Vaudeville: A High Performance, Voice-Activated Teleconferencing Application

Jyoti K. Parwatikar, T. Paul McCartney,
John D. DeHart, Maynard Engebretson and
Kenneth J. Goldman

WUCS-96-18

6/4/96

Distributed Programming Environments Group
Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

Vaudeville: A High Performance, Voice Activated Teleconferencing Application

Jyoti K. Parwatikar, T. Paul McCartney, John D. DeHart, Maynard Engebretson and Kenneth J. Goldman
Department of Computer Science
Washington University
St. Louis, Missouri 63130
{jp, paul, jdd, ame, kjg}@cs.wustl.edu

ABSTRACT

We present a voice-activated, hands-off, ATM-based video conferencing application. The application, called Vaudeville, features high quality NTSC video, voice-activated audio transmission, audio bridging of two audio streams, and voice-activated video switching. It supports multiple simultaneous multi-party conferences using a scalable multicast mechanism. We describe how Vaudeville was built using a component-based distributed programming environment. We also describe the algorithms used to control the audio and video of the application. Audio and video are encoded in hardware using an ATM hardware multimedia interface.

KEYWORDS: distributed computing, multimedia, video conferencing

1 INTRODUCTION

The main goal in developing Vaudeville was to create a teleconferencing application using ATM that facilitates a natural conferencing environment. This would be an environment that comes closest to a real-life conference that takes place in a meeting room. Video should be high quality. The application should be intuitive to use, and users should be required to touch the interface as little as possible during a conference. Conferences should be scalable. The application should be able to support multiple conferences simultaneously, and users should be able to move as easily among the conferences as they would between meeting rooms in a building (or in this case even more easily).

To achieve these goals, audio transmission is voice-activated, and the video automatically switches to reflect which person is speaking. The voice-activated audio means the user does not need to make any indication to the system that they wish to speak other than making noise. The video switching mimics the pattern of focus of an average conference participant. Also since the video is studio quality NTSC (30 frames/sec), the video is similar to what a participant would see if they were sitting in the same room with the speaker. Audio and video is described in more detail is section 1.2.

To allow conferences to be scalable, a two speaker paradigm is used. At any given moment, a conference has two designated speakers, a primary speaker and a secondary speaker. The designated speakers may change dynamically throughout the conference with voice-activated switching. Video and audio streams are associated with each speaker. The video of the primary speaker is shown by default and the audio streams of the two speakers are mixed. Two speakers are used rather than one to allow the case in which one person is speaking and one person wishes to interrupt. We do not have more than two speakers at once, because presently the hardware only supports the bridging of two audio streams. In practice, we have not seen a need for more than two speakers. Contention between audio streams is resolved in hardware, this resolution is described in section 1.2. Video for a participant is transmitted only when they are a speaker. Independent of the size of the conference, only two videos are transmitted per conference. However, participants have the opportunity to see the video transmission of anyone in the conference by explicitly selecting that participant.

The Vaudeville software was developed using The Programmer's Playground, a distributed programming environment. A brief overview of Playground is described in Section 1.1 of the paper. Details may be found elsewhere [4,5].

Unlike the mbone tools, we have decided on concentrating on high quality video. For two reasons, first, we believe that the high quality video makes one less conscious that the conference is virtual. Second, for conferences in which participants want to use a video source as a center of discussion it is important for video to be very high quality especially for medical applications. Like vic, the video is voice-activated, however, we do not have the ability to show several video streams at once.

1.1 THE PROGRAMMING ENVIRONMENT

Vaudeville is implemented in the context of *The Programmers' Playground,* a software library and run-time system that supports a new programming model for distributed applications [3,4,5]. The model, called I/O abstraction, provides a separation of computation from communication structure, making it well-suited for end-user construction of customized distributed applications from computational building blocks. Playground users do not need to write any source code to establish communication between the modules of a distributed application, nor do they need to understand the details of how communication occurs.

A distributed application consists of a set of communicating modules, written as C++ programs, using special data types from the Playground library. The library supports several standard data types such as integer, string, character, and mappings. Along with these data types, Playground supports a structure called a *tuple* that allows the combination of data types to form a user-defined type. Instances of these data types can be *published*, making information available to external modules. The communication structure among the modules of a Playground application is defined by a set of logical connections among published variables of the modules. The communication structure is configured dynamically at runtime through a graphical user interface for the connection management system. In our case, Vaudeville itself acts as a front-end for the Playground connection management system.

The connection management system consists of two parts, a protocol run by each Playground module and a connection manager that is a Playground module itself. The connection manager does type and protection checking when a connection is made, and informs the modules involved in the connection of the new link. After the modules have been informed of the new connection, all further negotiation and communication is handled individually by the endpoint modules' protocols.

Communication among modules of a distributed system occurs implicitly. When a published variable is modified, the new value of the variable is automatically sent to its connected variables. In this way, the programmer does not need to know the low level details of how the communication is actually achieved. Each module can be created independently of the modules with which it communicates.

Prior to Vaudeville, all Playground communication was over TCP/IP using Unix sockets. To support ATM connections, the ATM connection management system was combined with the Playground connection management system. Also, audio and video variables were added to the Playground runtime library. If a connection is made between audio or video variables, the combined connection management system makes an ATM connection between the two sites.

1.2 HARDWARE TESTBED

Vaudeville was built using the Project Zeus ATM testbed[1] and the MultiMediaExplorer (MMX)[7]. The MMX is an ATM desktop multimedia hardware interface that is host independent, and requires no computing resources from the host workstation for processing of multimedia streams. It is currently controlled over a RS-232. It allows users to simultaneously transmit and receive audio and video. Video is full-motion, JPEG compressed, NTSC video (60 fields/30 frames per second at 640x480 resolution). The MMX has two audio modes, a default mode and a conferencing mode. The default is CD-quality stereo audio. In conferencing mode, the audio is voice-quality monaural audio.

Vaudeville uses the MMX conferencing mode. The main features of the conferencing mode are the audio-bridging of two streams and voice activated switching. Each MMX is assigned a source id by the application. The id is placed in the payload of every audio cell it transmits. On the receiving side, each MMX then chooses two audio streams from all streams received to bridge and play out. This choice is made based on the source ids of the streams. Smaller source ids have higher priority. Once the two streams have been chosen, the source ids of the streams are written into the MMX's global memory where they are accessible by the application software. Audio cells are transmitted only if a signal is present, and at least one of the audios presently chosen has a priority less than that of the transmitting MMX. A signal is considered present if the signal is larger than the voice activated threshold. This threshold can be set by an outside application.

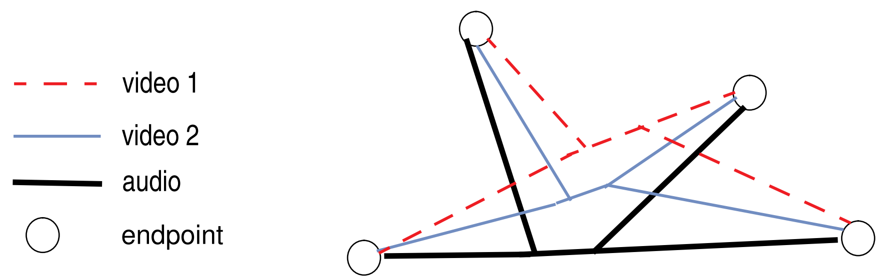

Figure 1: ATM Configuration

Since the ATM software does not presently support many to many multipoint connections, a point to multipoint connection is created from each member of the conference to everyone in the conference including the transmitting member. Since all MMX's receive the same audio streams, all MMX's choose the same two audio streams to play out. Speakers transmit their video on this same point-to-multipoint connection. To receive video from a non-speaker, a point to point connection is setup between the transmitter and the receiver. Ideally, with many to many connections, we would use the ATM configuration model as shown in Figure 1. There are three multipoint to multipoint connections. One connection is for audio, and two are for video one for each of the speakers. The transmitters in the video connections change as the speakers change.

2 FUNCTIONALITY

Figure 2 shows the Vaudeville direct manipulation user interface. The user interface can be divided into four parts -- the conference menu, the people directory, the file menu and the conference window. At the top, is the conference menu that displays the active conference name, and has a list of all currently available conferences. The people directory is the list of names on the right of the figure. This is a list of all of the people currently in the system. The file menu is a pull down menu that appears when File is selected at the top left of the interface. The rest of the interface constitutes the conference window.

Joining a Conference: A conference may be thought of as the virtual equivalent of a meeting room. Users may join a conference by selecting the name of the conference from the conference menu. This causes the user to be removed from any conference that they are currently attending, and to be added to the new conference. By selecting the New Conference option from the File Menu, the user may create a new conference. When this option has been selected, a dialog box appears that asks for the name and attributes of the new conference. After the conference has been created, the user automatically joins this conference. Users may leave a conference by selecting the Leave Conference option from the File menu.

While participating in a conference, the display shows the conference's current state. Each member of the conference is represented by a photo icon. The two current speakers are highlighted with a colored border; the primary speaker in red and the secondary speaker in gray.

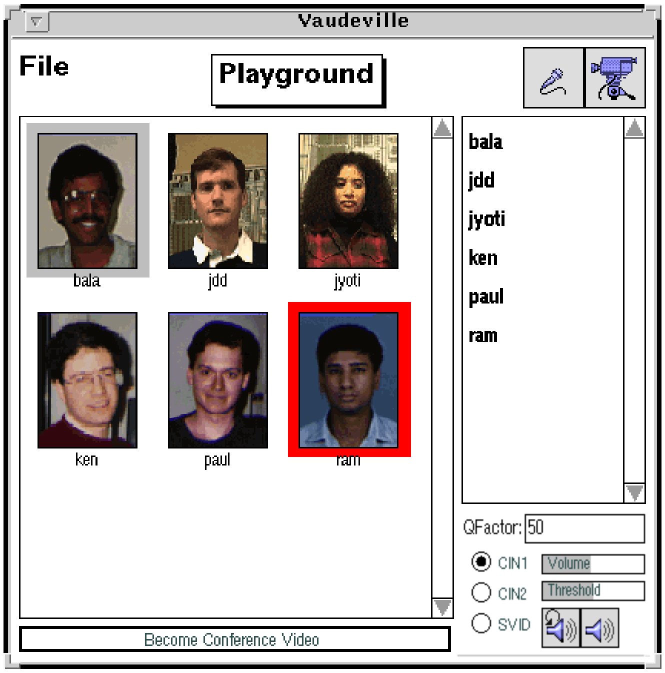

Figure 2: Vaudeville User Interface

Viewing the Conference: Video is viewed on a separate display. There are two options for receiving video. The first option is to receive the default video. In most cases, the default video is the primary speaker's video. However, for the primary speaker the default video is the secondary speaker's video. As speakers change, the video stream selection also changes. When the Become Conference Video button has been selected, the default video for the whole conference is the video belonging to the person who selected the button. This button represents a token belonging to the conference. To release the token, the user unselects the button. When no one holds the token, the default video follows the speakers. Anyone may request the token, and the token moves to whomever requests it. Users do not need to explicitly release the token in order for another user to request it. We rely on conference etiquette for floor control rather than imposing some locking mechanism.

The second option is an override option. The user selects the video received by selecting the icon of a person in the conference. At this point, the default video remains unchanged for the rest of the conference. The selected video is seen only by the user who selected it. To return to the default option from the override option, the user unselects the icon of the received video.

Settings: The set of controls in the bottom right hand corner of the interface allow the user to control the qfactor for the conference, the threshold, the playout volume, mute, selfmute and the video source. The qfactor is the JPEG encoding and decoding compression factor for the entire conference. This can be changed by any participant by typing in a new value for the qfactor. The threshold slider controls the threshold of the voice activated switching. The threshold is the level of noise that must be produced before an audio stream is transmitted. The mute buttons are the two speaker icons in the corner, these control the audio heard by the user. The selfmute button is the icon with the circular loop. When this button is selected the user does not hear their audio stream. When the other mute button is selected the user does not hear any of the audio streams. CIN1, CIN2, and SVID are radio buttons that represent a choice between three different video sources that allows the user to change their video transmission source.

The microphone and camera buttons in the top right-hand corner of the interface allow the user to control when they are transmitting. When transmission is turned off, a red slash appears through the icon. Note that transmitting video or audio only enables reception. It does not necessarily imply that others actually have chosen to see or hear the video or audio. This is useful if a user wants to take a phone call in the middle of a conference, or just wants to observe the conference and not actively participate.

Invitations: Once in a conference, a user may invite others to the conference by selecting a person's name from the people directory. At this time, the name changes from black to green and changes back to black when a reply has been received. On the receiving end of the invitation, an invitation window is created informing the user by whom they have been invited and to which conference. The window also contains buttons which allow the user to accept or reject the invitation. If the user accepts, the user interface responds in the same way as if the user had explicitly chosen to join the conference.

3 SOFTWARE COMPONENTS

Vaudeville consists of five Playground modules shown in Figure 3: the *user interface (GUI)*, the *MMX module*, the *Participant module*, the *conference register*, and the *conference module*. Each user has an GUI, a MMX and a Participant module. There is one centralized conference register, and there is one conference module for each active conference. The MMX module pipes commands to the MMX based on changes to its published variables made by the GUI and the Participant. It also outputs the source ids of the two audio's currently being played. The GUI outputs information supplied by the user to the MMX and Participant modules, and reflects any changes in the conference state. The Participant module's main function is to coordinate connections among the published variables of the five modules, and to control which video is played based on information supplied by the MMX module and the GUI.

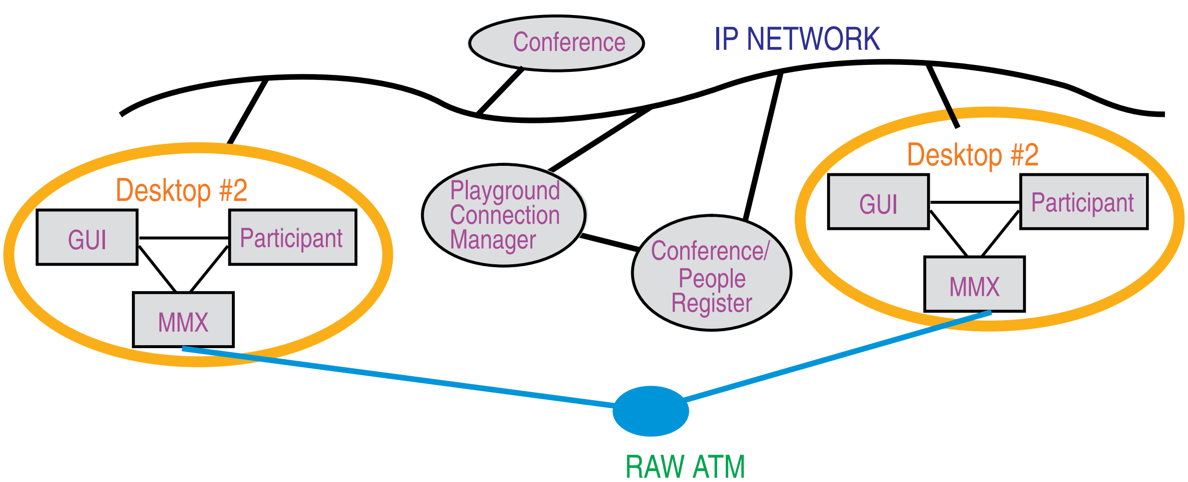

Figure 3: Vaudeville Software Components

For each conference in progress, there is a conference module that maintains the state of the conference and the members of the conference. Each member of the conference has both an incoming and an outgoing connection to the conference module. The outgoing connection supplies the conference with the participant's state information and any request of resources managed by the conference. The incoming connection to the participant supplies the modules with the state information of the conference. The conference register maintains lists of conferences and people maintained by the Vaudeville system. The contents of this register are displayed in the GUI.

The environment can be divided into two parts the desktop environment and the world outside. The desktop environment consists of the mmx, the video monitor, the microphone, the camera, the speaker and the user. Two modules interact directly with the desktop environment the MMX module which interacts with the mmx, and the GUI module which interacts with the user. The Participant module acts as a bridge between the desktop and the rest of the environment. It coordinates this outside information with information from the GUI and the MMX. The outside environment consists of devices, users and other Playground modules. Here resides the conference, the conference register, the connection manager and other users of the system each with their own MMX, GUI, and Participant modules.

3.1 Desktop Modules

MMX Module. The MMX module shown in Figure 4 has two functions, one active and one reactive. The active function polls the global memory of the MMX to read the ids of the two audio streams being played out. The reactive function pipes commands to the MMX when requested. The MMX module's interface consists of three published tuples. The first tuple acts as an interface to the GUI module, and is writable by the environment. It controls parameters which affect only the desktop area namely volume, mutes, video source, and threshold. The second tuple is also writable by the environment, and acts as an interface to the Participant module. It controls MMX functions which may depend on information provided from beyond the desktop. It consists of parameters corresponding to qfactor and a request to open/close a receiver/transmitter. Associated with each of these tuples is a set of reaction functions. When either of these tuples changes, the reaction function causes a command to be piped over an RS-232 serial line to the MMX to make the physical change requested as shown in Figure 4.

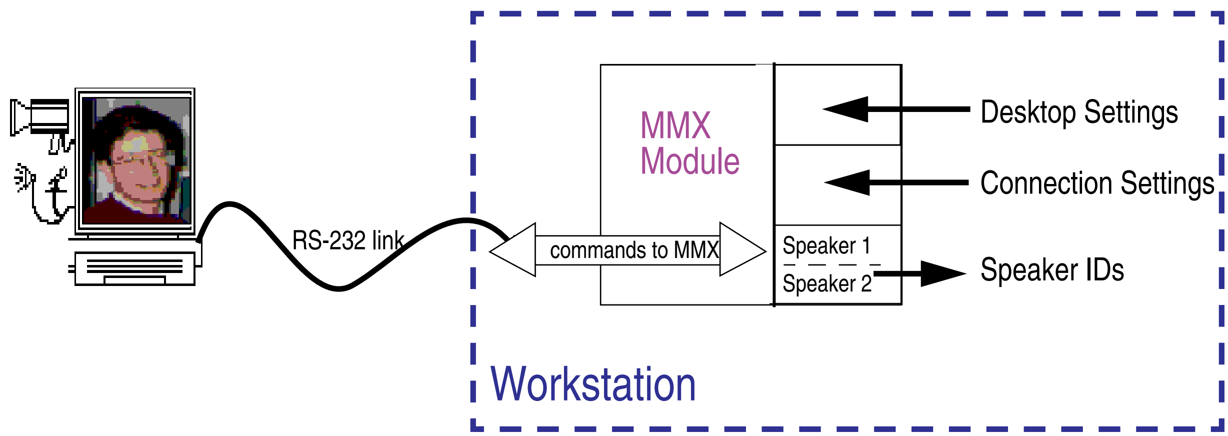

Figure 4: MMX Module

The third tuple is readable by the environment and provides the ids of the two speakers. Periodically, the MMX module polls the global memory of the MMX, and reads the ids of the two audio streams being played out. If there is a change in the ids, the module updates the tuple to reflect the most recent set of speakers in the following manner. If there are two streams reported by the MMX, the module updates the tuple with the two new streams. If there is only one stream reported by the MMX, the module updates the tuple with the new stream and the next highest priority speaker from the old set of speakers. If no streams are reported by the MMX, the tuple remains unchanged.

GUI Module. The GUI module manages the user interface shown in Figure 2. Its purpose is to pass information supplied by the user to the rest of the system, to keep the user informed of the state of the system, and to allow the outside world to communicate with the user. The interface of the GUI module consists of three published tuples. Fields of the tuples correspond to elements of the user interface. Two of the tuples are readable by the environment. These tuples are updated as the elements associated with them are manipulated by the user. Elements associated with the writable tuple are updated whenever a change occurs in the tuple.

The first readable tuple controls the local multimedia parameters, mutes, volume, etc. The second readable tuple allows the user to communicate with the system and make requests for resources held outside of the local environment. This includes invitations, conference selection, and which video to play out. The writable tuple maintains state information about the system and also allows the outside environment to communicate with the user. It includes changes in the conference/people register, and changes in the conference state.

Participant Module. The Participant module is the controlling module. It coordinates information about the whole environment and takes the appropriate action. This is where most of the decision making is done. There are four published tuples. The first is a readable/writable tuple that acts as an interface to the MMX module. This tuple maintains the two current speakers, and represents requests for opening or closing receivers or transmitters. The second tuple reflects the user's current preferences. This includes which video to play out, and which conference to join. This tuple is a writable tuple that acts as an interface to the GUI module. The third tuple is a readable tuple linked to the Playground Connection Manager. This tuple represents a request for links between modules. The fourth tuple is both readable and writable, and acts as an interface to the conference. The two main functions of this module are managing the video, and making connections between modules.

The Participant Module is responsible for deciding which video to play out and when to transmit video. Each Participant module maintains two video transmitters. When the user is a speaker, video is transmitted along with the audio. Otherwise video is transmitted from a second transmitter. When another person wants to view the user, a link is set up to the viewer from this second transmitter. This cuts down on the incoming bandwidth. It also makes the conference scalable, because the number of incoming video streams remains constant as the size of the conference increases. When the user becomes a speaker or is no longer a speaker, the Participant module reacts by switching the video transmitter.

When the speakers change, the Participant module may also have to change the receivers. As for the transmitters, the Participant module maintains two types of receivers, one for viewing speakers, and one for viewing non-speakers. If the user is viewing the default video, then each time the highest priority speaker changes the Participant module switches the video receivers to the video of the latest speaker. Note this switch would not be necessary with many to many connections. If the user has chosen a specific person to watch, then the module switches between the two types of video receivers, as the person being viewed switches between speaker and non-speaker status.

The Participant module is responsible for the configuration of the modules in its control. Upon initialization, connections are made for the initial configuration of the desktop modules. When joining a conference, connections are made to connect the module to a conference. This includes a connection to a Conference module, and the connection to the audio call of the conference. Connections are broken upon leaving a conference. The only other connections made are to support the reception of video associated with non-speaking sources.

3.2 Shared Modules

Conference Module: The Conference module maintains the state of the conference, and manages any parameters or resources that belong to the conference. It has a published list of members that is both readable and writable by the environment. Each time someone new joins the conference, an element is added to the list. Each element of the list is connected to a published variable in the corresponding Participant module. Through the elements of the list, the module assigns source ids to members, and receives requests for resources and changes to conference parameters. Currently, source ids are assigned randomly.

Conference/People Register: The Conference/People Register monitors the system, and maintains a published list of the available conferences and the people currently running Vaudeville. It has two published lists. One is writable by the environment. This is a list of all modules known to the connection manager. Using this list, the module filters out the Conference and Participant modules and places them in a list. The second list is readable by the environment.

4 EXPERIENCE

At present, Vaudeville has been running in the department for about one month and been running on about ten workstations for two months. It is currently running on both SGI and Sun platforms. Presently, all features described have been implemented. Conferences up to ten people have been tried. Because of the high quality video and hands-free floor control of audio, conferences have succeeded in being natural. Users have been taught to use the system in a matter of minutes, and with enough incentive (a good movie playing) they have taught themselves. By using different video sources such as a microscope and a Silicon Graphics machine, participants were able to meet and discuss work from their own labs and offices without any special preparation.

We discovered a few audio problems in the initial testing. At first, the range for the voice activated threshold and the default setting was too low. In large conferences, the audio was chaotic, because the streams were too easily cut off by interruptions from higher priority sources. We increased the default and maximum thresholds. This solved the problem. Another problem was audible pops and clicks. When a speaker paused between words, the voice activated switching caused the transmitter to be switched off which produced a click. This became a real distraction to users, so the algorithm used by the MMX to control the voice activation was changed to leave the transmitter on if there was only one speaker, and to fade out the audio when the transmission was cut. This greatly improved the audio quality of the system.

People almost immediately decided that they didn't want to have all of their conferences open to the public. So we added private conferences that would not appear in the conference directory and could only be joined through an invitation. To allow people to know in which conference a person could be found, we added a feature to the people directory that prints the name of the conference a person is in next to their name.

5 CONCLUSIONS AND FUTURE WORK

We have presented a teleconferencing application that facilitates a natural conferencing environment through voice-activated audio, "video follows voice" switching and simple user interface. We have discussed the components of the application, and how it was built using the Programmers' Playground.

At the moment, the only option for allocation of source ids is random. As a result, floor control assumes that all participants of the conference are peers, since there is no scheme for priority allocation. We would like to use the system for other settings such as a remote classroom setup or a conference with a chairman and a particular ordering of the speakers. As a result, we would like to have more control over the allocation of the source ids. Since the conference module controls the assigning of source ids, this would require a small change in the module to accommodate a new method for allocating ids.

Another problem is the voice activated threshold, since this parameter must be set by the user and depends on the individual office setup and voice. Through experience we have discovered a good default setting that works for most setups. We plan to incorporate an automated mechanism for setting the threshold to a level just above the measured background noise.

Audio and video conferences are useful, but ideally one would like to have the use of other collaborative tools in a conference. We intend to introduce exhibits to the conferences that allow the use of a set of commercially available tools as well as collaborative tools built using the Programmers' Playground. Each exhibit would represent a different tool. Conferences would be allowed to have an arbitrary number of exhibits associated with them, and exhibits could be added and deleted from the conference dynamically. Individuals would only receive those exhibits they could support.

We are also working on creating an H.261 gateway to allow conferencing with low bandwidth sources.

Presently, the connection management system is centralized, so this limits the scalability of the system. This centralization occurs in both the Playground and the ATM connection management systems. In the next implementation of Playground, the connection management system will be distributed this will solve the problem in Playground. The ATM connection management system is centralized because we are using dynamic PVC's. By switching to SVC's, we should also be able to distribute the ATM system.

ACKNOWLEDGMENTS

We thank Bala Swaminathan and Ram Sethuraman for their work in developing the Playground library. This research was supported in part by the National Science Foundation under grants CCR-91-10029 and CCR-94-12711, and by the Advanced Research Projects Agency under contract DABT-63-95-C-0083.

REFERENCES

[1] Jerome R. Cox, Jr., Mike Gaddis, and Jonathan S. Turner. Project Zeus: Design of a Broadband Network and its Application on a University Campus. *IEEE Network*, pages 20-30, March 1993.

[2] Kenneth J. Goldman, et. al. "Welcome to the Programmers' Playground!" http://www.cs.wustl.edu/cs/playground/.

[3] Kenneth J. Goldman, T. Paul McCartney, Ram Sethuraman, and Bala Swaminathan. The Programmers' Playground: A Demonstration. In *Proceedings of the 1995 ACM International Conference on Multimedia*, November 1995. To appear. See also the conference CD-ROM proceedings for a longer version.

[4] Kenneth J. Goldman, T. Paul McCartney, Ram Sethuraman, Bala Swaminathan, and Todd Rodgers. Building Interactive Distributed Applications in C++ with The Programmers' Playground. Washington University Department of Computer Science technical report WUCS-95-20, July 1995.

[5] Kenneth J. Goldman, Bala Swaminathan, T. Paul McCartney, Michael D. Anderson, and Ram Sethuraman. The Programmers' Playground: I/O Abstraction for User-Configurable Distributed Applications. *IEEE Transactions on Software Engineering*. To appear.

[6] T. Paul McCartney, Kenneth J. Goldman, and David E. Saff. EUPHORIA: End-User Construction of Direct Manipulation User Interfaces for Distributed Applications. Washington University Department of Computer Science technical report WUCS-95-29, August 1995.

[7] William D. Richard, Jerome R. Cox, Jr., A. Maynard Engebretson, Jason Fritts, Brian . Gottlieb and Craig Horn. The Washington University MultiMedia eXplorer. Washington University Department of Computer Science technical report WUCS-93-40, August 1993.

[8] Mark J. Handley, Peter T. Kirstein, and M. Angela Sasse. Multimedia Integrated Conferencing for European Reseachers (MICE): Piloting Activities and the Conference Management and Multiplexing Centre