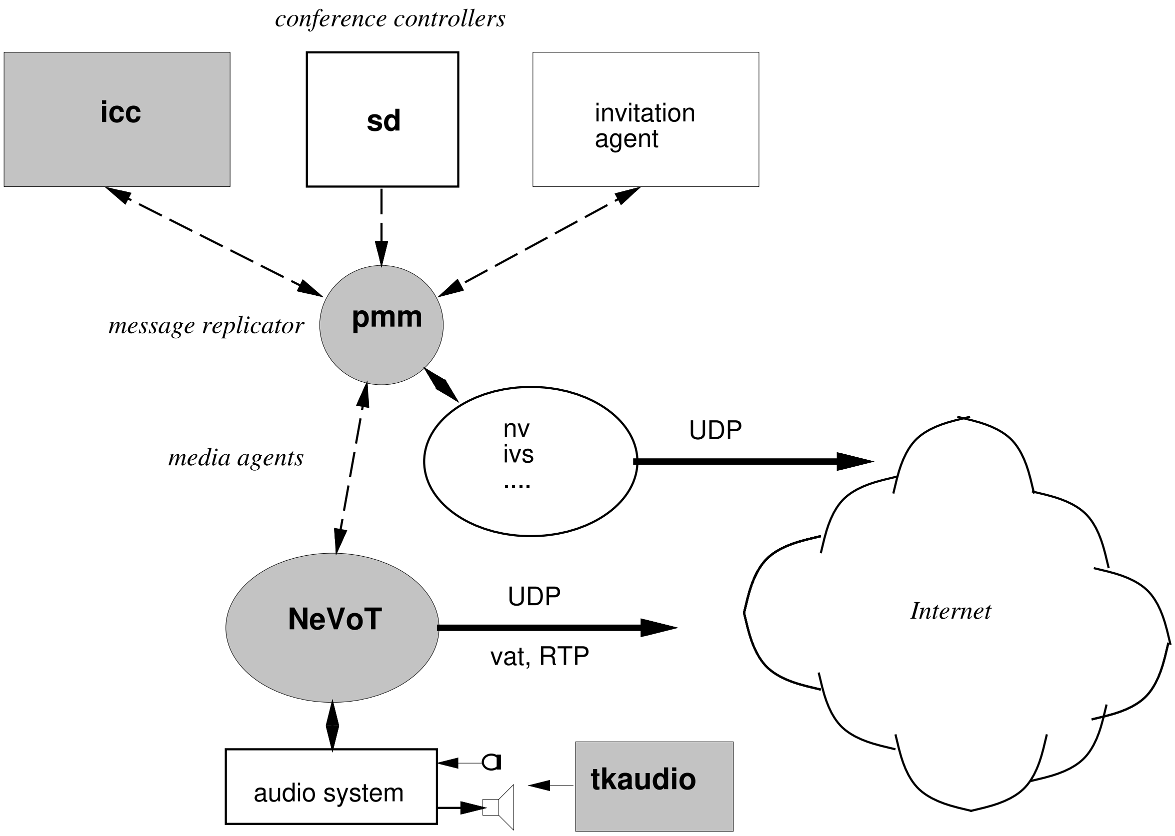

Figure 1: Example multimedia conferencing control architecture

network, typically applications operated by human workstation users, but also robots of various kinds (video-on-demand servers, for example). A session is described by a set of unicast adddresses or, preferably, a single multicast address, any other network addressing information such as port numbers, plus an encryption key, if any. Typical examples of sessions are audio, video, a shared application, a whiteboard, a pointing tool, and so on. Thus, a conference is really nothing more than a list of session tuples and possibly a globally unique name to refer to the aggregate. This definition is motivated by the use of separate IP multicast addresses for each media sesssion. It implicitly allows the case that the intersection of the sets of sesssion members is empty, i.e., that there are conference participants that share no media session. This notion of a conference differs from "traditional" tightly-controlled conferences. It does reflect the reality that it appears difficult to impossible to actually guarantee non-null intersection at all times in the face of equipment and network failures, users disabling reception as well the desire to support subsets of communication facilities for certain users. Indeed, there may be situations where communication is possible although there are members that have no common media session. Sign language and multilingual interpretation are two examples.

Unlike other systems, we allow (and encourage) several media agents for a single user within a single media session. The media session concept also allows several instances of the same type of media within a conference, something typically not thought of. This makes it easy to have multilingual audio, several video channels for, say, speaker and document cameras. However, for a single user, at most one application is currently "responsible" for each media session within a conference, as explained in Section 3.1.

The idea of subscribing to events is also found in [10], albeit for mediating access to shared hypermedia objects within a single system. A file-system event registration service called file-activity monitor (FAM) is offered within the Silicon Graphics Irix operating system. Mark Handley's and Ian Wakeman's work on naming components is also reflected in the message formats [11]. This approach could also be considered as a status-based system, as used for control networks [12]. The architecture is receiver-oriented, with a similar behavior as found in IP multicast: the sender does not know or care who receives its messages, rather, receivers express interest in particular kinds of messages. However, instead of a rigid set of IP multicast addresses, the receiver-directed filter allows a dynamic and fine-grained control over which messages reach a receiver. Receiver-directed filters are also central to the RSVP resource reservation protocol [13].

The system presented here uses constants and information provided by the Real-Time Transport Protocol (RTP) [14] and the audio/video profile of that protocol [15]. In that protocol, each member of a session periodically announces itself by transmitting RTP control (RTCP) packets containing a globally unique name, other "business-card" information and statistics about received sites. However, the architecture does not depend on the use of RTP/RTCP.

3 Message Format

All messages are sent as ASCII text and are interpretable by, for example, a Tcl [8] interpreter[1]. Parameters containing spaces are enclosed in braces. A "binary" encoding, using numeric constants instead of text constants and type-length fields instead of space-separated words, could be used, for a slight gain in processing efficiency and a loss of debugging capability. In the examples, we prefix the message with $C \rightarrow M$ to indicate messages from controller to media agent(s) and with $M \rightarrow C$ for the opposite direction. It should be emphasized that the protocol and architecture itself does not distinguish between those two types of entities, so the distinction serves mainly as an example of typical use.

3.1 Creating a Media Session

First, the conference controller sends a message creating a "blank" media session:

$C \rightarrow M$: `session 17/audio/3`

The identifier "17/audio/3" denotes the conference (17), the media type (audio) and the instance of the media (third audio channel). The controller creates the conference identifier and ensures that it is locally unique; it may be the same as that chosen by the conference initiator.[2] The conference name is arbitrary, case-sensitive, can be of any length, and may contain letters, digits, and punctuation characters except the slash. Slashes were chosen as a hierarchy indicator to align with URL usage [16] and since they are not used within domain names. An URL-like escape mechanism permits the use of other characters. A typical conference name would be the name advertised by the session directory tool `sd`.

The media agent responds:

$M \rightarrow C$: `17/audio/3 created`

The controller then proceeds to establish the parameters of the media session (in any order), shown here for audio:

$C \rightarrow M$: `17/audio/3 ttl 128`
$C \rightarrow M$: `17/audio/3 network UDP`
$C \rightarrow M$: `17/audio/3 transport RTP`
$C \rightarrow M$: `17/audio/3 key topsecret`
$C \rightarrow M$: `17/audio/3 port 3456`

[1]That is the current implementation.
[2]It may be necessary to add a user identifier.

$C \to M$: `17/audio/3 channel 0`
$C \to M$: `17/audio/3 media PCMU 1 8000 0.02`
$C \to M$: `17/audio/3 encrypt 0`
$C \to M$: `17/audio/3 resolve 0`
$C \to M$: `17/audio/3 exclusive 0`
$C \to M$: `17/audio/3 priority net`
$C \to M$: `17/audio/3 cname doe@foo.bar.com`
$C \to M$: `17/audio/3 name {J. Doe}`
$C \to M$: `17/audio/3 email doe@bar.com`
$C \to M$: `17/audio/3 phone {+1 908 555-1212}`
$C \to M$: `17/audio/3 txt {Out to lunch}`
$C \to M$: `17/audio/3 tool {Nevot 3.24}`
$C \to M$: `17/audio/3 loc {New Jersey}`

The RTP [14] source description (SDES) parameters (cname, name, email, phone, txt, loc) determine the information sent periodically for the local site. The ttl (time-to-live) value is only needed for multicast and ignored otherwise. The encryption key *key* is used for secure communication. The channel parameter is used only for the vat protocol. The media encoding names are specified in the RTP profile [15]. The second and third media parameters denote the number of audio channels (1 for mono, 2 for stereo) and the sampling rate. The final media parameter is the packetization interval. For video, only the media sub-command would differ. The `priority` parameter governs half-duplex mode: `none` is full duplex, `net` has network audio suppress locally generated audio, while `local` cuts out the speaker when the local user is talking.

A number of boolean options are defined: `resolve` governs the translation of network numbers to host names, `encrypt` enables encryption of data and control messages and `mute_new` disables receiving data from newly arriving site until explicitly enabled. Any of these parameters except `port` and `cname` can be modified during the session. To achieve idempotency, an application should silently disregard attempts to re-set these two parameters.

After the parameters have been set, the conference controller prods the media agent(s) to establish the actual network connection for this session. For multicast, a single class-D address is provided:

$C \to M$: `17/audio/3 open 224.2.0.1`

The multicast address needs to be selected to avoid collisions, e.g., with Van Jacobson's `sd` session directory or algorithms like [17]. For unicast or one-to-many protocols like ST-II with explicit target lists, a list of target host names (or "dotted-quad" IP addresses) is used instead:

$C \to M$: `17/audio/3 open ursa.fokus.gmd.de 192.35.149.140`

This operation is also used to add additional hosts to an on-going unicast connection. Applications should be prepared to ignore a second request for the same multicast address or existing unicast addresses. Making `open` idempotent allows to add applications to a media session simply by having the conference controller restart the conference, with running media agents ignoring `open` commands.

If the network connection was successfully established, the responsible media agent answers with

$M \to C$: `17/audio/3 opened`

This message might also be used by other applications to create a record for that particular session.

3.2 Removing Sites and Closing the Media Session

Sites can also be removed by the controller:

$C \rightarrow M$: `17/audio/3 close ursa.fokus.gmd.de`

A simple

$C \rightarrow M$: `17/audio/3 close`

closes the whole connection and yields a response from the media agent:

$M \rightarrow C$: `17/audio/3 closed`

This message could also be generated without a prior `close` command after a fatal error. If there was an error, the error message is appended:

$M \rightarrow C$: `17/audio/3 closed {multicast not supported}`

3.3 Controlling Sending and Receiving

During a session, the controller can send messages to the media agents.

$C \rightarrow M$: `17/audio/3 receive` 0|1

disables or enables receiving from that session (speaker muting).

$C \rightarrow M$: `17/audio/3 send` 0|1

disables or enables sending (microphone muting).

$C \rightarrow M$: `17/audio/3/doe@ursa.fokus.gmd.de receive` 0|1

mutes and unmutes an individual source. The local audio device is also toggled on and off by microphone muting.

The media agent responds with:

$C \rightarrow M$: `17/audio/3 receiving` 0|1

or

$C \rightarrow M$: `17/audio/3/doe@ursa.fokus.gmd.de receiving` 0|1

3.4 Statistics

An application may use the command

$C \rightarrow M$: `17/audio/3/doe@ursa.fokus.gmd.de statistics?`

to request statistics for the named site. The media agent replies with

$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de statistics`
`{{sent 1885949} {expected 80} ...}`

All statistical measurements except delay are cumulative and can appear in any order. Times are measured as in NTP, that is, since midnight, January 1, 1900. For content sources, the measurements are as reported by the bridge. The definitions of the measurements follow the RTCP values, where applicable.

name	description
sent	media packets sent (if local site)
expected	media packets expected from source
received	media packets actually received
bad_hdr	packets with bad header
bad_fmt	packets with bad media format
late	late packets
dup	duplicate packets
order	out-of-order packets
lost	packets lost (expected − received)
good	good packets received
samples	usable media samples received
control	control (RTCP) packets received
created	time member appeared
last	time of last data and control packet
delay	playout delay in seconds
media	last media format used

3.5 SDES Information

When audio data or session (RTCP) packets are first received from a source not heard from before, the media agent notifies the controller with a message like

$M \rightarrow C$: `17/audio/3 hello doe@ursa.fokus.gmd.de`

The last parameter is a handle for that new session member. (It is the CNAME value from the RTCP SDES packet.) The handle is to be treated as an opaque string of letters and digits and used to look up information for that new member. As information about the site becomes available, the media agent sends further details, e.g.,

$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de name {John Doe, Bit Recycler, Megacorp}`
$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de loc {Murray Hill, New Jersey}`
$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de email John.Doe@megacorp.com`
$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de txt {can't talk, having lunch}`

Currently, NEVOT sends this information only when it changes. Simply resending the received periodic RTCP messages would allow to attach applications to running conferences, but increase the message load.

3.6 Member Status Reports

The media agent informs the controller by messages when a session member has not been heard from for an extended period of time:

$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de !alive`

When the member returns,

$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de alive`

is sent.

3.7 Activity Indication

$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de active`

When a session member is sending data (e.g., talking for the case of an audio agent), an "active" message is sent, and when it returns to silence, an "!active" message.

3.8 Member Departure

If a session member leaves the session, for example, by issuing an RTCP BYE packet, the controller is notified through a

$M \rightarrow C$: `17/audio/3/doe@ursa.fokus.gmd.de bye {going to lunch}`

message. The last parameter is optional and contains the text in the RTCP BYE message identifying the reason for parting.

3.9 Error Reports

While asynchronicity and multicast simplify the remainder of the system, they tend to complicate error handling. Since commands are multicast, anonymous and asynchronous, there is no explicit success indication. However, there is higher-level handshaking sufficient to track the state of a session. A session exists only between the `opened` and `closed` commands issued by the media agent responsible for a session within a conference.

At any time, the media agent can send asynchronous error messages:

$M \rightarrow C$: `17/audio/3 error {ttl value 256 is illegal}`

System errors are indicated by a message like

$M \rightarrow C$: `17/audio/3 error {no buffers} ENOBUF`

The fourth parameter is the standard Posix error name. Error messages do not imply that the corresponding sesssion is now closed. Rather, if a media handler responsible for a session discovers an error, it issues an `error` command followed by a `closed` command.

Thus, in general, user applications should do as much of the parameter checking as possible locally. Errors should indicate either program bugs or network difficulties, not simple operator errors. If it turns out to be desirable to return error indications only to the source of the command, the message replicator R could prefix a source handle to each message, to be returned with the error indication.

3.10 Other Parameters

While the parameters shown so far are fairly generic, applying equally well to audio, video and shared applications, each media agent will feature some parameters and controls that are specific to either media or implementation. Examples for the NEVOT audio agent include parameters for silence detection, audio packet size, echo suppression, and VU meter.

If possible, global media parameters such as input or output gain[3], the audio source and audio sink should be set by a separate global, application for added flexibility. Figure 1 shows the Tcl/Tk-based tool `tkaudio` developed for that purpose.

[3]However, it may be desirable to be able to adjust gain on a per-session or even per-speaker basis.

4 Signaling Interest

Earlier, we had stated that the system presented here implements a form of selective, application-level multicast. We now present the current, rather simple implementation of this concept.

Applications register or unregister for messages with the replicator by sending a wildcarded string prefixed by + or −, respectively. Currently, these strings follow Unix shell 'globbing' conventions, but could just as easily be treated as full regular expressions. For example, a video media handler might send

$M \rightarrow R$: `+ 17/audio/* *active`

if it wants to track talker activity.

So far, we have glossed over the start-up phase and silently assumed that media agents are running and ready to receive messages. One approach might be to have the message replicator track whether an application has registered for '*/media/* create' messages. Through table lookup, the appropriate application is then invoked. If a media application can only handle a single media stream (e.g., vat or nv), it would then unregister for that creation event.

This scheme relies on reliably tracking which applications are still active. Applications might not bid farewell gracefully but rather crash or be killed with extreme prejudice by a signal. This is also one reason why we currently use stream sockets rather than datagrams, as it avoids having to implement time-out mechanisms. Note that with the TCP implementation, a successful return from a write call does not indicate that the message was actually delivered to the other side, but there will be an error indication eventually. Unfortunately, the RPC model does not fit here since clients will also receive unsolicited messages rather than just responses to commands. However, if available, a reliable datagram approach would be a better fit than a reliable byte stream.

5 Implementation

A preliminary implementation of this concept has been completed. We currently have written a single media agent, NEVOT, responsible for audio. The conference controller `icc` really consists of two pieces: a conference configuration utility presents a panel with parameters and sends the `open` command to the media agent. When the `sd` session directory is used, the parameter entry part is bypassed. The `opened` reply then creates an activity monitoring panel which displays session member status based on the received `active` and `alive` messages. Clicking on a site entry requests statistics from the media agent, which are then displayed numerically and as $y - t$ graphs. A single instance of `icc` can create several conferences, with a separate activity panel for each. The `icc` conference controller is almost exclusively implemented in Tcl/Tk, with Tcl-DP handling message sending and receiving.

A screen dump of the conference controller and media agent are shown in Fig. 2.

Work on integrating recording and video tools is in progress. It mainly requires adding a small interface layer that translates the messages into application-specific actions.

6 Discussion and Open Issues

Currently, the replicator only accepts local connections for security reasons. For single-host applications, Unix sockets can transfer capabilities. As always, true security requires encryption or authentication.

The media agents presented so far are symmetric, that is, the agents running at all participants are functionally the same, even though they might differ in their actual implementation. This is generally true for audio and video agents as well as shared applications that are group-aware, e.g., shared drawing editors or whiteboards. However, for shared-X applications, the actual application is only running at one site,

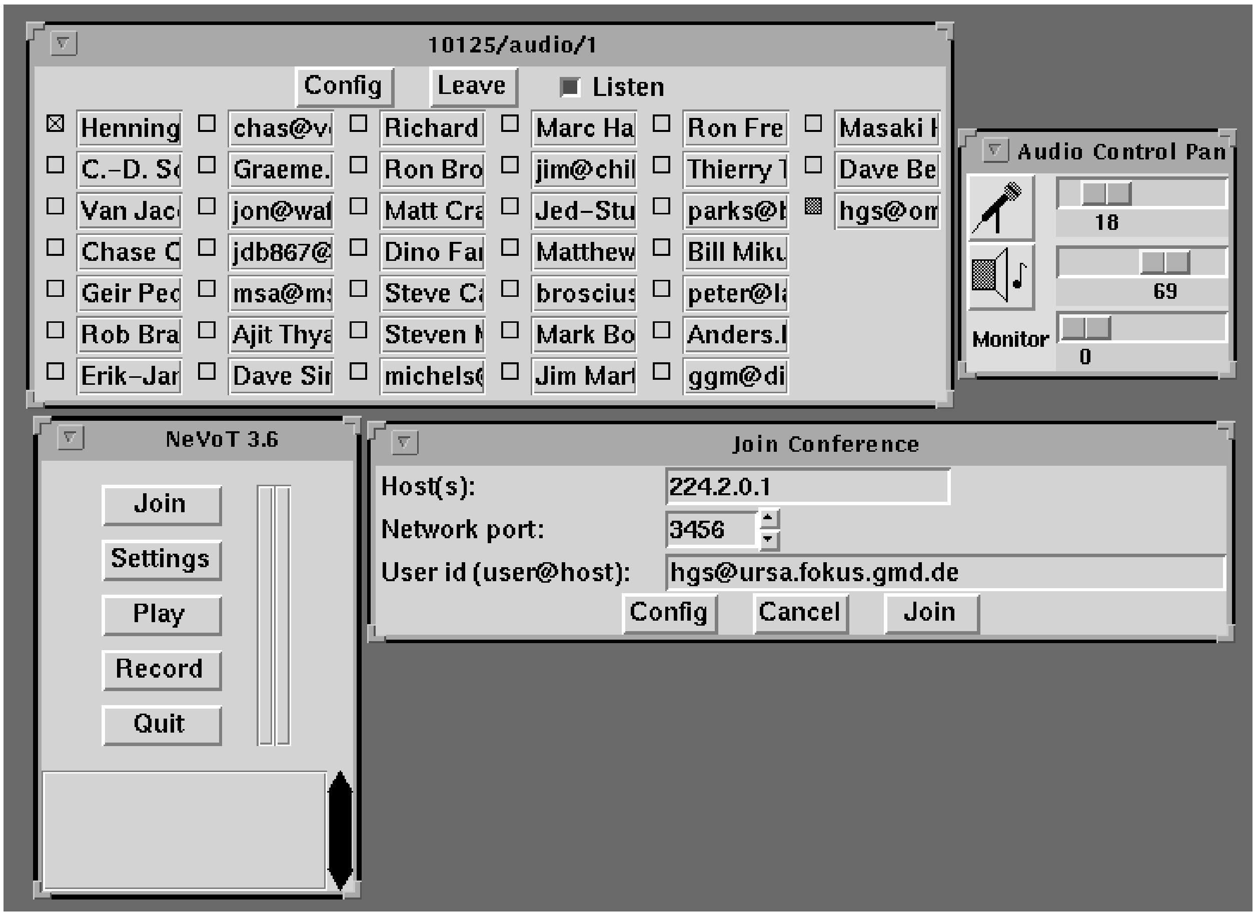

Figure 2: Screen dump of NEVOT, `icc` and `tkaudio`

with remote displays at other sites. It requires further investigation to determine how well the presented inter-application communication model fits for that class of asymmetric applications.

The flexibility of the system described here derives partly from the ability to have several controllers per conference or several media agents per media session. However, this does not imply that any odd combination of controllers and media agents will work. In particular, conflicts may arise when several conference control tools interact within a single conference. For example, if `sd` creates a conference, the activity panel has to be created as well. Excessive message traffic may be generated if several media agents within a single session generate member activity reports. Generally, conflicts should not occur as long as applications are prepared to deal with and ignore two consecutive `alive` messages, say. Applications must be careful about issuing `closed` commands. In the current implementation, for example, the activity panel for a session is destroyed on receiving a `closed` command. If, say, a recorder and the actual playback application exist within the same session and the recorder is stopped, it is likely undesirable to kill other media session applications. Designating, possibly at run-time, an application as a master (issuing `opened` and `closed`) or a slave (destroying itself when seeing `closed`) should be sufficient. It would also be possible to keep a tally of "net" `opened` messages, minus any `closed` messages.

In discussions on conference control it is often silently assumed once a media is brought into the conference, all participants will continue to use it until the medium is formally removed from the conference. Certainly, it should be possible to add media to an existing conference[4]. Current MBONE usage indicates

[4]This is currently not possible with `sd`.

that participants might temporarily drop out of one media session and rejoin later, for any number of reasons. Applications might crash, may need to be restarted for reconfiguration, or the user might want to try out a different instance or version of the same type of media handler. Probably the most common reason is to temporarily reduce network[5] and local CPU load. Facilities to rejoin selected media sessions should thus be provided, preferably without having to restart all applications. (This naturally applies to symmetric applications like audio or video or the remote windows of shared-X applications. Withdrawing a shared application cannot be done transparently.) Note that with the mechanism described here, only the packet replicator is aware of the change of application. Indeed, a user could move an application from one local host to another, without explicit knowledge of the conference controller or any other conference application. This facility might prove useful for local roaming, allowing to move the video display from an office to a lab, say. By making the `open` command idempotent, we can add new applications simply by restarting the conference locally, with the new applications added in.

References

[1] G. Dermler, T. Gutekunst, B. Plattner, E. Ostrowski, F. Ruge, and W. Michael, "Constructing a distributed multimedia joint viewing and tele-operation service for heterogeneous workstation enviroments." internal memorandum, 1994.

[2] S. Casner and S. Deering, "First IETF Internet audiocast," *ACM Computer Communication Review*, vol. 22, pp. 92–97, July 1992.

[3] H. Eriksson, "MBone – the multicast backbone," in *Proceedings of the International Networking Conference (INET)*, (San Francisco, California), pp. CCC–1 – CCC–5, Internet Society, Aug. 1993.

[4] E. M. Schooler, "A distributed architecture for multimedia conference control," Tech. Rep. ISI/RR-91-289, University of Southern California/Information Sciences Institute, Nov. 1991.

[5] E. M. Schooler, "The connection control protocol: Specification (version 1.1)," technical report, USC/Information Sciences Institute, Marina del Ray, California, Jan. 1992.

[6] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing system," *Journal of Internetworking: Research and Experience*, vol. 4, pp. 99–120, June 1993. ISI reprint series ISI/RS-93-359.

[7] E. Schooler and S. L. Casner, "An architecture for multimedia connection management," *ACM Computer Communication Review*, vol. 22, pp. 73–74, Mar. 1992.

[8] J. K. Ousterhout, *Tcl and the Tk Toolkit*. Reading, Massachusetts: Addison-Wesley, 1994.

[9] H. Schulz-Rinne, "The DSP workbench: Modeling parallel architectures as concurrent processes," in *1986 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Tokyo, Japan), pp. 54.9.1 – 54.9.4, IEEE, Apr. 1986.

[10] U. K. Wiil, "Using events as support for data sharing in collaborative work," in *International Workshop on CSCW*, (Berlin, Germany), pp. 162–176, Institute of Informatics and Computing Technique, Germany, Apr. 1991.

[11] M. Handley and I. Wakeman, "CCCP: conference control channel protocol – a scalable base for building conference control applications." V1.4, Mar. 1994.

[5] if multicast pruning is implemented

[12] R. S. Raji, “Smart networks for control,” *IEEE Spectrum*, vol. 31, pp. 49–55, June 1994.

[13] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, “Rsvp: a new resource ReSerVation protocol,” *IEEE Network*, vol. 7, pp. 8–18, Sept. 1993.

[14] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications.” Internet draft (work-in-progress) *draft-ietf-avt-rtp-*.txt*, Nov. 1994.

[15] H. Schulzrinne, “Sample profile for the use of RTP for audio and video conferences with minimal control.” Internet draft (work-in-progress) *draft-ietf-avt-profile-*.txt*, Sept. 1993.

[16] T. Berners-Lee, L. Masinter, and M. McCahill, “Uniform resource locators (url),” Internet Draft, URI Working Group, Sept. 1994. Work in progress.

[17] A. Eleftheriadis, S. Pejhan, and D. Anastassiou, “Architecture and algorithms of the Xphone multimedia communication system,” *Multimedia Systems Journal*, pp. –, 1994.