Dynamic Configuration of Conferencing Applications using Pattern-Matching Multicast

Henning Schulzrinne

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

Abstract. Multimedia conferencing systems are usually large, complex software systems. We describe a local control architecture and communication protocols that allow to tie together media agents, controllers and auxiliary applications such as media recorders and management proxies into a single conference application. Unlike other systems, control of a single conference can be shared between several controllers. Each media 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 replicator. The replicator 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 functionality was also implemented IP multicast restricted to the local host.

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 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, with a single application that at most would spawn some helper applications. Secondly, every such suite of applications had its own set of protocols, so that there was little opportunity to replace, say, one video agent by another.

The set of media agents[1] built by a diverse set of research groups for use

[1] We generally follow the terminology of [1], where media agents are defined 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".

on the Internet multicast backbone (MBONE) [2, 3] depart from this approach. Here, each media is typically handled by a distinct media agent, with control delegated to an external conference controller [4]. This separation allows graceful evolution, 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 multicast or centralized directories as well as a "telephone-style", "tightly-controlled" invitation-based conference [5]. Examples of a multicast directory include `sd` [6]), while WWW is becoming popular for centralized session rendezvous [7, 8].

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, 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 tools.

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

Controller(s) to media agent: A single conference may have multiple controllers, i.e., media agents may receive commands from multiple sources. In a typical scenario, there might be a basic conference controller that issues and answers invitations to join conferences, as well as manage conference parameters such as media encodings. (These might well change during a conference to accommodate new members or different interaction requirements.) In addition, a floor controller enables and disables individual media applications on the sending side or disables receiving from all but one session member. Both can be implemented independently, but the floor controller, for example, needs to know when conferences start and stop or members leave.

There may also be purely local controllers, e.g., a radio broadcast controller might automatically enable the audio media agent on the hour to tune in to the news.

It should also be possible to have both per-media 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 easier determination of how a conferee is participating in a conference, but also may be less natural than associating the information directly with the object controlled by the conferee, e.g., a pointer or a video thumbnail. 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 and other applications within a conference may also want to 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.

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 on 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, with different requirements in terms of functionality, naming, security and reliability.

Most of these control functions 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 [9].

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 [10]. 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 on an asynchronous communication model rather than the reply-response model. In all these properties, it differs from standard RPC[2].

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 and media agents. The former are conference control applications of various sorts, as indicated above, while examples of the latter also include shared applications, media recorders, and the like. The replicator simply forwards control messages to controllers and media agents that have expressed interest, without regard for content. The replicator is not aware of the distinction between controllers and media agents.

[2] [11, 12] discuss asynchronous RPC.

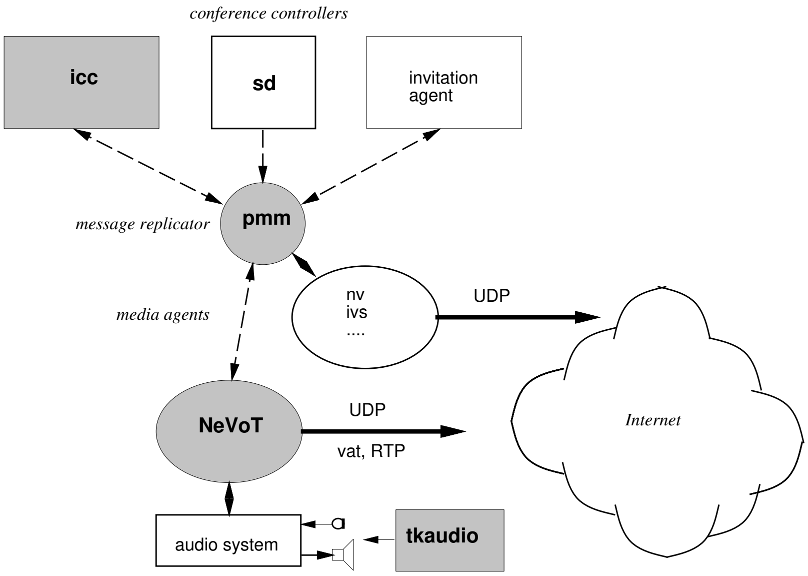

Fig. 1. Example multimedia conferencing control architecture

These control messages could be delivered through any Unix IPC mechanism (Unix sockets, TCP connections, UDP messages, RPC, pipes, stdin, ...) or specialized in-memory multicast support [13]. Three IPC methods have been implemented. In the first, a single message replicator process called `pmm` is used, listening for connections at a well-known TCP port. The second approach also uses a replicator process, but communicates using UDP datagrams. As described later, the replicator process can be told by each 'attached' process which messages it wants to receive, based on the message content. In the third approach, no replicator process is needed as applications send UDP datagram to an IP multicast address [14]. The reach of the multicast packets is restricted by setting their time-to-live value to zero[3]. Local multicast offers the advantage of dispensing with the replicator process and avoiding maintaining membership information in the replicator. 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 replicator since the interests of different listeners will have non-zero intersection. The relative run-time penalties of replicator vs. multicast will largely depend on the number of processes attached to the replicator or listening to the local multicast address.

The reason for choosing a network-based IPC rather than a Unix-domain socket or pipe is the easy N-to-1 connectivity, reaching if necessary beyond a

[3] To tie together applications on different hosts, a larger ttl value is needed.

single host. This allows the user to spread a single local session over several workstations, e.g., to harness additional processing power or special hardware resources.

A TCP-based message replicator has the advantage that it continues to work, if inefficiently, even if media agents are distributed over several hosts, while multicast and UDP-based replicators 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.

Currently, all conferences and all users on a workstation share a single replicator, 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 replicator per user eases the implementation of all communication modes listed earlier, including cross-conference. Similarly, a single well-known multicast address and port per user, as currently implemented, avoids rendezvous problems, particularly if there is no master application to track assignments. To reduce the number of processes needing to filter messages, per-conference or per-media multicast ports could be added. Some means of allocating a multicast address and UDP port must be found. This is more difficult if there is no single controlling entity.

The system presented here uses constants and information provided by the Real-Time Transport Protocol (RTP) [15] and the audio/video profile of that protocol [16]. In that protocol, 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 about received sites. The canonical name is formatted like an Internet email address and meant to be the same for each user, across media and conferences. Note that the architecture does not depend on the use of RTP/RTCP or the particular form of the canonical name[4].

3 Protocol Operation

3.1 Message Format and Naming

In the following, we describe the basic operation of the "protocol" between conference components. All messages are sent as ASCII text and are formatted to be directly interpretable by a standard Tcl [9] interpreter.

Generally, messages are idempotent, so that the second request to join the same multicast group is simply ignored. This allows helper applications that are added later to an ongoing session to be initialized; the new application could simply send a query broadcast to all media agents, who would then respond with the necessary commands to join the ongoing sessions. Alternatively, the conference controller could distribute the information periodically.

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

Each media session is named by a hierarchical descriptor containing identifiers for the conference, media and media instance, e.g., *C/audio/3* denotes the third audio session within conference C, where C is a locally unique name. Generally, each media session will have one media agent. It is assumed that one conference controller for a conference names the media sessions. A single application may implement several media agents, either for a single media type or several, but this is invisible to the protocol. Members of a media session are named by appending their RTP canonical name to the session name, e.g., *C/audio/3/hgs@ursa.fokus.gmd.de*. Messages are constructed to name the object, that is, a session or session member, followed by the operation to be applied to that object, followed by any parameters needed.

3.2 Basic Operation

Applications register or unregister for messages with the replicator by sending a wildcarded string prefixed by `+` or `-`, respectively. Currently, these strings follow Unix shell wildcard 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 replicator if it wants to track talker activity.

First, the conference controller sends a message creating a "blank" media session: `session C/audio/3` and waits for a `created` reply from the responsible media agent before proceeding. Auxiliary applications should not answer to avoid synchronization problems. If different media are to have different control ports to achieve reduced message processing load, the port number is conveyed as well.

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

```
C/audio/3 ttl 128
...
C/audio/3 transport RTP
C/audio/3 media PCMU 1 8000 0.02
C/audio/3 cname doe@foo.bar.com
```

The same commands are used to change parameters during the session. After the parameters have been set, the conference controller prods the media agent(s) to establish the actual network connection for this session: `C/audio/3 open 224.2.0.1`. For unicast or point-to-multipoint networks like ST-II or ATM, the `open` command can be repeated with different network addresses for the same session. If a network connection was successfully established, the responsible media agent answers with `C/audio/3 opened 224.2.0.1`. This message might also be used by other applications to create a record for that particular session.

A media agent rather than the controller introduces a new session member to the other local conference applications by sending a `hello` message, as in `C/audio/3 hello hgs@ursa.fokus.gmd.de`. Session members become known to media agents either through IP multicast or successful point-to-multipoint connection setup. Controllers can then send messages to the media agents to enable or disable sending and receiving, e.g., to achieve audio muting by a floor controller.

A number of other 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. A media agent sends `alive` and `active` messages to inform others about changes in member status. To remove individual member or a whole media session, the controller sends a `close` command, which the media agent confirms with a `closed` message.

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 usually explicit success indication, but rather error notices received by all receivers of the original command. Indeed, since a command may be processed by several receivers, there may be several error messages generated. However, as evidenced by the `create/created` and `open/opened` pairs, success indication for *one* receiver is used to keep the controller from sending messages to the media agent before the agent is ready to receive them.

3.3 Starting up Media Agents

So far, we have silently assumed that media agents are created 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. (For audio, it may be advantageous to have only a single media agent for all sessions to allow easy mixing.) If more control is desired, e.g., to support multi-media agents, the replicator could check whether any application has registered for '*/media/* create' messages and invoke missing applications if not. In the multicast case, the controller would issue a `create` command, wait for the `created` response, timing out and starting a media agent if there were no takers.

4 Implementation

The gray elements shown in Fig. 1 have been implemented. A screen dump of the conference controller and media agent are shown in Fig. 2. A media agent for packet audio called NeVoT, shown in the lower left of Fig. 2, communicates with a session creator and activity indicator labeled `icc`. `icc` combines a session creator, shown active in the lower right, which 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 activity indicator is rarely seen by the user, as `icc` is typically started from a session directory like `sd`, which supplies it with the necessary parameters.

Some initial helper applications have been implemented. A small applet listens to multicasts on the local network from the Active Badge [17] system. Whenever somebody enters or leaves the room, it changes the outgoing user

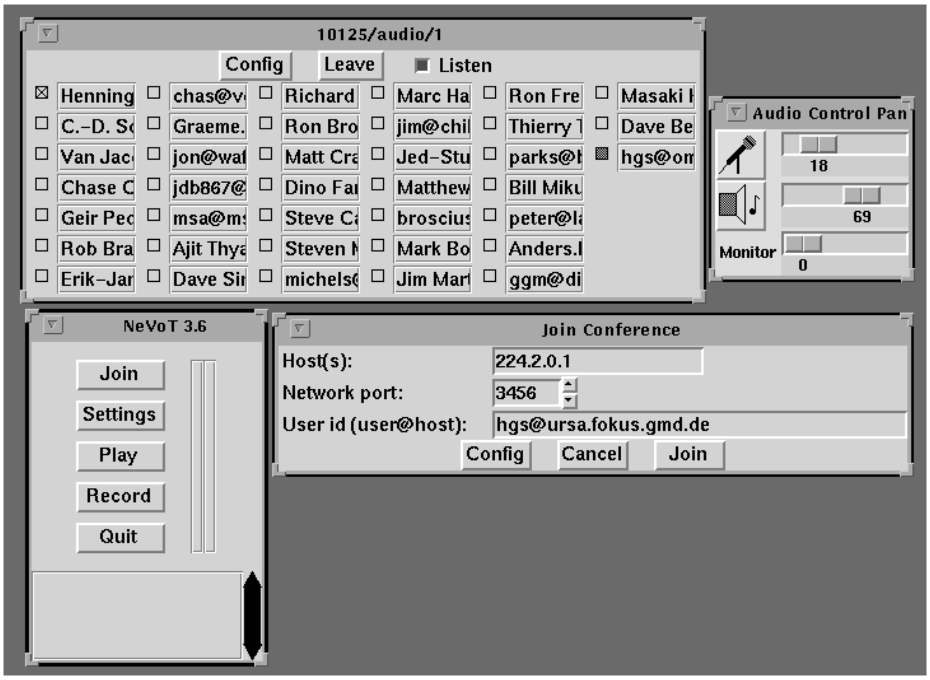

Fig. 2. Screen dump of NeVoT, `icc` and `tkaudio`

name indication to show the list of people gathered around the workstation for all active conferences.

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 add sliders as new users joined 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[5].

Another small application planned that is easy to implement within this framework is a talk timer, possibly sending commands destined for the floor controller. An SNMP management agent could gather statistics and control parameters, without the media agent having to be aware of the fact that it is now subject to yet another master.

Work on integrating a phone-style conference controller and video tools is in progress. Existing media agents require adding a small interface layer that translates the messages into application-specific actions. Adding a similar interface to a WWW browser would allow sharing of a single browser between applications

[5] Automatic placement around a graphically displayed "table" would be easy enough to add, again without changing the main session controller or media agent.

and remote-controlled "slide shows".

5 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, similar to the mechanism used by X11. Without encryption, discarding messages from other hosts offers some measure of protection.

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, 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, a session directory used to start a media session 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, infrequent 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 an XDR or ASN.1-based representation.

Applications beyond conferencing may well benefit from local multicast. For example, instead of repeatedly locking and then releasing a resource as quickly as possible in case some other process needs it, an application could simply hold the resource, waiting for a multicast message explicitly asking it to release it. This also allows to implement priorities, so that applications with lower priorities do not block those of higher priority. (This has been implemented in a different context for audio device access, as described below.)

For many applications, where currently signals are 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.

Current IP multicast implementations can only designate whether packets are to be looped back to the sending *host* or not. There is no way for a process to disable receiving its own local multicast packets; rather, it has to filter by content or source port.

6 Related Work

The idea of subscribing to events is also found in [18], 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. The CCCP approach to naming components is reflected in the message formats [19]. This approach could also be considered as a status-based system, as used for control networks [20]. However, the message replicator offers a selection mechanism beyond a predefined set of multicast addresses, somewhat similar to the receiver-directed filters are also central to the RSVP resource reservation protocol [21].

Independently, McCanne and Jacobson implemented an IPC mechanism based on local 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 ports are used to implement video-follows-audio, enlarging the video window of the current talker in the `vic` video tool. The port is assigned by the session directory `sd`.

7 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 remote 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) 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.

8 Acknowledgements

The stereo location tool was added by Thomas Becker.

References

1. 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.
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," *Communications ACM*, vol. 37, pp. 54–60, Aug. 1994.
4. E. Schooler and S. L. Casner, "An architecture for multimedia connection management," *ACM Computer Communication Review*, vol. 22, pp. 73–74, Mar. 1992.
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. 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. K. Ousterhout, *Tcl and the Tk Toolkit.* Reading, Massachusetts: Addison-Wesley, 1994.
10. 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.
11. 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.
12. E. Walker, P. Neves, and R. Floyd, "Asynchronous remote operation execution in distributed systems," in *Proc. 10th Intl. Conf. Distributed Computing Systems (ICDCS-10)*, (Paris, France), IEEE, May 1990.
13. B. Bhargava, E. Mafla, J. Riedl, and B. Sauder, "Implementation and measurements of an efficient communication facility for distributed database systems," Technical Report Purdue Technical Report CSD-TR-783, Department of Computer Science, Purdue University, West Lafayette, IN 47907-2004, June 1988.
14. 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.
15. 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.
16. H. Schulzrinne, "Sample profile and encodings for the use of RTP for audio and video conferences with minimal control," Internet Draft, GMD Fokus, May 1994. Work in progress.

17. 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.
18. 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.
19. M. Handley and I. Wakeman, "CCCP: conference control channel protocol – a scalable base for building conference control applications." V1.4, Mar. 1994.
20. R. S. Raji, "Smart networks for control," *IEEE Spectrum*, vol. 31, pp. 49–55, June 1994.
21. 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.