Dynamic Configuration of Conferencing Applications using Pattern-Matching Multicast

Henning Schulzrinne*
GMD Fokus, Berlin
schulzrinne@fokus.gmd.de

Abstract

Multimedia conferencing systems are usually large, complex software systems. We describe a local control architecture and communication protocols called pattern-matching multicast (PMM) that allow media agents, controllers and auxiliary applications such as media recorders and management proxies to be tied together into a single conference application. Unlike other systems, control of a single conference can be shared between several controllers. Each medium can be handled by one or more independent media agents. Parts of the system have been implemented using an IP-multicast-based audio conferencing tool (NeVoT). The communicating applications disseminate state and control information through a distributor. The distributor mainly limits distribution of messages based on expressed interest of other applications, thus implementing an application-level, receiver-driven local multicast. It also automatically starts applications as needed. The same inter-application protocol was also implemented using IP multicast restricted to the local host and can be based on other inter-process messaging services such as ToolTalk.

1 Introduction

The problem domain of multimedia conferencing can be roughly divided into three separate areas: the transport of data, the management of conferences and the local control and marshalling of components and resources. While there are numerous proposals for the first two areas, relatively little has been said about the third. Here, we define components as the individual applications that are brought into a conference by participants as well as those applications that are responsible for controlling these applications and the overall conference. There are probably two reasons of why this particular issue has received relatively little attention in the past. First, many conferencing applications are monolithic, often written by a single group. They consist of a single application that at most spawns some helper applications. Second, each of these conferencing applications has its own set of protocols for data and control, so that there was little opportunity to replace, say, one video tool by another.

The set of conferencing tools built by a diverse set of research groups for use on the Internet multicast backbone (MBONE) [1, 2] departs from this approach. Here, each medium is typically handled by a distinct *media agent*, with control delegated to an external conference controller [3]. This separation allows to evolve individual media agents, local control, diversity and reuse of media agents. For example, the same video application might be used both within seminar-style, loosely-controlled conferences initiated from a

*This work was supported in part by DeTeBerkom as part of the MMTnG project.

multicast session listing service or centralized directories as well as a telephone-style, tightly-controlled conference based on invitations [4, 5]. Examples of a multicast directory include sd [6]), while WWW is becoming popular for centralized session rendezvous [7, 8, 9, 10]. Our approach presented here supports the goal of composing multimedia conferencing applications and, more generally, synchronous CSCW (computer supported cooperative work) tools, out of individual applications, both in the MBONE environment and elsewhere. For conciseness, the architecture and protocols will be refered to as PMM (pattern-matching multicast).

The paper is structured as follows. In Section 2, the goals and requirements for interactions between applications handling media and various controlling applications are introduced. Section 3 describes the architecture of a conferencing system based on interacting applications, while the details of the protocols are discussed in Section 4. Details about the implementation can be found in Section 5. The paper concludes in Section 6 with a summary of some of the open issues and mentions extensions to other applications.

2 Goals and Requirements

PMM allows for continuous communication and control message exchange between media agents and controllers, as opposed to existing MBONE conference controllers that are of the "fire-and-forget" variety. The MBONE conference controllers simply start the media agent processes, passing parameters such as multicast address, encryption key or media encoding as command line arguments. Once a conference has been started, media agents are on their own, unaware of other local media agents within the same or other conferences and beyond reach of the conference controller. While it would be possible to add applications to an ongoing conference, this is not currently implemented in the MBONE conference controllers mentioned.

For a number of applications, more elaborate control is desirable. We distinguish local controllers and global controllers. A local controller controls only the applications of a single user, without communicating with controllers of other users. For example, a radio broadcast controller might automatically enable the audio media agent to tune in to the news on the hour. Global controllers are in contact with their peers run by other conference participants. Some types of global controllers might only be involved in the start-up phase of the conference, e.g., for inviting users, or they may stay involved with the conference, for example, to negotiate common media encodings as new members join the conference. Other global controllers might come into play once the conference is in progress. A common example of the latter is a floor controller that limits access to media or shared applications. A floor controller enforces the floor policy by muting either media sources or disabling reception of data from all but the floor holder.

Controllers and media agents communicate in a number of different ways:

Controller(s) to media agent: A single conference may have multiple controllers, i.e., media agents may receive commands from multiple sources, such as an invitation agent and a floor controller. Controllers can be implemented independently, but the floor controller in our example needs to know when conferences start and stop or members join and leave.

It should also be possible to have both per-medium activity indication, as is used by most of the existing MBONE tools, or a per-session joint activity indication across all media. A single activity display saves screen real-estate and allows one to determine with a glance how a conferee is participating in a conference. On the other hand, it is often more natural to closely associate participant information with the object controlled by that participant, for example, by labeling a pointer on shared workspace or having a row of labeled video thumbnails. Using the architecture described here, it is relatively easy to use both approaches at the same time, simply by adding another controller that just tracks membership and activity information.

Media agents within the same conference: Media agents within a conference may also want to communicate locally with each other. Examples include variations of video-follows-audio or highlighting-follows-audio as well as recording and playback applications. 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.

Applications across conferences: In some cases, auxiliary applications might wish to communicate across sessions. For example, a priority mechanism might automatically lower the volume or reduce the video image size of a less important conference when activity in another is detected.

All of these interactions are between applications run by a single user. Communications between users is considered the realm of a conference control protocol. While it is possible to have a single protocol for both local, inter-application and inter-user interactions [11, 12], we believe that requirements and functionality are sufficiently different to merit a split. We will briefly discuss the arguments in favor of splitting from the perspective of inter-application interaction at a single conferee. For one, a major goal of this work is to make the media agents oblivious to the conference style and the conference control protocol used to implement that style. Indeed, there may not be a conference control protocol at all, rather only manual configuration or a conference directory. Furthermore, much of the information exchanged between applications, e.g., about the activity of individual users, is only of local interest. It should be noted that for a conference setup protocol, different media are likely best handled together to allow an invitee to accept or reject an invitation based on the overall requirements of the conference. For an inter-application protocol, ease of parsing suggests separating media control messages, as is done here.

Also, restricting the functional range of the protocol to inter-application interaction for a single user allows a number of significant simplifications. Since the number of applications per conference and user is small, we can use a distributor process (see Section 3) with TCP connections, RPC or multicast. As long as we restrict ourselves to a single host, no reliable multicast protocol or periodic state updates are needed. Even if network-wide IP multicast is not available (e.g., when using most current wide-area ATM networks), it is likely available locally. There is no need to adapt to widely varying available network bandwidths; security can rely on file-system based mechanisms if deemed necessary. Naming and rendezvous (e.g., agreeing on a common port) are far easier when one can rely on a shared file system. Also, by separating the inter-application protocol from the inter-user conference control protocol, the inter-application protocol can be implemented in a manner appropriate to a computing platform's interprocess communication facilities, without affecting the ability to enter into conferences with applications on other platforms.

Overall, a simple, extensible protocol for a well-defined application seems preferable and more in the Internet protocol spirit than trying to design a general conference control protocol. Despite these arguments for separating inter-application interactions from conference control, many of the concepts and parameters may well carry over into the design of a conference control protocol. It may also turn out that the messages used in the invitation protocol [5] can be directly used for controlling applications, albeit with more parsing effort. However, using IP multicast for conference control poses a bootstrapping problem. All participants of a conference would somehow have to find out which multicast address and port to subscribe to first, requiring some form of global directory.

Most of the interactions between controllers and media agents mentioned above could be explicitly coded into the applications. However, these functions also tend to be 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 [13] or Perl [14].

The Tcl tool command language and the Unix shell share the same philosophy of composing larger applications from communicating components. While Tcl uses the `send` command to implement a 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 [15]. The approach presented here is geared towards exchanging control information rather than data; it is point-to-multipoint rather than two-party and is based, for the most part, on an asynchronous communication model rather than the reply-response model. In all these properties, it differs from standard RPC. ([16, 17] discuss asynchronous RPC.).

3 Local Conference Control Architecture

3.1 Conferences, Sessions and Members

Since the terminology for multimedia conferencing seems to have several dialects, it is useful to define a few key terms. We view a *conference* as consisting of several *sessions*, each with their own set of *members* (conferees or participants). Each session typically consists of one media type, although bundling of, say, MPEG audio and video into a single session is possible. Each session has its own network association (say, multicast address and transport address) and can thus be handled by a separate application, although a single application or *media agent* may well be responsible for several sessions. We define media agents as "a software entity that handles media-specific functions such as encoding, compression and transport packetization that are used by conferences. Media in a conference might include audio, video, graphics and text" [18]. A conference may well have several distinct sessions of the same media type, say, two audio sessions with different languages or distinct video sessions for "talking heads" and document cameras. If confusion seems likely, the term *local conference* refers to the parts of the conference, such as controllers and media agents, controlled and used by a single conferee within the network-wide conference. Naturally, our main focus is on a single conferee rather than the conference as a whole.

This definition makes conferences fairly loose "federations", where it is theoretically possible for the intersection of the membership sets of the sessions to be empty. It is up to the conference control protocol to assure that conference participants can communicate. This definition reflects the properties of MBONE conferences, but can be applied to other settings as well. It differs in layering, but not functionality, from common notions where a conference is made up of members, each of whom participates in a set of media session. Hierarchical conference models are discussed in [12] and can be implemented by making the conference naming scheme more elaborate than the one described in Section 4.1.

3.2 Composing Conferencing Tools

Based on the descriptions and requirements presented in the introduction, we now 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 generic architecture is depicted in Fig. 1, using the components discussed already. Conference controllers are shown as rectangles and media agents as circles. The media agents communicate directly with their peers at other Internet sites, by either unicast or network multicast, with the media flows shown as thick,

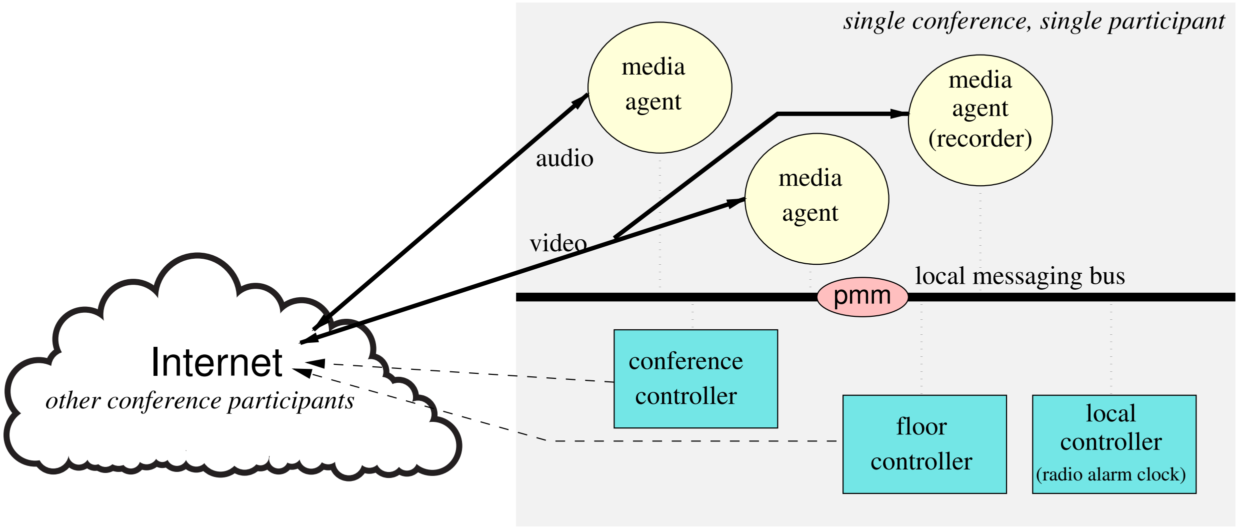

Figure 1: Generic multimedia conferencing control architecture

solid arrows. Note that a single media stream may reach more than one media agent for a single user, e.g., a video tool and a "VCR"[1]. Global conference controllers also communicate with their peers across the network.

The local logical messaging bus connects these controllers and media agents, which communicate via control messages, shown dotted. Unlike, say, a bus-based LAN, messages on this logical bus are not addressed to a single destination but rather a (possibly empty) group of processes. The sender does not know or care which process acts on the message, but may require confirmation that some process has acted on it before proceeding. Details of the protocol will be discussed in Section 4.

3.3 IPC for Message Exchange

These control messages could be delivered through any Unix IPC mechanism (Unix sockets, TCP connections, UDP messages, RPC, pipes, file system, ...) or specialized in-memory multicast support [19]. Three IPC methods have been investigated. In the first, a per-user or per-host message distributor process called pmm (pattern matching multicastor) is used, listening for connections at a well-known TCP port. Here, TCP is not needed for its reliability or flow control features, but rather to simplify detection of a process that has ceased to exist. However, the connection timeout interval can cause difficulties when restarting applications. The second approach also uses a distributor process, but communicates using UDP datagrams. In both cases, media agents and controllers connect to this distributor process when they start up. As described in more detail in Section 4.2, the distributor process can be told by each process connected to it which messages that process wants to receive. By default, a process receives no messages.

In the third approach to inter-application communication, no distributor process is needed; instead, applications send UDP datagrams to an IP multicast address [20]. The reach of the multicast packets is restricted to the local host by setting their time-to-live value to zero[2]. Local multicast offers the advantage of dispensing

[1] It is one of the advantages of receiver-based IP multicast that such arrangements require no cooperation or awareness on part of the sender. However, the application has to allow address re-use.

[2] To tie together applications on different hosts, a larger time-to-live value is needed.

with the distributor process and avoiding maintaining filtering and connection information in the distributor. It has the disadvantage that every message wakes up every process that subscribes to the local multicast address, even though each process may only care about a small subset of the messages. It is generally not possible to set up a set of multicast addresses or ports with the same filtering effect as the content-based filtering in the distributor since the interests of different listeners will have non-zero intersection. The relative run-time costs of distributor vs. multicast will largely depend on the number of processes attached to the distributor or listening to the local multicast address.

A TCP- or RPC-based message distributor has the advantage that it continues to work, if inefficiently, even if media agents are distributed over several hosts, while multicast and UDP-based distributors require additional reliability mechanisms to deal with packet loss. Since data rates are modest and losses are low in a local environment, a relatively simple reliable multicast protocol may be sufficient.

Standard remote procedure calls, without intervention of an intermediary, could not provide the same functionality. For example, neither media agents nor conference controllers are strictly clients or servers, as either can initiate requests to the other. Also, each application would have to track which other components might be interested in particular events. RPC could serve as a communication mechanism between components and the distributor, but the additional overhead does not seem to be balanced by any great advantages.

The reason for choosing a network-based IPC rather than a Unix-domain socket or pipe is the ability to have several clients connect to the distributor server simultaneously. Also, network-based IPC can be extended beyond a single host, allowing a set of media agents to spread over several workstations, e.g., to harness additional processing power or special hardware resources. Most importantly, only network-based IPC is likely to be available across operating systems.

Recently, there has been general interest in having applications cooperate through extensions of the RPC model. In particular, the ToolTalk service "enables independent applications to communicate with each other without having direct knowledge of each other. Applications create and send ToolTalk messages to communicate with each other. The ToolTalk service receives these messages, determines the recipients, and then delivers the messages to the appropriate applications." [21] Similar ideas can be implemented within a distributed object environment such as CORBA [22]. As discussed in Section 5, our messaging protocol can easily move to these environments as well when they are generally available.

The various approaches to implementing a messaging bus are summarized in Fig. 2; implementation aspects are discussed in Section 5.

PMM uses constants and information provided by the Real-Time Transport Protocol (RTP) [23] and the audio/video profile of that protocol [24]. In RTP, each member of a session periodically announces itself by transmitting RTP control (RTCP) packets containing a globally unique *canonical name*, other "business-card" information and statistics describing media data sent and received.

Note that the architecture does not depend on the use of RTP/RTCP or the particular form of the canonical name[3].

4 Protocol Operation

4.1 Message Format and Naming

In the following, we describe the basic operation of the "protocol" between local conference components. All messages are sent as ASCII text and are formatted to be directly interpretable by a standard Tcl [13] interpreter.

[3]This is also the reason for not having controllers listening directly to the RTCP multicast address.

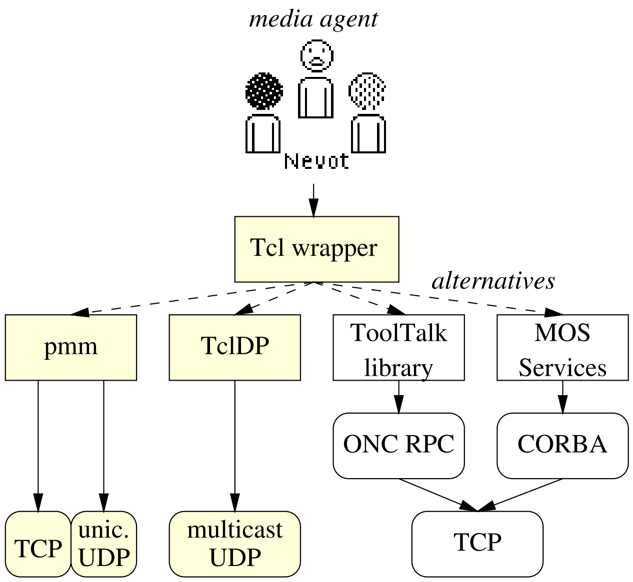

Figure 2: Alternatives for implementing local message distribution

Messages name the object, followed by the operation to be applied to that object, followed by any parameters needed. A sample message might be "`NASA%20Shuttle/audio/1 open 224.2.01`". The message components will be explained in more detail below.

Objects can be sessions and session members. Each media session is named by a hierarchical descriptor containing identifiers for the conference, media and media instance, e.g., the name "`NASA%20Shuttle/audio/3`" denotes the audio session 3 within the NASA shuttle launch conference. Media sessions are named by their creator, typically a conference controller. The conference name only has to be locally unique, but typically, that name will be the same as a global conference identifier advertised by, say, a session directory[4]. Media type names are drawn from the list of Internet media types [26].

The naming of media instances is a local matter, with the application creating it responsible for naming[5]. The media instances do not have to be numbered sequentially; a random identifier might simplify creation of sessions by several different controllers. While there often is a one-to-one mapping from media instances to media agents, this is by no means required. A single application may handle several sessions concurrently, for one or more media types, but this is invisible to the protocol.

Members of a session are named by appending their RTP canonical name to the session name. The RTP canonical name is formatted like an Internet email address: a user name, the at sign, and the name of the Internet host from which the member participates in the conference. Each conferee has exactly one such name, which is globally unique, constant across media sessions, permanent while she participates in the conference, but dependant on the host the conferee is logged in at[6]. *C/audio/3/hgs@ursa.fokus.gmd.de* is an example of such a fully qualified member name. Other naming schemes with the same properties can be used. In subsequent descriptions of messages, we will denote by *sessionname* the three-part session name and by *membername* the four-part member name.

[4] We chose the URL [25] escaping convention for white space characters.

[5] Conference control protocols may not name media instances at all.

[6] Some manual configuration may be required if a user participates from several workstations.

Note that without a central conference manager, naming is the only mechanism to tie together members and sessions into conferences.

4.2 Setting up Message Filtering

Applications register or unregister for messages with the distributor by sending a message pattern prefixed by + or −, respectively, to the distributor. Messages with these prefixes are the only messages interpreted by the distributor. Currently, the pattern follow Unix shell wildcard ('globbing') conventions, but could just as easily be treated as full regular expressions. For example, a video media handler might send

```
+ C/audio/* *active
```

to the distributor if it wants to receive messages about the active/non-active status of the members of all audio sessions within conference C. This particular pattern might be used by an application interested in tracking talk durations. Note that any part of the message can be used for filtering. Messages with the + and − prefixes are ignored by applications to allow use of multicast instead of a message distributor.

4.3 The Control Protocol

Central to PMM is the requirement that messages are idempotent, so that a message can be repeated any number of times without harm. This allows helper applications that are added later to an ongoing session to be initialized (see Section 4.6) without confusing already active media agents. Applications silently ignore messages when they do not implement a particular operation.

Sessions can be in one of three basic states: *closed*, *created*, or *opened*. In the *created* state, a session exists and has a name, but has no network connectivity. Before multimedia data can flow, the session has to go to the *opened* state. The state transitions are shown in Fig. 3. Sessions can only have members while in the *opened* state; almost all session parameters can be set and changed in either the *created* or *opened* state, with some restrictions explained below. Beyond their basic state, sessions are described by two additional state variables. Sessions can be in the sending or muted state, depending on whether the local application is allowed to generate media data or not. The sending state is only meaningful if the media agent has also been *opened*. The second session state variable can assume the values "receiving" and "not receiving". A session member has one state variable: it can be in the receiving or not receiving (muted) state depending on whether incoming media data is rendered or not. Note that these session and member states are independent of the media type. Session states in PMM are strictly a local matter; coordination of states between conferees is the role of a conference control protocol.

PMM distinguishes two classes of messages: *requests* and *notifications*. Requests are answered by responses, notifications are only answered if an error occurs in one of the recipients. Requests are used for state changes and for queries about the current system state. Confirming requests ensures that controllers and media agents agree on the current state. All parameter changes and announcements of new session members are sent as notifications, with the expectation of success (see Section 4.4).

Having covered the general structure, we now describe the protocol interactions in more detail. Message parts shown in *italics* are variables; literals are printed `monospaced`.

To create and name a session, a controller sends a message "`session` *sessionname*", and waits for the `created` reply from a media agent before proceeding. Only one media agent should answer to avoid synchronization problems. Media agents already handling this session should not reply to the request to facilitate reliable creation of several media agents per session.

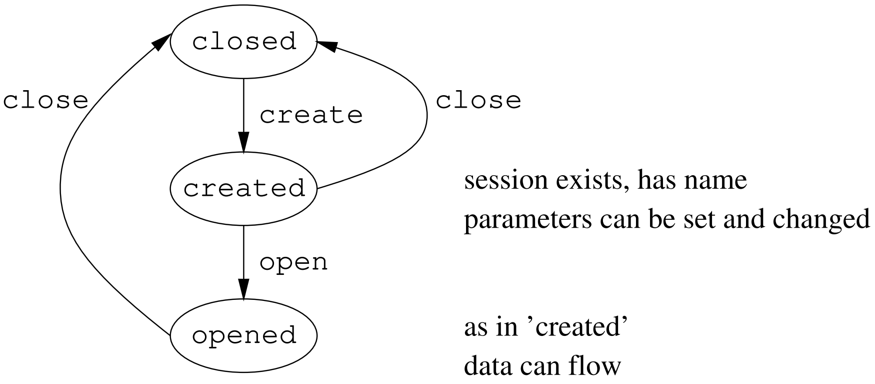

Figure 3: Transitions between basic session states

The controller then proceeds to establish the parameters of the media session (in any order). A sequence of messages for an audio conference might be:

sessionname `ttl 128`
...
sessionname `transport RTP`
sessionname `media PCMU 1 8000 0.02`
sessionname `cname hgs@ursa.fokus.gmd.de`

The example sets up a multicast time-to-live value of 128 and RTP as the media transport protocol. The sending format for audio is set to be PCMU (μ-law PCM), with one channel, a sampling rate of 8000 Hz and a packetization interval of 0.02 seconds. The canonical name of the local sender, i.e., the name used when transmitting media from the local host, for this session is set to `hgs@ursa.fokus.gmd.de`. Messages of the same format are later used to change parameters of a session in the *opened* state.

After the parameters have been set, the conference controller prods the media agent(s) to establish the actual network connection for this session with the message "*sessionname* `open` *network address*", where the network address is either an IP multicast address or a list of unicast addresses. Hosts reached by unicast can also be added later by issuing more `open` messages. The media agent answers with a message of the format "*sessionname* `opened` *network address*" for each network association created. The network address is repeated so that other applications can use it to create a record for that particular session without having to process the `open` message.

New session members become known to media agents either through multicast control messages as in the RTCP protocol or successful connection setup in ATM or ST-II networks. The media agent introduces a new session member to the local conference applications by sending a `hello` message, as in "*sessionname* `hello` *canonical name*".

A number of messages have been defined for the interaction of controllers and media agents. Media agents can be asked for statistics, either for an individual member or the whole media session, with a "*sessionname* `statistics?`" request. The media agents responds with a message of the form

sessionname `statistics` {*parameter value*} …

It lists, for example, the number of packets lost or received out-of-order, as well as all the RTCP source description values.

A media agent can declare a member active if the agent is currently receiving media data from that session member. This indication is of interest for intermittent senders, e.g., audio streams with silence suppression or receiver-side talker indication. The `active` and `!active` (not active) messages indicate the current member activity.

In centralized or connection-oriented conferencing, a conferee is a session member as long as the connection exists. Members of IP multicast-based conferences do not maintain connections to each other, so a different notion of membership and reachability has to be used. RTP media agents, for example, multicast an RTCP control packet every few seconds to tens of seconds, depending on the network bandwidth available for the session and the current number of session members. If a media agent has not received such a packet from another member for a small multiple of this interval, it is likely that network connectivity to that member has been temporarily lost or that the member's media agent has crashed. Conference controllers can use this indication, relayed with the `alive` and `!alive` (not alive) messages, to display member status information in their user interface.

A controller can issue a "*sessionname* `send` *flag*" message to temporarily enable or disable the media data flow from the local media agent, where *flag* is a boolean value. The initiative for not rendering (displaying video or playing out audio) incoming media data can come from either the media agent or a controller. The former can occur if, for example, a session member is sending data of an unknown format. In the more common case, the conferee uses the local conference controller to mute audio from a particular session member if, say, that session member has accidentally left his microphone on. A receiver-oriented floor controller also uses these messages to grant and withdraw the floor, without needing cooperation from the sending side [27]. Since both media agents and controllers can initiate muting, we define a request message, "*membername* `receive` *flag*", and a status indication, "*membername* `receiving` *flag*". The controller would only change its member status display after receiving confirmation through the `receiving` message. The request-indication approach also avoids inconsistencies if several controllers influence the receiving status.

The media agent or a specialized recording tool can be told to play back and record sessions with the `play` and `record` messages. Note that the conference controller does not have to be aware of whether these commands are handled by the media agent rendering the media data (as is the case in our implementation) or by a separate "VCR" tool.

To remove an individual member or a whole media session, the controller sends a "*membername* `close`" or "*sessionname* `close`" request, which the media agent confirms with a `closed` message.

4.4 Error Handling

While asynchronicity and multicast simplify the remainder of the system, they tend to complicate error handling. Since commands are multicast to a receiver group of unknown size, there can be no explicit success indication. We operate under the assumption that commands usually succeed, with error notices generated if not. This 'negative acknowledgement' approach also avoids the ACK implosion problem, where the sender of a request is inundated with the responses of all the recipients. If the recipient of a message needs to signal an error condition, it resends the offending message, prefixed with the `error` command. Generally, the sender should do most of the validity and range checking before sending parameter change messages.

4.5 Starting up Media Agents

So far, we have silently assumed that media agents are started up when necessary. This is reasonable for systems with a single controller and as long as a media agent handles either exactly one or all media sessions of a given type. (It may, for example, be advantageous to have only a single media agent for all audio sessions to allow easy mixing.) If more control is desired, e.g., to support multi-media agents, pmm can queue a message and start a new media agent if the message cannot be delivered to any connected application. It is sufficient to implement this for the `create` message type. pmm maps the media type to the proper executable through a configuration file. Whenever a message filter is changed, pmm checks whether it can now deliver the queued request.

In order to also handle the multicast case, the controller starts a timer after issuing a `create` command. If no `created` response has been received within the timeout period of a few seconds, the controller itself starts a copy of the media agent, again based on some local configuration file. This mechanism allows media handlers to decide whether to handle one, a few or all instances of a certain media type. However, if an application has been suspended by the operating system, multiple agents risk being started up without need. ToolTalk has built-in facilities to start applications when there is no registered recipient for a particular message.

4.6 Late Joiners

There is no requirement that all media are brought into a conference at the same time, as each media instance and media type has separate messages. Also, starting several applications for a single media instance requires no additional mechanisms, whether they are started together or added sequentially, as applications ignore `create` and `open` commands for their own sessions and existing network associations, respectively. However, an explicit query mechanism for sessions and session members offers additional flexibility. Media agents should answer a "*sessionname* `hello?`" request with a sequence of `hello` messages for each of their session members. Instead of implementing requests for each possible session or member parameter, the `statistics?` and `statistics` messages described earlier provide a complete snapshot of the current session or member state.

Instead of explicit queries, media agents could also periodically multicast their session and member identities for the benefit of late-joining applications, just as it is done with RTCP packets. The additional processing overhead, the delayed start-up and the relative rarity of applications joining an on-going session make this alternative unattractive.

5 Implementation

The architecture of Fig. 1 is shown instantiated with MBONE applications in Fig. 4. A screen dump of the conference controller and media agent are shown in Fig. 5. A media agent for packet audio tool called NeVoT (Network Voice Terminal) [28, 29], shown in the lower left of Fig. 5, communicates with a session creator and activity indicator, icc. The same "audio engine" is also available without a user interface, to allow invisible integration into other applications, e.g., virtual reality environments [30].

icc starts up as a session creator, shown active in the lower right quadrant of the figure. It then becomes an activity indicator after starting up the media agents. icc as an activity indicator for an MBONE conference is shown in the upper left. The session creator view is rarely seen by the user, as icc is typically started from a session directory like sd or a conference invitation tool, which supply it with the necessary parameters. Tkaudio, shown at the top right, is responsible for adjusting the audio volume, input and output ports for all local conferences. Implementation complexity has been modest: the message distributor pmm consists of about 400 lines of C code, the command handling in the media agent weighs in at about 900 lines. Due to their design, messages can be parsed without translation by a Tcl interpreter, greatly simplifying the message receiver code. Each session and member is simply dynamically defined as a Tcl procedure. Currently, applications use a small Tcl interface routine that allows them to be dynamically configured for either unicast TCP or UDP communication with the pmm distributor or to use local multicast. The Tcl-DP package provides access to Unix sockets from within Tcl. With minor changes to that interface routine, ToolTalk could be fitted underneath as well (Fig. 2).

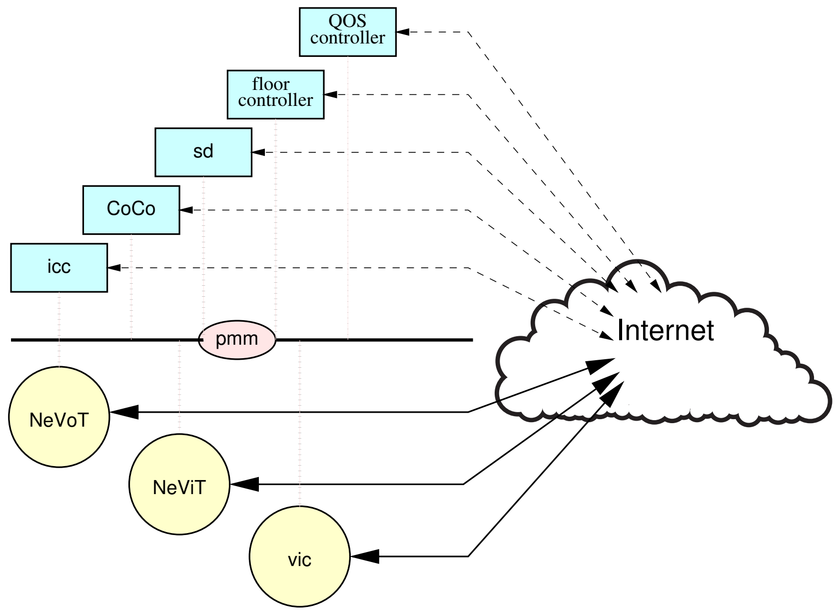

Figure 4: Currently implemented conference elements

A simple video tool, NeViT (Network Video Terminal), that is shown in Fig. 4 is mainly intended for performance measurements. Existing media agents require adding a small interface layer that translates the messages into application-specific actions. This has been accomplished easily for the vic video tool [27], as its user interface is also Tcl-based. Adding a similar interface to a WWW browser would allow sharing of a single browser between applications and remote-controlled "slide shows".

Some initial helper applications have been implemented. A small applet listens to multicasts on the local network from the Active Badge [31] system. Whenever a person enters or leaves the room, the active badge tool sends a "`* name` *names*" message, which then has all media agents change their outgoing RTCP name field to indicate who is present.

Recently, stereo placement of session members was added to NeVoT. Instead of modifying the conference controller, another small application containing nothing but names and sliders was added. The application listens for `hello` messages and adds sliders as new users join the session. Moving the slider sends a message understood by the audio media agent that acoustically moves the speaker to the left or right, allowing the creation of a virtual conference table. Automatic placement around a graphically displayed "table" would be easy enough to add, again without changing the main session controller or media agent.

An SNMP management agent and quality of service controller [32] gather statistics and set bandwidth control parameters or media encodings, without the media agent having to be aware of the fact that it is now subject to another master for a part of its functionality.

Finally, work on integrating a phone-style conference controller called CoCo is in progress [5]. Another

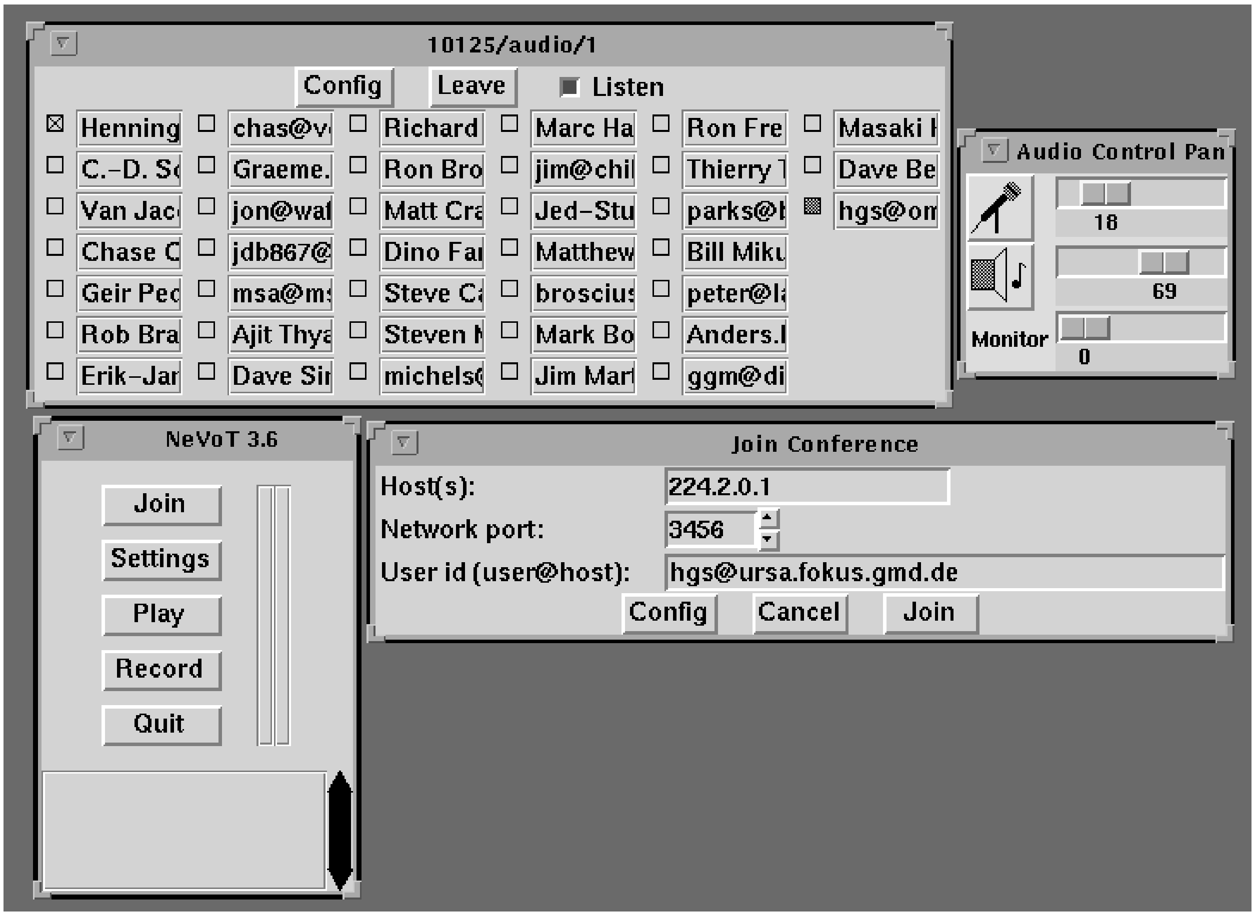

Figure 5: Screen dump of NeVoT (bottom left), icc (top left and bottom right) and tkaudio (top right)

small application planned that is easy to implement within this framework is a talk timer that would periodically summarize the activity of conferees, e.g., for the floor controller.

Currently, all local conferences and all users on a workstation share a single distributor, however, a simple rendezvous mechanism (say, through a file at a well-known location created by pmm) could be used to restrict pmm to a single user. Maintaining a single message distributor per user eases the implementation of all communication modes listed earlier, including across local conferences.

For multicast, all applications share a single multicast address, with different UDP ports for each user. The port is allocated based on the numeric user id (uid). Instead of differentiating by port, we could allocate a new multicast address, but it is more difficult to determine if a multicast address is already in use locally. A user's active conferences share this port, since a single instance of the NeVoT audio engine handles audio input and output for all local conferences. This is necessary since many operating system audio services provide only a single concurrent audio input and output stream, so that mixing has to be done by the application [33]. Once the kernel or the audio device handles mixing, each conference should obtain its own port to reduce message handling overhead, as implemented in [27] (see Section 7). That port would be assigned by the application creating the conference.

6 Discussion and Open Issues

While the information passed between controllers and media agents is likely not to be as sensitive as the actual media data, there is still a need for additional security. Otherwise, any user with access to the host might enable the microphone for convenient eavesdropping. It is probably best to simply encrypt the messages after including a message digest, offering both privacy and message integrity. Since communication is between processes owned by a single user, keys can be handled through the file system. Without encryption, discarding messages from other hosts offers some measure of protection. pmm can provide security similar to that offered by the xauth "magic cookie" method in the X11 windowing system.

The media agents presented so far are symmetric, that is, the agents running at all conferees 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, 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.

While there is great flexibility in combining controllers and media agents, there has to be some coordination between controllers. For example, if a session directory starts a media session, it must start the conference controller containing activity indications as well.

Currently, a text-based protocol is being used as it simplifies debugging and allows direct use of tools like Tcl. Messages are sent on the order of once every few seconds, infrequently compared to the typical arrival rate of media packets, and thus the parsing costs are likely to be fairly small. Other considerations, like the necessity to escape various characters or carry extended character sets, might lead one to consider a binary format, such as XDR (used by ToolTalk) or ASN.1.

Applications beyond conferencing may also benefit from this local messaging service. For example, a database engine might continue to hold a lock on a database until it is asked to release it by a message, rather than repeatedly locking and unlocking the database just in case some other process might need access. Simply by limiting message distribution, priorities can be provided, so that applications of lower priority do not block those of higher priority. (This has been implemented in a different context for audio device access, as described below [27].)

For many applications, where signals are currently used, with their attendant problems due to asynchronous operation, local multicast would be preferable. Examples include notifying a number of processes of audio device changes (like speaker volume) or mail spool file changes.

Inter-application messaging also supports the ability to the separate user interface and the actual media processing, at a significantly higher semantic level of abstraction than, say, the X protocol. This makes it particularly easy to provide user interfaces of very different shape and complexity, without altering the underlying media handling.

Current IP multicast implementations can only designate whether packets are to be looped back to the sending *host*. There is no way for a process to disable receiving its own local multicast packets, but still receive those from other processes running on the same host. The receiving process has to recognize its own transmissions by message content or source port. This deficiency should be addressed if local multicast is to become widely used for local interapplication communication,

7 Related Work

The idea of subscribing to events is also found in [34], 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. As was mentioned earlier, the ToolTalk system [35] implements a generic inter-application service, based on a message distributor called `ttsession` and ONC RPC. It also offers the notion of filters, but with a somewhat more restricted functionality. Unfortunately, it is not available on all Unix platforms, nor on non-Unix operating systems, and seems to be largely unused except for the Sun desktop tools.

The CCCP approach [11] to naming components is reflected in the message formats. PMM could also be considered as a status-based system, as used for control networks [36]. The idea of receiver-controlled message filters described here is similar in spirit to the stream filters central to the RSVP resource reservation protocol [37].

Rangan and Swinehart [12] present a conference control architecture that shares the division into conference controllers and media agents with PMM. However, there a single central entity per conference, the connection manager, handles both conference control and message distribution, while our approach separates the two functionalities. Similar to the request/response model used here, "action reports" ensure synchronized execution even with asynchronous message exchanges. The system in [12] also shares our concern with the ability to introduce recording and playback agents into conferences, but does not consider the issue of having several independent controlling entities.

Independently, McCanne and Jacobson implemented an IPC mechanism based on local UDP/IP multicast that mediates access to the workstation audio device by several concurrently running vat audio tools through a per host multicast address and port. Per-conference UDP ports are used to implement voice-switched-video, enlarging the video window of the current talker in the vic video tool. The port is selected by the session directory sd [27]. In [27], the authors mention their desire to move to an integrated user interface and discuss other applications such as floor control and synchronization.

8 Conclusion

This paper has presented a local conference control architecture, allowing the composition of multimedia conferences in a modular, incremental and transparent fashion. Media agents and controllers can be designed independently, with a "narrow" interface between them, and can be combined with minimal manual configuration. Each component offers information, which is then selected by other components according to their needs. Applications that implement this scheme can be remotely controlled during the whole lifetime of a session. The same application can appear as a GUI-less daemon, as part of a larger application (e.g., a WWW browser, a help system or a virtual environment) or as an independent application, without affecting the remainder of the system. The media agents need not be aware of whether they are part of a "loosely controlled" or "tightly controlled" session, or indeed anything resembling a traditional multimedia conference.

This approach allows the construction of powerful collaboration tools with minimal mutual awareness. This multicast control greatly simplifies programming as all interested parties can track the global state, without explicit assistance from the party causing the change.

While the discussion has focused on managing conferencing applications, local multicast, possibly enhanced with filtering, offers a powerful tool for other application domains.

9 Acknowledgements

The stereo location tool was added by Thomas Becker. The video sender and the QOS monitor were implemented by Bernd Deffner and Dorgham Sisalem. Frank Oertel wrote the multimedia conference controller CoCo. Ingo Busse fitted an SNMP agent to the ensemble. Timur Friedman provided valuable comments.

References

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

[2] H. Eriksson, "MBONE: The multicast backbone," *Communications ACM*, vol. 37, pp. 54–60, Aug. 1994.

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

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

[5] H. Schulzrinne, F. Oertel, and C. Zahl, "Personal mobility for multimedia services in the internet." submitted to *European Workshop on Interactive Distributed Multimedia Systems and Services*, Mar. 1996.

[6] V. Jacobson, "sd, the LBL session directory." Manual page, Nov. 1992.

[7] T. J. Frivold, R. E. Lang, and M. W. Fong, "Extending WWW for synchronous collaboration," in *Proc. of the Second World Wide Web Conference '94: Mosaic and the Web*, (Chicago, Illinois), Oct. 1994.

[8] J. Glicksman and V. Kumar, "A SHAREd collaborative environment for mechanical engineers," in *Proc. of Groupware'93*, pp. 335–447, 1993.

[9] J. S. Donath and N. Robertson, "The sociable web," in *Proc. of the Second World Wide Web Conference '94: Mosaic and the Web*, (Chicago, Illinois), Oct. 1994.

[10] E. Shapiro, "Virtual places – a foundation for human interaction," in *Proc. of the Second World Wide Web Conference'94*, (Chicago, Illinois), Oct. 1994.

[11] M. Handley, I. Wakeman, and J. Crowcroft, "The conference control protocol (CCCP): a scalable base for building conference control applications," in *SIGCOMM Symposium on Communications Architectures and Protocols*, (Cambridge, Massachusetts), pp. 275–287, Sept. 1995.

[12] P. V. R. Rangan and D. C. Swinehart, "Software architecture for integration of video services in the Etherphone environment," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1395–1404, Dec. 1991.

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

[14] L. Wall and R. L. Schwartz, *Programming perl*. Sebastopol, California: O'Reilly, 1991.

[15] 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.

[16] A. L. Ananda, B. H. Tay, and E. K. Koh, "A survey of asynchronous remote procedure calls," *ACM Operating Systems Review*, vol. 26, pp. 92–109, Apr. 1992.

[17] E. Walker, P. Neves, and R. Floyd, "Asynchronous remote operation execution in distributed systems," in *Proc. 10th International ConferenceDistributed Computing Systems (ICDCS-10)*, (Paris, France), IEEE, May 1990.

[18] T. J. Frivold and R. E. Lang, "Conference control glossary." based on presentation to MMUSIC working group at 27th IETF meeting, Amsterdam, Netherlands, July 1993.

[19] B. Bhargava, E. Mafla, J. Riedl, and B. Sauder, "Implementation and measurements of an efficient communication facility for distributed database systems," Technical Report CSD-TR-783, Department of Computer Science, Purdue University, West Lafayette, IN 47907-2004, June 1988.

[20] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.

[21] SunSoft, *ToolTalk 1.1.1*. SunSoft, Mountain View, California, Nov. 1993.

[22] SunSoft, "The messaging object service," white paper, SunSoft, Mountain View, California, Apr. 1994.

[23] 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*, Mar. 1995.

[24] H. Schulzrinne, "RTP profile for audio and video conferences with minimal control," Internet Draft, GMD Fokus, July 1995. Work in progress.

[25] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform resource locators (URL)," RFC 1738, Internet Engineering Task Force, Dec. 1994.

[26] J. Postel, "Media type registration procedure," RFC 1590, Internet Engineering Task Force, Mar. 1994.

[27] S. McCanne and V. Jacobson, "vic: A flexible framework for packet video," in *Proc. of ACM Multimedia '95*, Nov. 1995.

[28] H. Schulzrinne, "Voice communication across the Internet: A network voice terminal," Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.

[29] H. Schulzrinne, *Guide to NeVoT*. GMD Fokus, Berlin, Germany, 3.32 ed., Sept. 1995. The software is available from ftp://gaia.cs.umass.edu/pub/hgschulz/nevot.

[30] E. Frécon, H. Eriksson, and C. Carlsson, "Audio and video communication in distributed virtual environments," in *Proceedings of the 5th MultiG Workshop*, (Stockholm, Sweden), Dec. 1992.

[31] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems*, vol. 10, pp. 91–102, Jan. 1992. also Olivetti Research Limited Technical Report ORL 92-1.

[32] I. Busse, B. Deffner, and H. Schulzrinne, "Dynamic QoS control of multimedia applications based on RTP," in *First International Workshop on High Speed Networks and Open Distributed Platforms*, (St. Petersburg, Russia), June 1995.

[33] H. Schulzrinne, "When can we unplug the phone and the radio?," in *Proc. International Workshop on Network and Operating System Support for for Digital Audio and Video (NOSSDAV)*, (Durham, New Hampshire), pp. 183–184, Apr. 1995.

[34] 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.

[35] A. M. Julienne and B. Holtz, *ToolTalk and Open Protocols*. Mountain View: SunSoft Press/Prentice Hall, 1993.

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

[37] 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.