Architecture of HeiPhone:
A Testbed for Audio/Video Teleconferencing

Thomas Kaeppner and Lars Wolf

IBM European Networking Center
Vangerowstr. 18
D-6900 Heidelberg 1

{kaeppner, lwolf}@dhdibm1.bitnet

Keywords: Multimedia conferencing, audio transmission protocol, video transmission protocols, conference control protocols

Architecture of HeiPhone:
A Testbed for Audio/Video Teleconferencing

Summary

Due to recent advances in computer technology the integration of multimedia data with computing is becoming feasible. Such integration will allow for scenarios in which computer systems support collaborative conferencing. We have developed a testbed for multimedia teleconferencing that is used to study experimental protocols that can manage and organize multiparty conferences and transmit digital video and audio data. We distinguish between real-time communication, which is needed to support the transfer of continuous-media data, and non-real-time communication, which is used to exchange control messages such as invitations to a conference. These communication types are supported by different communication stacks within our testbed: Digital video and audio are transmitted via ST-II and HeiTP with media-specific protocols on top of them and control information for conference management is transmitted via remote procedure calls (RPC). An overview of the working system, its current communication mechanisms, and the services provided is the subject matter of this paper.

1 Introduction

1.1 Motivation

Due to recent advances in computer technology, high performance workstations with digital audio and video capabilities are becoming available. For the first time the integration of multimedia data with computing is becoming feasible. This integration will allow for scenarios in which computer systems support services such as video conferencing, news distribution, advertisement, and entertainment. Conferencing is a particularly important mechanism, since it is a prerequisite in any collaborative environment. We have built a working testbed in order to experiment with communication protocols for multiparty audio/video conferencing. An overview of the structure of the system, its current communication mechanisms, and the services provided is the subject matter of this paper.

1.2 Related Work

There are a number of multimedia projects that are investigating mechanisms for person-to-person conferences using shared text-oriented workspaces, graphics, audio, and video. Whereas [Lantz 86] and [Sarin and Greif 85] have studied conferencing architectures for text

and graphic, [Forsdick 85] and [Aguilar et al. 86] support audio conferencing. [Ludwig and Dunn 88], [Addeo et al. 88], and [Ahuja et al. 88] have built conferencing systems for video. However, in none of these systems was audio/video conferencing fully integrated with the workstation. Recently [Jeffay et al. 92] presented an integrated system built on existing protocols. Our work is targeted to providing an environment in which experimental protocols for both control of conferences and transmission of audio/video data can be evaluated.

The rest of this paper is organized as follows: The next section will briefly describe the hardware environment. In Section 3, we present an overview of the layered software architecture. Section 4 is concerned with conference control mechanisms in the upper layer. Section 5 describes the protocol used for communication between the layers and Section 6 presents structure and communication mechanisms in the lower layer of the system. Experience with the current system is described in Section 7, and finally Section 8 concludes.

2 Hardware Environment

Our hardware environment is illustrated in Figure 1. We are experimenting with HeiPhone on a network of IBM's RISC System/6000 workstations running under AIX Release 3.2.

M-Audio Capture and Playback Adapters (M-ACPA) which are built into the workstation are used to digitize sound and convert it back to analog form. The supported data types include 8-bit and 16-bit linear Pulse Code Modulation (PCM), μ-law companded, and A-law companded sample formats at 8k, 11k, 22k, and 44k samples per second. Using Adaptive Differential Pulse Code Modulation (ADPCM), the supported sampling modes consist of voice, music and music stereo.

Video boards manufactured by Rasterops allow the display and capturing of video images in a window.

Network connectivity is provided by a Token Ring that features a bandwidth of 16 Mbit/s and is used for transmission of both, continuous-media and traditional computer data. The underlying continuous-media transport system is also being ported to Ethernet.

3 Structure of the System

The challenge of multimedia systems lies in integrating the processing of traditional computer data, such as text and graphics, with processing of continuous-media data. Since continuous-

media data must obey strict time constraints, their processing requires real-time support [Hanko et al. 91]. On the other hand, operations that control the state of a conference or change the flow of data are not strictly time-bound but should provide a fast response. These different requirements have a strong impact on our model of the system.

As part of the HeiProjects [Herrtwich 92] we develop a system environment [Herrtwich and Wolf 92] that allows the processing of continuous-media data in real-time. This system environment is used by the conference testbed to implement all operations dealing with the flow of audio and video. Conference control and other computations are running on top of regular operating system mechanisms. This distinction leads to the two-layered structure that is depicted in Figure 2. The upper layer is referred to as the User Interface Agent (UIA), combining the functional modules of (1) controlling conferences and (2) providing the user interface. The lower layer is called the Audio-Visual Component (AVC) and handles the flow of continuous-media data within the system. By the Source and Sink Control Protocol (SSCP), implemented as Remote Procedure Calls (RPC), the UIA requests the AVC to establish data streams to other conference participants.

Different mechanisms are utilized for communication within each layer: Whereas conference control protocols are implemented as RPCs on top of TCP/IP, real time communication is supported by a stack of protocols that we refer to as the Heidelberg Transport System (HeiTS) [Hehmann et al. 91]. Exploiting different mechanisms for transport of control and audio/video data is not only motivated by differing service requirements, but also makes the system easier to extend and allows for an independent exchange of data transmission technology.

The following sections give more detailed explanations of each component.

4 The User Interface Agent

User Interface Agents are communicating via a Conference Control Protocol (CCP) in order to exchange state information and to manage conferences. Currently this management is fully distributed, i.e., there is no central entity of control. Moreover, UIAs provide the user with a graphical user interface in order to give him control over the conference. Since for experimenting with different conference control protocols the user interface must be adapted, these two functional blocks are interdependent and have been combined into one module.

4.1 Conference Model

The current CCP implements a model that we call generic conferencing. All participants have the same access rights during ongoing conferences. A conference is established by a user acting as an initiator. The initiator selects users in a phone book he wants to invite to a conference (see Figure 3). The initiator can also choose the types of media that he wants to communicate with. By clicking on a call-button he initiates the conference. Invitees are alerted by a window presenting information about the proposed conference and potential participants (see Figure 4). They can independently decide to either accept or reject the call. As soon as the first person has accepted, data streams, with types according to the selection of the initiator, are established between the participants. Further connections are established as invitees join the conference. Every participant can leave the conference without affecting communication between other members. The conference is closed when no other participants are left with whom to communicate.

4.2 Conference Control Protocol

Currently the CCP distinguishes only two message types as shown in Table 1. The *Invitation* message is used to deliver information about the proposed conference to all invitees. It contains the initiator's address and a time stamp, which together serve as a unique conference identifier for future messages belonging to this conference. A list of all invitees is included as supplemental information. Finally, the media which will be used during the conference are specified in detail.

The *State* message is used to distribute changes of state with regard to the conference to other participants. Apart from the conference identifier, the message contains the identification of the sender and the new state, to which the participant is advancing. Within the generic conferencing model we distinguish very few states (see Figure 5).

In order to avoid synchronization problems during state advances, we follow a simple policy by which changes of state are distributed to the initiator and all invitees whose state is not 'idle'. If *State* messages belong to a conference for which an invitation has not been received yet, they are stored in a database and their processing is postponed. When an invitation arrives, the database is searched and matching state messages are processed. Since state changes originating simultaneously at different sites can lead to inconsistencies and result in unwanted

growth of the data base, very old entries are deleted cyclically based on the time stamp information.

5 The Source and Sink Control Protocol

Communication between the User Interface Agent and the Audio-Visual Component (AVC) is achieved via the Source and Sink Control Protocol (SSCP). Within SSCP data streams are viewed as connecting a single source with potentially many sinks. One AVC manages all endpoints of data streams that are local to a single participant's workstation. SSCP[1] is endpoint-oriented, in the sense that all protocol primitives operate on these endpoints (see Table 2).

A UIA can request information about the media types that an AVC can support (via `ListEndpointAVTypes`), so that the selection, which is presented to the user, only includes supported media types. A data stream is established by issuing several requests:

- `OpenSink()` prepares the sink, so that data can flow from the network interface to appropriate output devices. If no sink is currently opened an initialization takes place, otherwise modules are started (e.g., Mixer) to handle an additional incoming data stream. The request returns address information that is necessary for establishment of audio/video connections.
- `OpenSource()` prepares the source, so that data can flow from appropriate input devices to the network interface.
- `AddSinks()` logically adds one or several sinks to the data stream originating at the specified source. This operation can be executed regardless whether the sinks are already connected.

The transmission of data over a specific data stream can be suspended and resumed by the `SetTransmission()` operation. With `GetAVOption()` the AVC is requested to report the actual settings for specified options, e.g., volume, treble, hue, saturation, etc. A UIA can manipulate these settings by issuing `SetAVOption()` operations.

State changes, which originate at the AVC mainly in error situations, are reported to the UIA by the events *EndpointClosed* and *SinkDropped*. That allows for recovery of failing connections without user interaction.

1. This protocol is a modified version of the SSCP used in [Altenhofen et al.].

6 The Audio Visual Component

The AVC handles the flow of audio and video data through the system. However, it has been designed not only to support SSCP, which is a protocol highly specialized to conferencing, but more generally to allow for flexible processing of audio/video data that can serve other multimedia applications as well.

6.1 Architecture of AVC

The AVC consists of a set of stream handlers each capable of executing a specific task related to the processing of continuous-media data. A Control module provides the SSCP-interface for the upper layer. The set of stream handlers consists of: Filer, Audio-device, Video-device, Network, and Mixer.

The Control module instantiates these stream handlers when they are needed. By means of a message based protocol, the behavior of stream handlers is steered. The Control module can logically connect stream handlers to a sequence, thus setting up the flow of data. (See Figure 6, which illustrates the stream handler structure at one site during 3-way audio conferencing.)

The Filer allows to store and retrieve continuous-media data on disk. A description of the data is stored in a supplemental file for use by editors or viewers.

The Audio-device and Video-device manage input and output on the respective adapters. They make data types and functionality of the adapter accessible by the Control module.

The Mixer combines several audio streams into one by digitally mixing audio samples. Different mixing strategies are provided for conferencing situations and mixing of data read from disk, in order to optimize the delay and the synchronization of media streams, respectively.

The Network stream handler allows the establishment of network connections via HeiTS. For every connection, the Network stream handler instantiates either a sender or a receiver stream handler. These handle data packets that are sent and received by HeiTS, whereas control packets of HeiTS are passed through the Network stream handler itself.

6.2 Continuous-Media Transport System HeiTS

We exploit HeiTS for transmission of time-critical audio and video data. Coexisting with TCP/IP, HeiTS enables to exchange streams of data with quality of service (QoS) guarantees. Users of the transport system specify their QoS requirements with regard to the transmission. Service is provided by a stack of protocols consisting of HeiTP (Heidelberg Transport Protocol)

[Delgrossi et al. 92], ST-II [Topolcic 90], and HeiDL (Heidelberg Data Link) [Twachtmann 92]. HeiDL is a data link access method, which interfaces several networks. On top of it ST-II, as a connection-oriented network layer protocol negotiates resource reservations and transmits continuous-media data in a stream-oriented fashion. HeiTP provides error correction and fragmentation/reassembly functionality.

Most parts of HeiTS are implemented in user space to ease porting. Currently HeiTS has been implemented for AIX 3 and OS/2. In order to achieve real-time characteristics, HeiTS, as well as the stream handlers introduced, make use of the real-time environment and resource management that is described in the next section.

6.3 Real-time Support and Supplemental Services

Guaranteeing real-time characteristics requires knowledge and control about the amount of resources needed for certain operations. The Heidelberg Resource Administration Technique (HeiRAT) [Vogt 92] contains the mechanisms to calculate requirements of streams with regard to buffer space, CPU time, and network bandwidth during their establishment phase. Based on available resources HeiRAT either admits or rejects the establishment of new streams. When a new stream is admitted, the resources are reserved and then scheduled according to HeiRAT's scheduling mechanisms. Scheduling utilizes a rate-monotonic algorithm [Liu and Layland 73], which assigns fixed priorities, except for necessary shifts, that can be caused by the establishment of new connections. Thus, the overhead of priority determination and setting is avoided.

The CPU scheduler is built on top of basic real-time capabilities offered by the AIX 3 operating system. This operating system supports fixed priorities which (1) are higher than the one of the standard AIX scheduler and (2) will not be changed by the standard AIX scheduler.

In order to avoid unpredictable processing delays caused by paging operations, all memory that is used during the processing of continuous-media data is 'pinned' in physical memory.

Data movement operations are reduced to a minimum by a buffer management system, i.e., stream handlers exchange pointers to a common data pool rather than the actual data when data is flows from one stream handler to the next.

6.4 Audio Processing

Since the streams of audio form a fully connected network between the *n* participants of the conference, every member receives *n-1* data streams which must be mixed together. Compared

to a situation in which mixing is performed at a central location, this distributed mixing generally causes a shorter execution path since packets are transferred directly to the endsystem. Whereas centralized mixing requires fewer network connections than distributed mixing, the latter not only gives the capability to independently control the volume of any received data stream, but also leads to a straightforward integration of local multimedia applications that use the audio adapter simultaneously.

6.5 Video Processing

The Video-device stream handler is responsible for capturing and displaying frames. Color depth and size of frames are set by the user. The full-size of the displayed video is captured and sent to all conference participants. Since at present we perform no compression of the image, frames are generally larger than the maximum transmission unit of the network. Thus, frames must be split into fragments before they can be handed to the network interface.

The current version of HeiTS provides reliable service by retransmission of lost packets. Since retransmissions can violate the time constraints of audio and video data, QoS parameters are chosen so that HeiTS does not provide reliable service.

A simple video protocol is implemented by adding a header to the video data containing additional information: A fragment counter contains the sequence number of the fragment, relative to the beginning of the frame. A frame counter denotes the sequence number of the frame since connection establishment. A flag reports, whether the received fragment is the last fragment belonging to a frame.

Using the video header, the display algorithm presents a frame when either the last fragment of a frame has been received or a fragment has been received with a frame number higher than those of previously received fragments. The video data is copied into a display buffer at a location corresponding to its fragment counter. Since only one display buffer is used to store all received fragments, data of lost fragments is replaced by data that has been presented at the same location within the most recent frame.

7 Experience and Work in Progress

Data streams of audio and video are flowing separately during ongoing conferences because (1) in a more general model of conferencing than the current implemented one, audio and video streams need not flow to the same targets, and more important (2) separate streams allow

to prioritize audio processing over video, which we have found very important for user acceptance.

We did not have to prioritize audio over video, but experimenting with conditions, in which no QoS guarantees could be given, we have experienced overload situations, in which an increasing backlog within the sequence of receiving network stream handler and video stream handler leads to a shortage of buffer space. Since video buffers can only be allocated up to the reserved maximum, the audio stream is left unaffected. Moreover the shortage gives chance to recover, since the load is not increased any further. This leads to a down-scaling of the video frame rate in overload situations.

The current implementation of the HeiPhone testbed is used within our own working group and has been presented at several forums and fairs (e.g., COMDEX Fall '92). User feedback has been overwhelming positive, the main criticism demanding more flexibility in adding and releasing video during ongoing conferences. The high-quality audio offered by the system was highly appreciated. Although the delay is very acceptable and not noticed during conversations, we know that some of it is caused by pre-buffering on the audio adapter. We have optimized the packet size of audio with regard to this prebuffering, which lead to packets containing 45 ms of sound.

Currently, we are using the testbed to implement several experiments:

- The Conference Control Protocol is extended such that media types can be flexibly changed during ongoing conferences, allowing for asymmetric participation with regard to the media types used. The enhanced model of conferences enables to switch between moderated conferences, in which a chairperson passes the floor of a conversation, to simple conferences, in which floor passing is managed by social protocols.
- The protocol for audio is extended in order to implement silence detection. Silence detection has the capability to reduce the amount of transmitted data, however sometimes background noise of the listening party is considered as a desirable feedback. We are particularly interested in how silence detection can be used to suppress disturbing echos which can be caused by the combination of regular speakers and microphones.
- Time stamp information is included in both the audio and video protocol. A stream handler, that can synchronize audio and video based on that information is being implemented.

8 Conclusion

Conferencing is a particularly important mechanism, since it is a prerequisite in any collaborative environment. We have designed and implemented a testbed in order to experiment with communication protocols for multiparty audio/video conferencing. We distinguish between real-time communication, which is needed to support the transfer of continuous-media data, and non-real-time communication, which is used to exchange control messages such as invitations to a conference. The testbed has a layered structure dividing the system into a real-time supported layer for the handling of audio/video data and a layer which is concerned with conference management. The communication needs of the different layers are supported by different communication stacks. Using the testbed we have implemented a simple conference model that enabled us to achieve full integration of multimedia conferencing with computing. We have designed and implemented experimental protocols for the transmission of audio and video data and gained first experience in user trials.

Acknowlegements

We would like to thank Andreas Schröer for implementing large parts of the AVC, Ralf Guido Herrtwich for implementing the user interface, and Dietmar Hehmann for implementing the Video-device stream handler.

References

[Addeo et al. 88] E.J. Addeo, A.B. Dayao, A.D. Gelman, V.F. Massa: An Experimental Multi-Media Bridging System. Bell Communications Research, 1988.

[Aguilar et al. 86] L. Aguilar, J.J. Garcia-Luna-Aceves, D. Moran, E.J. Craighill, R. Brungardt: Architecture for a Multi-Media Tele-Conferencing System, ACM SIGCOMM, August 1986.

[Ahuja et al. 88] S.R. Ahuja, J. Ensor, D. Horn: The Rapport Multimedia Conferencing System. Proceedings of the Conference on Office Information Systems, March 1988.

[Altenhofen et al. 93] M. Altenhofen, J. Dittrich, R. Hammerschmidt, R.G. Herrtwich, T. Kaeppner, C. Kruschel, A. Kückes, T. Steinig: The BERKOM Multimedia Collaboration Service. Submitted to First ACM International Conference on Multimedia ‘93.

[Delgrossi et al. 92] L. Delgrossi, C. Halstrick, R.G. Herrtwich, H. Stüttgen: HeiTP: A

Transport Protocol for ST-II. Proceedings of GLOBECOM' 92, December 1992, Orlando, Florida.

[Forsdick 85] H.C. Forsdick: Explorations in Real-Time Multi-Media Conferencing. Proceedings of the Second International Symposium on Computer Message Systems, IFIP, September, 1985.

[Hanko et al. 91] J.G. Hanko, E.G. Kuerner, J.D. Northcutt, G.A. Wall: Workstation Support for Time-Critical Applications. Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 18-19, 1991, Heidelberg.

[Hehmann et al. 91] D. Hehmann, R.G. Herrtwich, W. Schulz, T. Schütt, R. Steinmetz: Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High-Speed Transport System. Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 18-19, 1991, Heidelberg.

[Herrtwich 92] R.G. Herrtwich: The HeiProjects: Support for Distributed Multimedia Applications. IBM ENC Heidelberg, Technical Report No. 43.9206.

[Herrtwich and Wolf 92] R.G. Herrtwich, L.C. Wolf: A System Software Structure for Distributed Multimedia Systems. Proceedings of the Fifth ACM SIGOPS European Workshop, September 21-23, 1992, Le Mont Saint-Michel, France.

[Jeffay et al. 92] K. Jeffay, D.L. Stone, T. Talley, F.D. Smith: Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks. Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, November 12-13, 1992, San Diego, California.

[Lantz 86] K.A. Lantz: An Experiment in Integrated Multimedia Conferencing. Olivetti Research Center, Menlo Park, CA, December 1986.

[Liu and Layland 73] C.L. Liu, J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, Vol. 20, No. 1, 1973, pp. 46-61.

[Ludwig and Dunn 88] Laboratory for Emulation and Study of Integrated and Coordinated Media Communication. ACM SIGCOMM, August 1988.

[Sarin and Greif 85] S. Sarin, I. Greif: Computer-Based Real-Time Conferences. IEEE Computer, October 1985.

[Topolcic 90] C. Topolcic (Ed.): Experimental Internet Stream Protocol, Version 2 (ST-II). Internet Request for Comments 1190, Network Working Group, October 1990.

[Twachtmann 92] B. Twachtmann: The Heidelberg Transport System: Data Link Layer. IBM ENC Heidelberg, Internal working paper, October 1992.

[Vogt 92] C. Vogt: The Heidelberg Resource Administration Technique: Design Fundamentals, QoS Calculation, and Resource Reservation. IBM European Networking Center Heidelberg, Technical Report No. 43.9213, October 1992. Also: to be presented at Kommunikation in verteilten Systemen. March, 3-5 1993, Munich.

Figures

Figure 1: Hardware Environment.

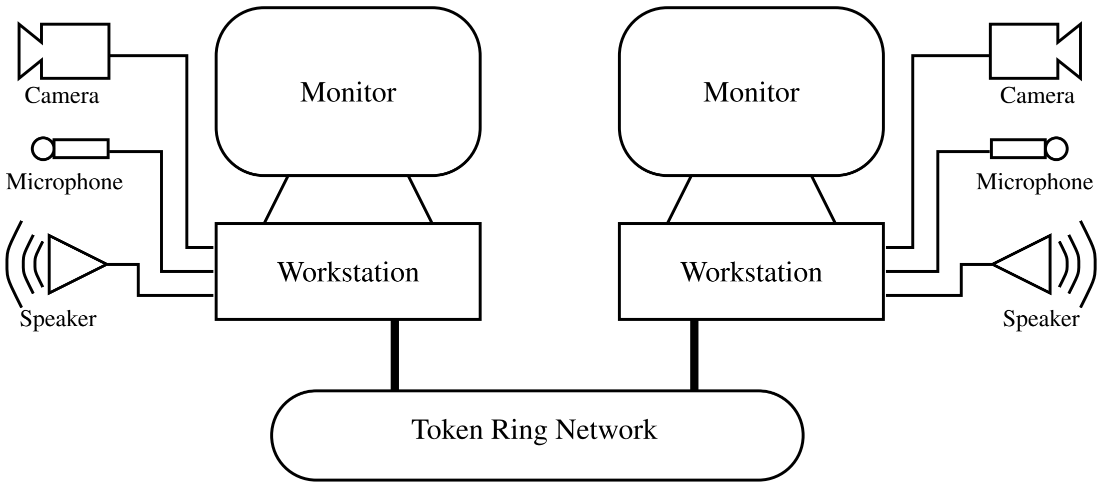


Figure 2: Separation of User Interface and Multimedia Handling

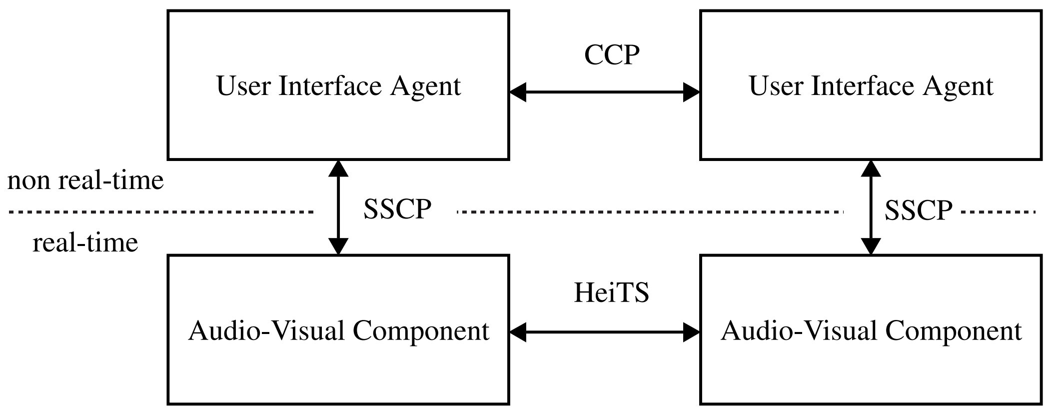


Figure 3: HeiPhone Main Window — Users Phone Book on the Left Side

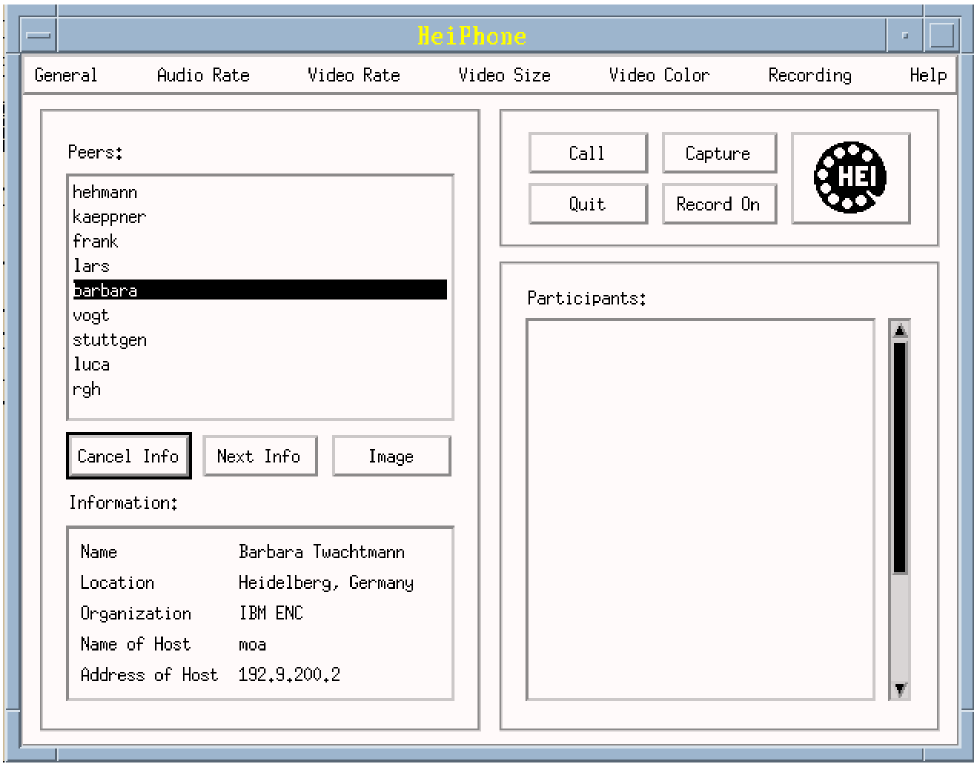


Figure 4: Invitation to a Conference

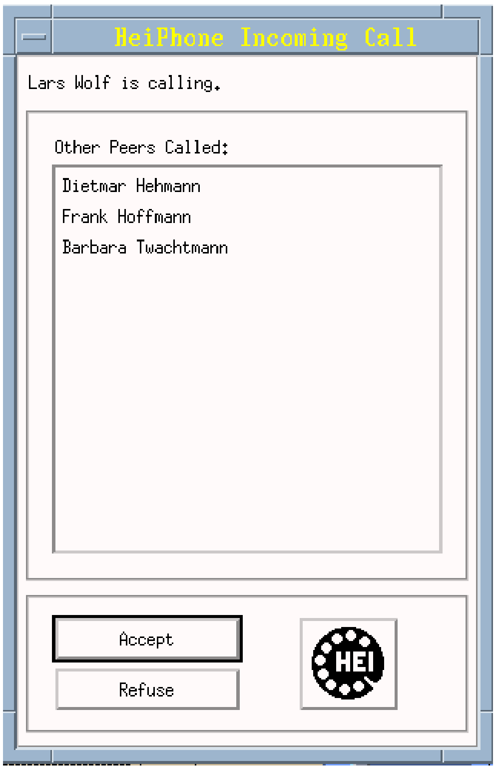


Figure 5: Conferencing States

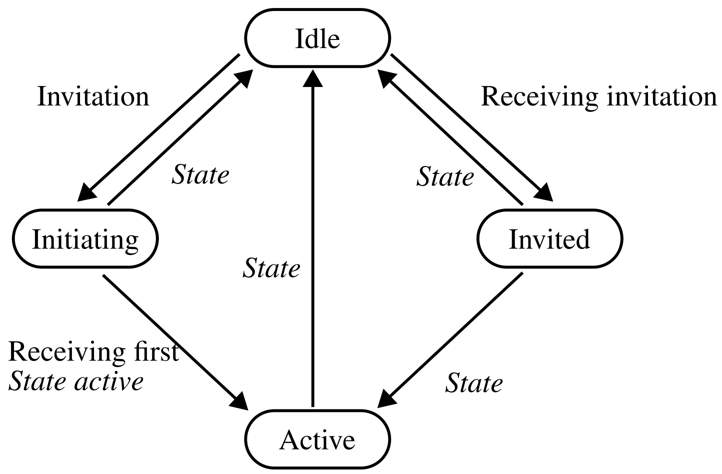
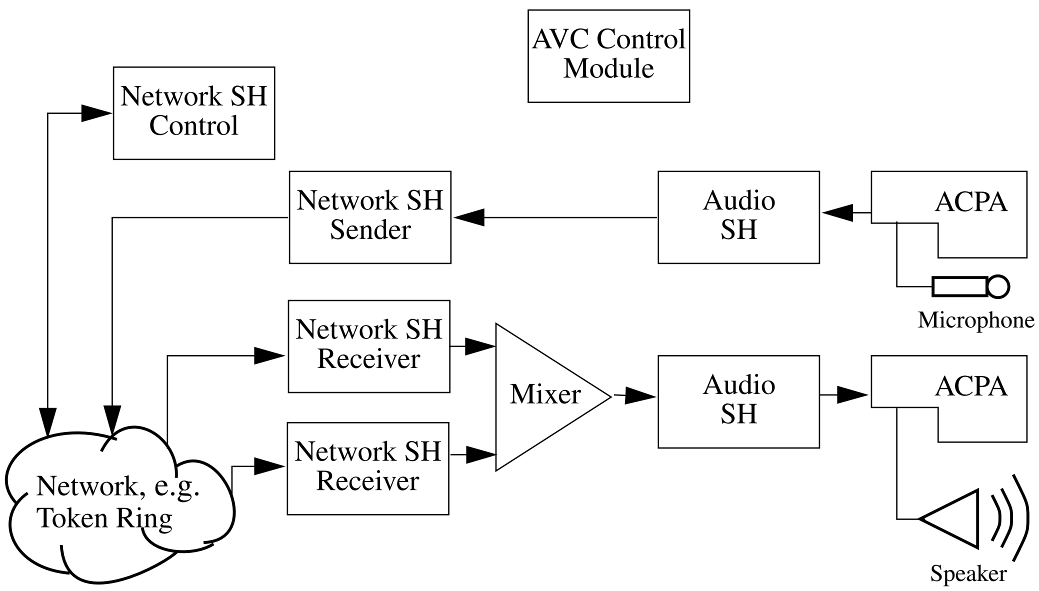


Figure 6: Structure of stream handlers during 3-way audio conferencing



Tables

Table 1: Conference Control Protocol Primitives

Primitive	Content
Invitation	Initiator Time stamp List of Invitees
State	Initiator Time stamp Participant New state

Table 2: Source and Sink Control Protocol Primitives

Request	Parameters	Return Values
ListEndpointAVTypes	EndpointType AVMode	AVTypeList
OpenSource	AVType	Endpoint
OpenSink	AVType	Endpoint
CloseEndpoint	Endpoint	none
AddSinks	SourceEndpoint SinkEndpointList	none
RemoveSinks	SourceEndpoint SinkEndpointList	none
SetTransmission	SourceEndpoint Mode	none
GetAVOption	Option	OptionValue
SetAVOption	Option OptionValue	none

Event	Parameters
EndpointClosed	Endpoint
SinkDropped	SourceEndpoint SinkEndpoint Reason

Architecture of HeiPhone: A Testbed for Audio/Video Teleconferencing

Thomas Kaeppner and Lars Wolf

IBM European Networking Center
Vangerowstr. 18
D-6900 Heidelberg 1

{kaeppner, lwolf}@dhdibm1.bitnet

Keywords: Multimedia conferencing, audio transmission protocol, video transmission protocols, conference control protocols

Architecture of HeiPhone:
A Testbed for Audio/Video Teleconferencing

Summary

Due to recent advances in computer technology the integration of multimedia data with computing is becoming feasible. Such integration will allow for scenarios in which computer systems support collaborative conferencing. We have developed a testbed for multimedia teleconferencing that is used to study experimental protocols that can manage and organize multiparty conferences and transmit digital video and audio data. We distinguish between real-time communication, which is needed to support the transfer of continuous-media data, and non-real-time communication, which is used to exchange control messages such as invitations to a conference. These communication types are supported by different communication stacks within our testbed: Digital video and audio are transmitted via ST-II and HeiTP with media-specific protocols on top of them and control information for conference management is transmitted via remote procedure calls (RPC). An overview of the working system, its current communication mechanisms, and the services provided is the subject matter of this paper.

1 Introduction

1.1 Motivation

Due to recent advances in computer technology, high performance workstations with digital audio and video capabilities are becoming available. For the first time the integration of multimedia data with computing is becoming feasible. This integration will allow for scenarios in which computer systems support services such as video conferencing, news distribution, advertisement, and entertainment. Conferencing is a particularly important mechanism, since it is a prerequisite in any collaborative environment. We have built a working testbed in order to experiment with communication protocols for multiparty audio/video conferencing. An overview of the structure of the system, its current communication mechanisms, and the services provided is the subject matter of this paper.

1.2 Related Work

There are a number of multimedia projects that are investigating mechanisms for person-to-person conferences using shared text-oriented workspaces, graphics, audio, and video. Whereas [Lantz 86] and [Sarin and Greif 85] have studied conferencing architectures for text

and graphic, [Forsdick 85] and [Aguilar et al. 86] support audio conferencing. [Ludwig and Dunn 88], [Addeo et al. 88], and [Ahuja et al. 88] have built conferencing systems for video. However, in none of these systems was audio/video conferencing fully integrated with the workstation. Recently [Jeffay et al. 92] presented an integrated system built on existing protocols. Our work is targeted to providing an environment in which experimental protocols for both control of conferences and transmission of audio/video data can be evaluated.

The rest of this paper is organized as follows: The next section will briefly describe the hardware environment. In Section 3, we present an overview of the layered software architecture. Section 4 is concerned with conference control mechanisms in the upper layer. Section 5 describes the protocol used for communication between the layers and Section 6 presents structure and communication mechanisms in the lower layer of the system. Experience with the current system is described in Section 7, and finally Section 8 concludes.

2 Hardware Environment

Our hardware environment is illustrated in Figure 1. We are experimenting with HeiPhone on a network of IBM's RISC System/6000 workstations running under AIX Release 3.2.

M-Audio Capture and Playback Adapters (M-ACPA) which are built into the workstation are used to digitize sound and convert it back to analog form. The supported data types include 8-bit and 16-bit linear Pulse Code Modulation (PCM), μ-law companded, and A-law companded sample formats at 8k, 11k, 22k, and 44k samples per second. Using Adaptive Differential Pulse Code Modulation (ADPCM), the supported sampling modes consist of voice, music and music stereo.

Video boards manufactured by Rasterops allow the display and capturing of video images in a window.

Network connectivity is provided by a Token Ring that features a bandwidth of 16 Mbit/s and is used for transmission of both, continuous-media and traditional computer data. The underlying continuous-media transport system is also being ported to Ethernet.

3 Structure of the System

The challenge of multimedia systems lies in integrating the processing of traditional computer data, such as text and graphics, with processing of continuous-media data. Since continuous-

media data must obey strict time constraints, their processing requires real-time support [Hanko et al. 91]. On the other hand, operations that control the state of a conference or change the flow of data are not strictly time-bound but should provide a fast response. These different requirements have a strong impact on our model of the system.

As part of the HeiProjects [Herrtwich 92] we develop a system environment [Herrtwich and Wolf 92] that allows the processing of continuous-media data in real-time. This system environment is used by the conference testbed to implement all operations dealing with the flow of audio and video. Conference control and other computations are running on top of regular operating system mechanisms. This distinction leads to the two-layered structure that is depicted in Figure 2. The upper layer is referred to as the User Interface Agent (UIA), combining the functional modules of (1) controlling conferences and (2) providing the user interface. The lower layer is called the Audio-Visual Component (AVC) and handles the flow of continuous-media data within the system. By the Source and Sink Control Protocol (SSCP), implemented as Remote Procedure Calls (RPC), the UIA requests the AVC to establish data streams to other conference participants.

Different mechanisms are utilized for communication within each layer: Whereas conference control protocols are implemented as RPCs on top of TCP/IP, real time communication is supported by a stack of protocols that we refer to as the Heidelberg Transport System (HeiTS) [Hehmann et al. 91]. Exploiting different mechanisms for transport of control and audio/video data is not only motivated by differing service requirements, but also makes the system easier to extend and allows for an independent exchange of data transmission technology.

The following sections give more detailed explanations of each component.

4 The User Interface Agent

User Interface Agents are communicating via a Conference Control Protocol (CCP) in order to exchange state information and to manage conferences. Currently this management is fully distributed, i.e., there is no central entity of control. Moreover, UIAs provide the user with a graphical user interface in order to give him control over the conference. Since for experimenting with different conference control protocols the user interface must be adapted, these two functional blocks are interdependent and have been combined into one module.

4.1 Conference Model

The current CCP implements a model that we call generic conferencing. All participants have the same access rights during ongoing conferences. A conference is established by a user acting as an initiator. The initiator selects users in a phone book he wants to invite to a conference (see Figure 3). The initiator can also choose the types of media that he wants to communicate with. By clicking on a call-button he initiates the conference. Invitees are alerted by a window presenting information about the proposed conference and potential participants (see Figure 4). They can independently decide to either accept or reject the call. As soon as the first person has accepted, data streams, with types according to the selection of the initiator, are established between the participants. Further connections are established as invitees join the conference. Every participant can leave the conference without affecting communication between other members. The conference is closed when no other participants are left with whom to communicate.

4.2 Conference Control Protocol

Currently the CCP distinguishes only two message types as shown in Table 1. The *Invitation* message is used to deliver information about the proposed conference to all invitees. It contains the initiator's address and a time stamp, which together serve as a unique conference identifier for future messages belonging to this conference. A list of all invitees is included as supplemental information. Finally, the media which will be used during the conference are specified in detail.

The *State* message is used to distribute changes of state with regard to the conference to other participants. Apart from the conference identifier, the message contains the identification of the sender and the new state, to which the participant is advancing. Within the generic conferencing model we distinguish very few states (see Figure 5).

In order to avoid synchronization problems during state advances, we follow a simple policy by which changes of state are distributed to the initiator and all invitees whose state is not 'idle'. If *State* messages belong to a conference for which an invitation has not been received yet, they are stored in a database and their processing is postponed. When an invitation arrives, the database is searched and matching state messages are processed. Since state changes originating simultaneously at different sites can lead to inconsistencies and result in unwanted

growth of the data base, very old entries are deleted cyclically based on the time stamp information.

5 The Source and Sink Control Protocol

Communication between the User Interface Agent and the Audio-Visual Component (AVC) is achieved via the Source and Sink Control Protocol (SSCP). Within SSCP data streams are viewed as connecting a single source with potentially many sinks. One AVC manages all endpoints of data streams that are local to a single participant's workstation. SSCP[1] is endpoint-oriented, in the sense that all protocol primitives operate on these endpoints (see Table 2).

A UIA can request information about the media types that an AVC can support (via `ListEndpointAVTypes`), so that the selection, which is presented to the user, only includes supported media types. A data stream is established by issuing several requests:

- `OpenSink`() prepares the sink, so that data can flow from the network interface to appropriate output devices. If no sink is currently opened an initialization takes place, otherwise modules are started (e.g., Mixer) to handle an additional incoming data stream. The request returns address information that is necessary for establishment of audio/video connections.
- `OpenSource`() prepares the source, so that data can flow from appropriate input devices to the network interface.
- `AddSinks`() logically adds one or several sinks to the data stream originating at the specified source. This operation can be executed regardless whether the sinks are already connected.

The transmission of data over a specific data stream can be suspended and resumed by the `SetTransmission`() operation. With `GetAVOption`() the AVC is requested to report the actual settings for specified options, e.g., volume, treble, hue, saturation, etc. A UIA can manipulate these settings by issuing `SetAVOption`() operations.

State changes, which originate at the AVC mainly in error situations, are reported to the UIA by the events *EndpointClosed* and *SinkDropped*. That allows for recovery of failing connections without user interaction.

1. This protocol is a modified version of the SSCP used in [Altenhofen et al.].

6 The Audio Visual Component

The AVC handles the flow of audio and video data through the system. However, it has been designed not only to support SSCP, which is a protocol highly specialized to conferencing, but more generally to allow for flexible processing of audio/video data that can serve other multimedia applications as well.

6.1 Architecture of AVC

The AVC consists of a set of stream handlers each capable of executing a specific task related to the processing of continuous-media data. A Control module provides the SSCP-interface for the upper layer. The set of stream handlers consists of: Filer, Audio-device, Video-device, Network, and Mixer.

The Control module instantiates these stream handlers when they are needed. By means of a message based protocol, the behavior of stream handlers is steered. The Control module can logically connect stream handlers to a sequence, thus setting up the flow of data. (See Figure 6, which illustrates the stream handler structure at one site during 3-way audio conferencing.)

The Filer allows to store and retrieve continuous-media data on disk. A description of the data is stored in a supplemental file for use by editors or viewers.

The Audio-device and Video-device manage input and output on the respective adapters. They make data types and functionality of the adapter accessible by the Control module.

The Mixer combines several audio streams into one by digitally mixing audio samples. Different mixing strategies are provided for conferencing situations and mixing of data read from disk, in order to optimize the delay and the synchronization of media streams, respectively.

The Network stream handler allows the establishment of network connections via HeiTS. For every connection, the Network stream handler instantiates either a sender or a receiver stream handler. These handle data packets that are sent and received by HeiTS, whereas control packets of HeiTS are passed through the Network stream handler itself.

6.2 Continuous-Media Transport System HeiTS

We exploit HeiTS for transmission of time-critical audio and video data. Coexisting with TCP/IP, HeiTS enables to exchange streams of data with quality of service (QoS) guarantees. Users of the transport system specify their QoS requirements with regard to the transmission. Service is provided by a stack of protocols consisting of HeiTP (Heidelberg Transport Protocol)

[Delgrossi et al. 92], ST-II [Topolcic 90], and HeiDL (Heidelberg Data Link) [Twachtmann 92]. HeiDL is a data link access method, which interfaces several networks. On top of it ST-II, as a connection-oriented network layer protocol negotiates resource reservations and transmits continuous-media data in a stream-oriented fashion. HeiTP provides error correction and fragmentation/reassembly functionality.

Most parts of HeiTS are implemented in user space to ease porting. Currently HeiTS has been implemented for AIX 3 and OS/2. In order to achieve real-time characteristics, HeiTS, as well as the stream handlers introduced, make use of the real-time environment and resource management that is described in the next section.

6.3 Real-time Support and Supplemental Services

Guaranteeing real-time characteristics requires knowledge and control about the amount of resources needed for certain operations. The Heidelberg Resource Administration Technique (HeiRAT) [Vogt 92] contains the mechanisms to calculate requirements of streams with regard to buffer space, CPU time, and network bandwidth during their establishment phase. Based on available resources HeiRAT either admits or rejects the establishment of new streams. When a new stream is admitted, the resources are reserved and then scheduled according to HeiRAT's scheduling mechanisms. Scheduling utilizes a rate-monotonic algorithm [Liu and Layland 73], which assigns fixed priorities, except for necessary shifts, that can be caused by the establishment of new connections. Thus, the overhead of priority determination and setting is avoided.

The CPU scheduler is built on top of basic real-time capabilities offered by the AIX 3 operating system. This operating system supports fixed priorities which (1) are higher than the one of the standard AIX scheduler and (2) will not be changed by the standard AIX scheduler.

In order to avoid unpredictable processing delays caused by paging operations, all memory that is used during the processing of continuous-media data is 'pinned' in physical memory.

Data movement operations are reduced to a minimum by a buffer management system, i.e., stream handlers exchange pointers to a common data pool rather than the actual data when data is flows from one stream handler to the next.

6.4 Audio Processing

Since the streams of audio form a fully connected network between the *n* participants of the conference, every member receives *n-1* data streams which must be mixed together. Compared

to a situation in which mixing is performed at a central location, this distributed mixing generally causes a shorter execution path since packets are transferred directly to the endsystem. Whereas centralized mixing requires fewer network connections than distributed mixing, the latter not only gives the capability to independently control the volume of any received data stream, but also leads to a straightforward integration of local multimedia applications that use the audio adapter simultaneously.

6.5 Video Processing

The Video-device stream handler is responsible for capturing and displaying frames. Color depth and size of frames are set by the user. The full-size of the displayed video is captured and sent to all conference participants. Since at present we perform no compression of the image, frames are generally larger than the maximum transmission unit of the network. Thus, frames must be split into fragments before they can be handed to the network interface.

The current version of HeiTS provides reliable service by retransmission of lost packets. Since retransmissions can violate the time constraints of audio and video data, QoS parameters are chosen so that HeiTS does not provide reliable service.

A simple video protocol is implemented by adding a header to the video data containing additional information: A fragment counter contains the sequence number of the fragment, relative to the beginning of the frame. A frame counter denotes the sequence number of the frame since connection establishment. A flag reports, whether the received fragment is the last fragment belonging to a frame.

Using the video header, the display algorithm presents a frame when either the last fragment of a frame has been received or a fragment has been received with a frame number higher than those of previously received fragments. The video data is copied into a display buffer at a location corresponding to its fragment counter. Since only one display buffer is used to store all received fragments, data of lost fragments is replaced by data that has been presented at the same location within the most recent frame.

7 Experience and Work in Progress

Data streams of audio and video are flowing separately during ongoing conferences because (1) in a more general model of conferencing than the current implemented one, audio and video streams need not flow to the same targets, and more important (2) separate streams allow

to prioritize audio processing over video, which we have found very important for user acceptance.

We did not have to prioritize audio over video, but experimenting with conditions, in which no QoS guarantees could be given, we have experienced overload situations, in which an increasing backlog within the sequence of receiving network stream handler and video stream handler leads to a shortage of buffer space. Since video buffers can only be allocated up to the reserved maximum, the audio stream is left unaffected. Moreover the shortage gives chance to recover, since the load is not increased any further. This leads to a down-scaling of the video frame rate in overload situations.

The current implementation of the HeiPhone testbed is used within our own working group and has been presented at several forums and fairs (e.g., COMDEX Fall '92). User feedback has been overwhelming positive, the main criticism demanding more flexibility in adding and releasing video during ongoing conferences. The high-quality audio offered by the system was highly appreciated. Although the delay is very acceptable and not noticed during conversations, we know that some of it is caused by pre-buffering on the audio adapter. We have optimized the packet size of audio with regard to this prebuffering, which lead to packets containing 45 ms of sound.

Currently, we are using the testbed to implement several experiments:

- The Conference Control Protocol is extended such that media types can be flexibly changed during ongoing conferences, allowing for asymmetric participation with regard to the media types used. The enhanced model of conferences enables to switch between moderated conferences, in which a chairperson passes the floor of a conversation, to simple conferences, in which floor passing is managed by social protocols.
- The protocol for audio is extended in order to implement silence detection. Silence detection has the capability to reduce the amount of transmitted data, however sometimes background noise of the listening party is considered as a desirable feedback. We are particularly interested in how silence detection can be used to suppress disturbing echos which can be caused by the combination of regular speakers and microphones.
- Time stamp information is included in both the audio and video protocol. A stream handler, that can synchronize audio and video based on that information is being implemented.

8 Conclusion

Conferencing is a particularly important mechanism, since it is a prerequisite in any collaborative environment. We have designed and implemented a testbed in order to experiment with communication protocols for multiparty audio/video conferencing. We distinguish between real-time communication, which is needed to support the transfer of continuous-media data, and non-real-time communication, which is used to exchange control messages such as invitations to a conference. The testbed has a layered structure dividing the system into a real-time supported layer for the handling of audio/video data and a layer which is concerned with conference management. The communication needs of the different layers are supported by different communication stacks. Using the testbed we have implemented a simple conference model that enabled us to achieve full integration of multimedia conferencing with computing. We have designed and implemented experimental protocols for the transmission of audio and video data and gained first experience in user trials.

Acknowlegements

We would like to thank Andreas Schröer for implementing large parts of the AVC, Ralf Guido Herrtwich for implementing the user interface, and Dietmar Hehmann for implementing the Video-device stream handler.

References

[Addeo et al. 88] E.J. Addeo, A.B. Dayao, A.D. Gelman, V.F. Massa: An Experimental Multi-Media Bridging System. Bell Communications Research, 1988.

[Aguilar et al. 86] L. Aguilar, J.J. Garcia-Luna-Aceves, D. Moran, E.J. Craighill, R. Brungardt: Architecture for a Multi-Media Tele-Conferencing System, ACM SIGCOMM, August 1986.

[Ahuja et al. 88] S.R. Ahuja, J. Ensor, D. Horn: The Rapport Multimedia Conferencing System. Proceedings of the Conference on Office Information Systems, March 1988.

[Altenhofen et al. 93] M. Altenhofen, J. Dittrich, R. Hammerschmidt, R.G. Herrtwich, T. Kaeppner, C. Kruschel, A. Kückes, T. Steinig: The BERKOM Multimedia Collaboration Service. Submitted to First ACM International Conference on Multimedia '93.

[Delgrossi et al. 92] L. Delgrossi, C. Halstrick, R.G. Herrtwich, H. Stüttgen: HeiTP: A

Transport Protocol for ST-II. Proceedings of GLOBECOM' 92, December 1992, Orlando, Florida.

[Forsdick 85] H.C. Forsdick: Explorations in Real-Time Multi-Media Conferencing. Proceedings of the Second International Symposium on Computer Message Systems, IFIP, September, 1985.

[Hanko et al. 91] J.G. Hanko, E.G. Kuerner, J.D. Northcutt, G.A. Wall: Workstation Support for Time-Critical Applications. Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 18-19, 1991, Heidelberg.

[Hehmann et al. 91] D. Hehmann, R.G. Herrtwich, W. Schulz, T. Schütt, R. Steinmetz: Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High-Speed Transport System. Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 18-19, 1991, Heidelberg.

[Herrtwich 92] R.G. Herrtwich: The HeiProjects: Support for Distributed Multimedia Applications. IBM ENC Heidelberg, Technical Report No. 43.9206.

[Herrtwich and Wolf 92] R.G. Herrtwich, L.C. Wolf: A System Software Structure for Distributed Multimedia Systems. Proceedings of the Fifth ACM SIGOPS European Workshop, September 21-23, 1992, Le Mont Saint-Michel, France.

[Jeffay et al. 92] K. Jeffay, D.L. Stone, T. Talley, F.D. Smith: Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks. Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, November 12-13, 1992, San Diego, California.

[Lantz 86] K.A. Lantz: An Experiment in Integrated Multimedia Conferencing. Olivetti Research Center, Menlo Park, CA, December 1986.

[Liu and Layland 73] C.L. Liu, J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, Vol. 20, No. 1, 1973, pp. 46-61.

[Ludwig and Dunn 88] Laboratory for Emulation and Study of Integrated and Coordinated Media Communication. ACM SIGCOMM, August 1988.

[Sarin and Greif 85] S. Sarin, I. Greif: Computer-Based Real-Time Conferences. IEEE Computer, October 1985.

[Topolcic 90] C. Topolcic (Ed.): Experimental Internet Stream Protocol, Version 2 (ST-II). Internet Request for Comments 1190, Network Working Group, October 1990.

[Twachtmann 92] B. Twachtmann: The Heidelberg Transport System: Data Link Layer. IBM ENC Heidelberg, Internal working paper, October 1992.

[Vogt 92] C. Vogt: The Heidelberg Resource Administration Technique: Design Fundamentals, QoS Calculation, and Resource Reservation. IBM European Networking Center Heidelberg, Technical Report No. 43.9213, October 1992. Also: to be presented at Kommunikation in verteilten Systemen. March, 3-5 1993, Munich.

Figures

Figure 1: Hardware Environment.

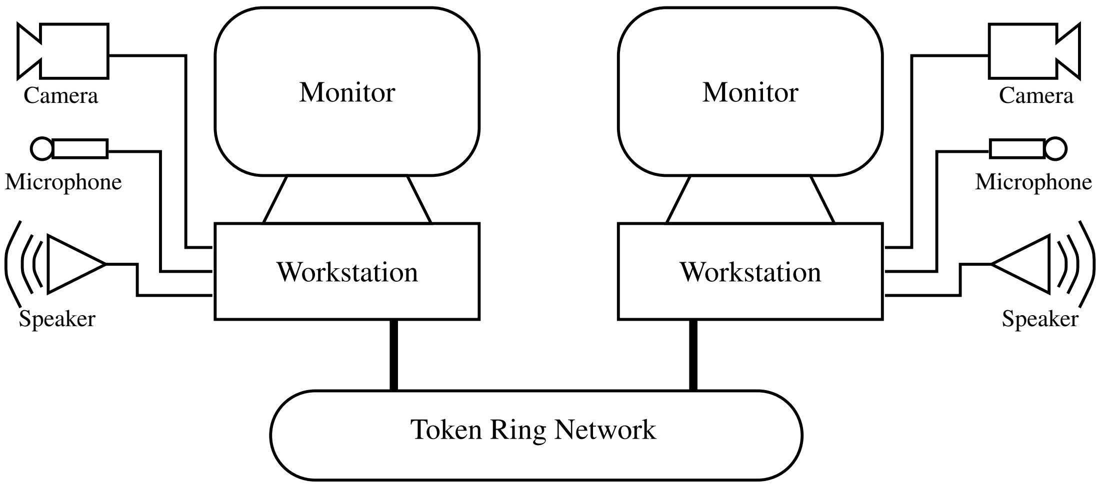


Figure 2: Separation of User Interface and Multimedia Handling

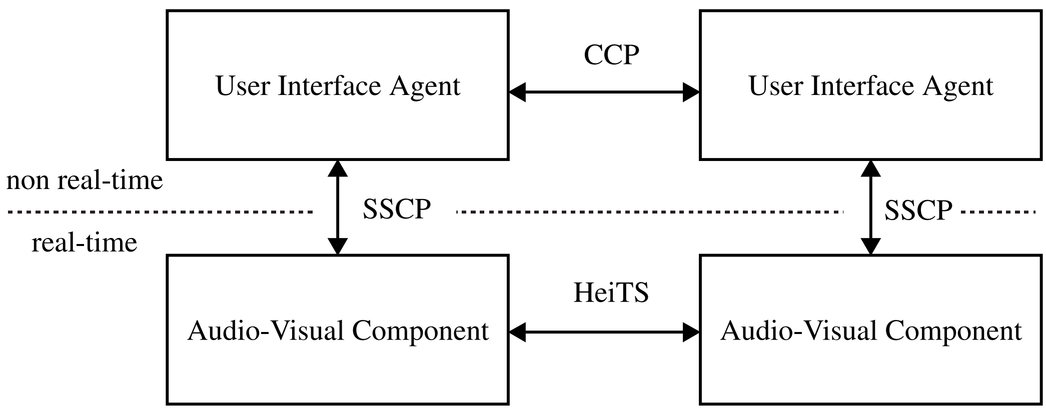


Figure 3: HeiPhone Main Window — Users Phone Book on the Left Side

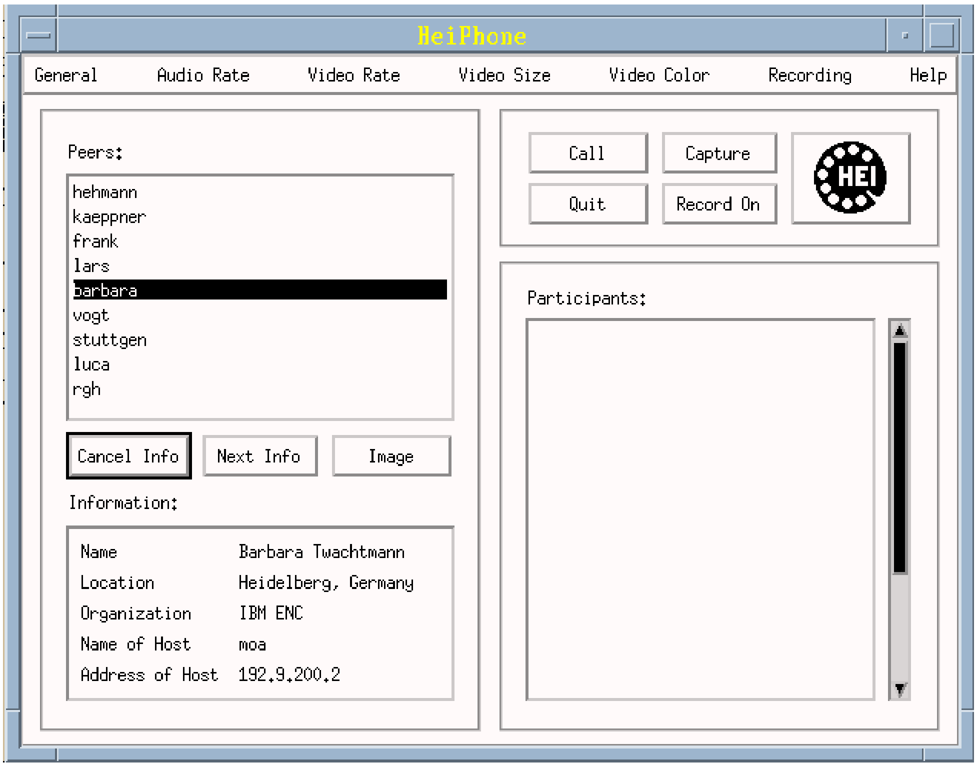


Figure 4: Invitation to a Conference

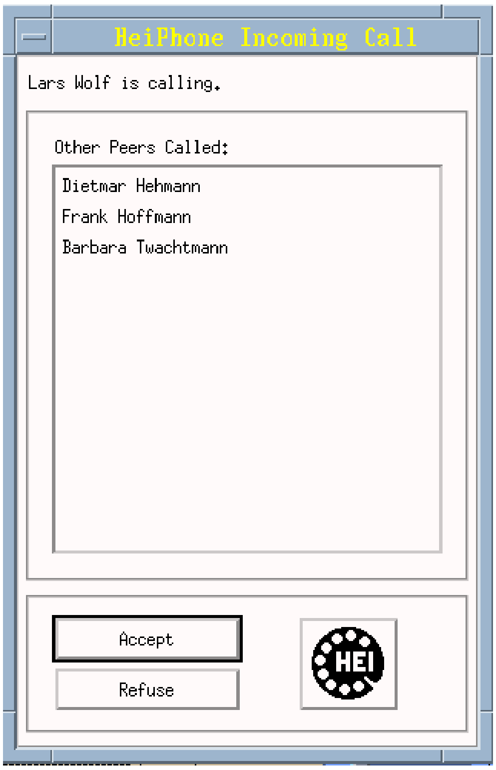


Figure 5: Conferencing States

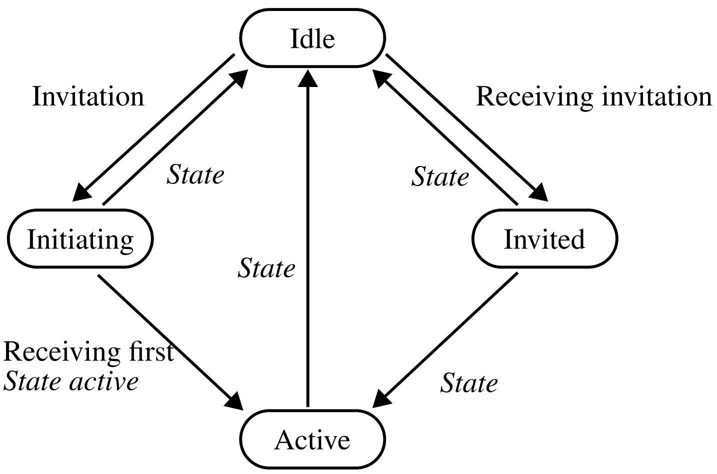
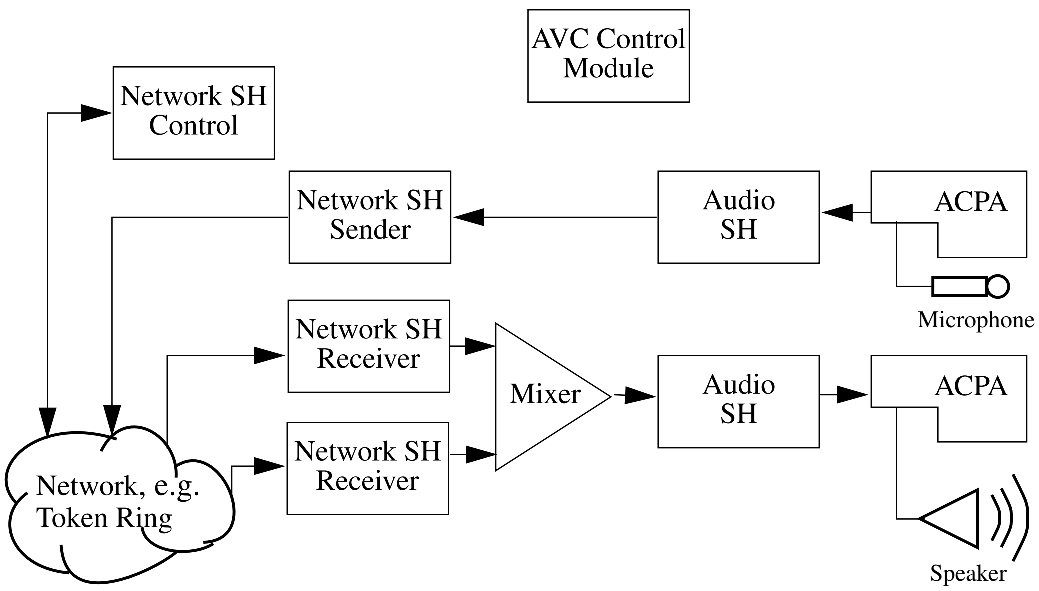


Figure 6: Structure of stream handlers during 3-way audio conferencing



Tables

Table 1: Conference Control Protocol Primitives

Primitive	Content
Invitation	Initiator Time stamp List of Invitees
State	Initiator Time stamp Participant New state

Table 2: Source and Sink Control Protocol Primitives

Request	Parameters	Return Values
ListEndpointAVTypes	EndpointType AVMode	AVTypeList
OpenSource	AVType	Endpoint
OpenSink	AVType	Endpoint
CloseEndpoint	Endpoint	none
AddSinks	SourceEndpoint SinkEndpointList	none
RemoveSinks	SourceEndpoint SinkEndpointList	none
SetTransmission	SourceEndpoint Mode	none
GetAVOption	Option	OptionValue
SetAVOption	Option OptionValue	none

Event	Parameters
EndpointClosed	Endpoint
SinkDropped	SourceEndpoint SinkEndpoint Reason

Architecture of HeiPhone: A Testbed for Audio/Video Teleconferencing

Thomas Kaeppner and Lars Wolf

IBM European Networking Center
Vangerowstr. 18
D-6900 Heidelberg 1

{kaeppner, lwolf}@dhdibm1.bitnet

Keywords: Multimedia conferencing, audio transmission protocol, video transmission protocols, conference control protocols

Architecture of HeiPhone: A Testbed for Audio/Video Teleconferencing

Summary

Due to recent advances in computer technology the integration of multimedia data with computing is becoming feasible. Such integration will allow for scenarios in which computer systems support collaborative conferencing. We have developed a testbed for multimedia teleconferencing that is used to study experimental protocols that can manage and organize multiparty conferences and transmit digital video and audio data. We distinguish between real-time communication, which is needed to support the transfer of continuous-media data, and non-real-time communication, which is used to exchange control messages such as invitations to a conference. These communication types are supported by different communication stacks within our testbed: Digital video and audio are transmitted via ST-II and HeiTP with media-specific protocols on top of them and control information for conference management is transmitted via remote procedure calls (RPC). An overview of the working system, its current communication mechanisms, and the services provided is the subject matter of this paper.

1 Introduction

1.1 Motivation

Due to recent advances in computer technology, high performance workstations with digital audio and video capabilities are becoming available. For the first time the integration of multimedia data with computing is becoming feasible. This integration will allow for scenarios in which computer systems support services such as video conferencing, news distribution, advertisement, and entertainment. Conferencing is a particularly important mechanism, since it is a prerequisite in any collaborative environment. We have built a working testbed in order to experiment with communication protocols for multiparty audio/video conferencing. An overview of the structure of the system, its current communication mechanisms, and the services provided is the subject matter of this paper.

1.2 Related Work

There are a number of multimedia projects that are investigating mechanisms for person-to-person conferences using shared text-oriented workspaces, graphics, audio, and video. Whereas [Lantz 86] and [Sarin and Greif 85] have studied conferencing architectures for text

and graphic, [Forsdick 85] and [Aguilar et al. 86] support audio conferencing. [Ludwig and Dunn 88], [Addeo et al. 88], and [Ahuja et al. 88] have built conferencing systems for video. However, in none of these systems was audio/video conferencing fully integrated with the workstation. Recently [Jeffay et al. 92] presented an integrated system built on existing protocols. Our work is targeted to providing an environment in which experimental protocols for both control of conferences and transmission of audio/video data can be evaluated.

The rest of this paper is organized as follows: The next section will briefly describe the hardware environment. In Section 3, we present an overview of the layered software architecture. Section 4 is concerned with conference control mechanisms in the upper layer. Section 5 describes the protocol used for communication between the layers and Section 6 presents structure and communication mechanisms in the lower layer of the system. Experience with the current system is described in Section 7, and finally Section 8 concludes.

2 Hardware Environment

Our hardware environment is illustrated in Figure 1. We are experimenting with HeiPhone on a network of IBM's RISC System/6000 workstations running under AIX Release 3.2.

M-Audio Capture and Playback Adapters (M-ACPA) which are built into the workstation are used to digitize sound and convert it back to analog form. The supported data types include 8-bit and 16-bit linear Pulse Code Modulation (PCM), μ-law companded, and A-law companded sample formats at 8k, 11k, 22k, and 44k samples per second. Using Adaptive Differential Pulse Code Modulation (ADPCM), the supported sampling modes consist of voice, music and music stereo.

Video boards manufactured by Rasterops allow the display and capturing of video images in a window.

Network connectivity is provided by a Token Ring that features a bandwidth of 16 Mbit/s and is used for transmission of both, continuous-media and traditional computer data. The underlying continuous-media transport system is also being ported to Ethernet.

3 Structure of the System

The challenge of multimedia systems lies in integrating the processing of traditional computer data, such as text and graphics, with processing of continuous-media data. Since continuous-

media data must obey strict time constraints, their processing requires real-time support [Hanko et al. 91]. On the other hand, operations that control the state of a conference or change the flow of data are not strictly time-bound but should provide a fast response. These different requirements have a strong impact on our model of the system.

As part of the HeiProjects [Herrtwich 92] we develop a system environment [Herrtwich and Wolf 92] that allows the processing of continuous-media data in real-time. This system environment is used by the conference testbed to implement all operations dealing with the flow of audio and video. Conference control and other computations are running on top of regular operating system mechanisms. This distinction leads to the two-layered structure that is depicted in Figure 2. The upper layer is referred to as the User Interface Agent (UIA), combining the functional modules of (1) controlling conferences and (2) providing the user interface. The lower layer is called the Audio-Visual Component (AVC) and handles the flow of continuous-media data within the system. By the Source and Sink Control Protocol (SSCP), implemented as Remote Procedure Calls (RPC), the UIA requests the AVC to establish data streams to other conference participants.

Different mechanisms are utilized for communication within each layer: Whereas conference control protocols are implemented as RPCs on top of TCP/IP, real time communication is supported by a stack of protocols that we refer to as the Heidelberg Transport System (HeiTS) [Hehmann et al. 91]. Exploiting different mechanisms for transport of control and audio/video data is not only motivated by differing service requirements, but also makes the system easier to extend and allows for an independent exchange of data transmission technology.

The following sections give more detailed explanations of each component.

4 The User Interface Agent

User Interface Agents are communicating via a Conference Control Protocol (CCP) in order to exchange state information and to manage conferences. Currently this management is fully distributed, i.e., there is no central entity of control. Moreover, UIAs provide the user with a graphical user interface in order to give him control over the conference. Since for experimenting with different conference control protocols the user interface must be adapted, these two functional blocks are interdependent and have been combined into one module.

4.1 Conference Model

The current CCP implements a model that we call generic conferencing. All participants have the same access rights during ongoing conferences. A conference is established by a user acting as an initiator. The initiator selects users in a phone book he wants to invite to a conference (see Figure 3). The initiator can also choose the types of media that he wants to communicate with. By clicking on a call-button he initiates the conference. Invitees are alerted by a window presenting information about the proposed conference and potential participants (see Figure 4). They can independently decide to either accept or reject the call. As soon as the first person has accepted, data streams, with types according to the selection of the initiator, are established between the participants. Further connections are established as invitees join the conference. Every participant can leave the conference without affecting communication between other members. The conference is closed when no other participants are left with whom to communicate.

4.2 Conference Control Protocol

Currently the CCP distinguishes only two message types as shown in Table 1. The *Invitation* message is used to deliver information about the proposed conference to all invitees. It contains the initiator's address and a time stamp, which together serve as a unique conference identifier for future messages belonging to this conference. A list of all invitees is included as supplemental information. Finally, the media which will be used during the conference are specified in detail.

The *State* message is used to distribute changes of state with regard to the conference to other participants. Apart from the conference identifier, the message contains the identification of the sender and the new state, to which the participant is advancing. Within the generic conferencing model we distinguish very few states (see Figure 5).

In order to avoid synchronization problems during state advances, we follow a simple policy by which changes of state are distributed to the initiator and all invitees whose state is not 'idle'. If *State* messages belong to a conference for which an invitation has not been received yet, they are stored in a database and their processing is postponed. When an invitation arrives, the database is searched and matching state messages are processed. Since state changes originating simultaneously at different sites can lead to inconsistencies and result in unwanted

growth of the data base, very old entries are deleted cyclically based on the time stamp information.

5 The Source and Sink Control Protocol

Communication between the User Interface Agent and the Audio-Visual Component (AVC) is achieved via the Source and Sink Control Protocol (SSCP). Within SSCP data streams are viewed as connecting a single source with potentially many sinks. One AVC manages all endpoints of data streams that are local to a single participant's workstation. SSCP[1] is endpoint-oriented, in the sense that all protocol primitives operate on these endpoints (see Table 2).

A UIA can request information about the media types that an AVC can support (via `ListEndpointAVTypes`), so that the selection, which is presented to the user, only includes supported media types. A data stream is established by issuing several requests:

- `OpenSink`() prepares the sink, so that data can flow from the network interface to appropriate output devices. If no sink is currently opened an initialization takes place, otherwise modules are started (e.g., Mixer) to handle an additional incoming data stream. The request returns address information that is necessary for establishment of audio/video connections.
- `OpenSource`() prepares the source, so that data can flow from appropriate input devices to the network interface.
- `AddSinks`() logically adds one or several sinks to the data stream originating at the specified source. This operation can be executed regardless whether the sinks are already connected.

The transmission of data over a specific data stream can be suspended and resumed by the `SetTransmission`() operation. With `GetAVOption`() the AVC is requested to report the actual settings for specified options, e.g., volume, treble, hue, saturation, etc. A UIA can manipulate these settings by issuing `SetAVOption`() operations.

State changes, which originate at the AVC mainly in error situations, are reported to the UIA by the events *EndpointClosed* and *SinkDropped*. That allows for recovery of failing connections without user interaction.

1. This protocol is a modified version of the SSCP used in [Altenhofen et al.].

6 The Audio Visual Component

The AVC handles the flow of audio and video data through the system. However, it has been designed not only to support SSCP, which is a protocol highly specialized to conferencing, but more generally to allow for flexible processing of audio/video data that can serve other multimedia applications as well.

6.1 Architecture of AVC

The AVC consists of a set of stream handlers each capable of executing a specific task related to the processing of continuous-media data. A Control module provides the SSCP-interface for the upper layer. The set of stream handlers consists of: Filer, Audio-device, Video-device, Network, and Mixer.

The Control module instantiates these stream handlers when they are needed. By means of a message based protocol, the behavior of stream handlers is steered. The Control module can logically connect stream handlers to a sequence, thus setting up the flow of data. (See Figure 6, which illustrates the stream handler structure at one site during 3-way audio conferencing.)

The Filer allows to store and retrieve continuous-media data on disk. A description of the data is stored in a supplemental file for use by editors or viewers.

The Audio-device and Video-device manage input and output on the respective adapters. They make data types and functionality of the adapter accessible by the Control module.

The Mixer combines several audio streams into one by digitally mixing audio samples. Different mixing strategies are provided for conferencing situations and mixing of data read from disk, in order to optimize the delay and the synchronization of media streams, respectively.

The Network stream handler allows the establishment of network connections via HeiTS. For every connection, the Network stream handler instantiates either a sender or a receiver stream handler. These handle data packets that are sent and received by HeiTS, whereas control packets of HeiTS are passed through the Network stream handler itself.

6.2 Continuous-Media Transport System HeiTS

We exploit HeiTS for transmission of time-critical audio and video data. Coexisting with TCP/IP, HeiTS enables to exchange streams of data with quality of service (QoS) guarantees. Users of the transport system specify their QoS requirements with regard to the transmission. Service is provided by a stack of protocols consisting of HeiTP (Heidelberg Transport Protocol)

[Delgrossi et al. 92], ST-II [Topolcic 90], and HeiDL (Heidelberg Data Link) [Twachtmann 92]. HeiDL is a data link access method, which interfaces several networks. On top of it ST-II, as a connection-oriented network layer protocol negotiates resource reservations and transmits continuous-media data in a stream-oriented fashion. HeiTP provides error correction and fragmentation/reassembly functionality.

Most parts of HeiTS are implemented in user space to ease porting. Currently HeiTS has been implemented for AIX 3 and OS/2. In order to achieve real-time characteristics, HeiTS, as well as the stream handlers introduced, make use of the real-time environment and resource management that is described in the next section.

6.3 Real-time Support and Supplemental Services

Guaranteeing real-time characteristics requires knowledge and control about the amount of resources needed for certain operations. The Heidelberg Resource Administration Technique (HeiRAT) [Vogt 92] contains the mechanisms to calculate requirements of streams with regard to buffer space, CPU time, and network bandwidth during their establishment phase. Based on available resources HeiRAT either admits or rejects the establishment of new streams. When a new stream is admitted, the resources are reserved and then scheduled according to HeiRAT's scheduling mechanisms. Scheduling utilizes a rate-monotonic algorithm [Liu and Layland 73], which assigns fixed priorities, except for necessary shifts, that can be caused by the establishment of new connections. Thus, the overhead of priority determination and setting is avoided.

The CPU scheduler is built on top of basic real-time capabilities offered by the AIX 3 operating system. This operating system supports fixed priorities which (1) are higher than the one of the standard AIX scheduler and (2) will not be changed by the standard AIX scheduler.

In order to avoid unpredictable processing delays caused by paging operations, all memory that is used during the processing of continuous-media data is 'pinned' in physical memory.

Data movement operations are reduced to a minimum by a buffer management system, i.e., stream handlers exchange pointers to a common data pool rather than the actual data when data is flows from one stream handler to the next.

6.4 Audio Processing

Since the streams of audio form a fully connected network between the *n* participants of the conference, every member receives *n-1* data streams which must be mixed together. Compared

to a situation in which mixing is performed at a central location, this distributed mixing generally causes a shorter execution path since packets are transferred directly to the endsystem. Whereas centralized mixing requires fewer network connections than distributed mixing, the latter not only gives the capability to independently control the volume of any received data stream, but also leads to a straightforward integration of local multimedia applications that use the audio adapter simultaneously.

6.5 Video Processing

The Video-device stream handler is responsible for capturing and displaying frames. Color depth and size of frames are set by the user. The full-size of the displayed video is captured and sent to all conference participants. Since at present we perform no compression of the image, frames are generally larger than the maximum transmission unit of the network. Thus, frames must be split into fragments before they can be handed to the network interface.

The current version of HeiTS provides reliable service by retransmission of lost packets. Since retransmissions can violate the time constraints of audio and video data, QoS parameters are chosen so that HeiTS does not provide reliable service.

A simple video protocol is implemented by adding a header to the video data containing additional information: A fragment counter contains the sequence number of the fragment, relative to the beginning of the frame. A frame counter denotes the sequence number of the frame since connection establishment. A flag reports, whether the received fragment is the last fragment belonging to a frame.

Using the video header, the display algorithm presents a frame when either the last fragment of a frame has been received or a fragment has been received with a frame number higher than those of previously received fragments. The video data is copied into a display buffer at a location corresponding to its fragment counter. Since only one display buffer is used to store all received fragments, data of lost fragments is replaced by data that has been presented at the same location within the most recent frame.

7 Experience and Work in Progress

Data streams of audio and video are flowing separately during ongoing conferences because (1) in a more general model of conferencing than the current implemented one, audio and video streams need not flow to the same targets, and more important (2) separate streams allow

to prioritize audio processing over video, which we have found very important for user acceptance.

We did not have to prioritize audio over video, but experimenting with conditions, in which no QoS guarantees could be given, we have experienced overload situations, in which an increasing backlog within the sequence of receiving network stream handler and video stream handler leads to a shortage of buffer space. Since video buffers can only be allocated up to the reserved maximum, the audio stream is left unaffected. Moreover the shortage gives chance to recover, since the load is not increased any further. This leads to a down-scaling of the video frame rate in overload situations.

The current implementation of the HeiPhone testbed is used within our own working group and has been presented at several forums and fairs (e.g., COMDEX Fall '92). User feedback has been overwhelming positive, the main criticism demanding more flexibility in adding and releasing video during ongoing conferences. The high-quality audio offered by the system was highly appreciated. Although the delay is very acceptable and not noticed during conversations, we know that some of it is caused by pre-buffering on the audio adapter. We have optimized the packet size of audio with regard to this prebuffering, which lead to packets containing 45 ms of sound.

Currently, we are using the testbed to implement several experiments:

- The Conference Control Protocol is extended such that media types can be flexibly changed during ongoing conferences, allowing for asymmetric participation with regard to the media types used. The enhanced model of conferences enables to switch between moderated conferences, in which a chairperson passes the floor of a conversation, to simple conferences, in which floor passing is managed by social protocols.
- The protocol for audio is extended in order to implement silence detection. Silence detection has the capability to reduce the amount of transmitted data, however sometimes background noise of the listening party is considered as a desirable feedback. We are particularly interested in how silence detection can be used to suppress disturbing echos which can be caused by the combination of regular speakers and microphones.
- Time stamp information is included in both the audio and video protocol. A stream handler, that can synchronize audio and video based on that information is being implemented.

8 Conclusion

Conferencing is a particularly important mechanism, since it is a prerequisite in any collaborative environment. We have designed and implemented a testbed in order to experiment with communication protocols for multiparty audio/video conferencing. We distinguish between real-time communication, which is needed to support the transfer of continuous-media data, and non-real-time communication, which is used to exchange control messages such as invitations to a conference. The testbed has a layered structure dividing the system into a real-time supported layer for the handling of audio/video data and a layer which is concerned with conference management. The communication needs of the different layers are supported by different communication stacks. Using the testbed we have implemented a simple conference model that enabled us to achieve full integration of multimedia conferencing with computing. We have designed and implemented experimental protocols for the transmission of audio and video data and gained first experience in user trials.

Acknowlegements

We would like to thank Andreas Schröer for implementing large parts of the AVC, Ralf Guido Herrtwich for implementing the user interface, and Dietmar Hehmann for implementing the Video-device stream handler.

References

[Addeo et al. 88] E.J. Addeo, A.B. Dayao, A.D. Gelman, V.F. Massa: An Experimental Multi-Media Bridging System. Bell Communications Research, 1988.

[Aguilar et al. 86] L. Aguilar, J.J. Garcia-Luna-Aceves, D. Moran, E.J. Craighill, R. Brungardt: Architecture for a Multi-Media Tele-Conferencing System, ACM SIGCOMM, August 1986.

[Ahuja et al. 88] S.R. Ahuja, J. Ensor, D. Horn: The Rapport Multimedia Conferencing System. Proceedings of the Conference on Office Information Systems, March 1988.

[Altenhofen et al. 93] M. Altenhofen, J. Dittrich, R. Hammerschmidt, R.G. Herrtwich, T. Kaeppner, C. Kruschel, A. Kückes, T. Steinig: The BERKOM Multimedia Collaboration Service. Submitted to First ACM International Conference on Multimedia ‘93.

[Delgrossi et al. 92] L. Delgrossi, C. Halstrick, R.G. Herrtwich, H. Stüttgen: HeiTP: A

Transport Protocol for ST-II. Proceedings of GLOBECOM' 92, December 1992, Orlando, Florida.

[Forsdick 85] H.C. Forsdick: Explorations in Real-Time Multi-Media Conferencing. Proceedings of the Second International Symposium on Computer Message Systems, IFIP, September, 1985.

[Hanko et al. 91] J.G. Hanko, E.G. Kuerner, J.D. Northcutt, G.A. Wall: Workstation Support for Time-Critical Applications. Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 18-19, 1991, Heidelberg.

[Hehmann et al. 91] D. Hehmann, R.G. Herrtwich, W. Schulz, T. Schütt, R. Steinmetz: Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High-Speed Transport System. Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, November 18-19, 1991, Heidelberg.

[Herrtwich 92] R.G. Herrtwich: The HeiProjects: Support for Distributed Multimedia Applications. IBM ENC Heidelberg, Technical Report No. 43.9206.

[Herrtwich and Wolf 92] R.G. Herrtwich, L.C. Wolf: A System Software Structure for Distributed Multimedia Systems. Proceedings of the Fifth ACM SIGOPS European Workshop, September 21-23, 1992, Le Mont Saint-Michel, France.

[Jeffay et al. 92] K. Jeffay, D.L. Stone, T. Talley, F.D. Smith: Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks. Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video, November 12-13, 1992, San Diego, California.

[Lantz 86] K.A. Lantz: An Experiment in Integrated Multimedia Conferencing. Olivetti Research Center, Menlo Park, CA, December 1986.

[Liu and Layland 73] C.L. Liu, J.W. Layland: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. Journal of the ACM, Vol. 20, No. 1, 1973, pp. 46-61.

[Ludwig and Dunn 88] Laboratory for Emulation and Study of Integrated and Coordinated Media Communication. ACM SIGCOMM, August 1988.

[Sarin and Greif 85] S. Sarin, I. Greif: Computer-Based Real-Time Conferences. IEEE Computer, October 1985.

[Topolcic 90] C. Topolcic (Ed.): Experimental Internet Stream Protocol, Version 2 (ST-II). Internet Request for Comments 1190, Network Working Group, October 1990.

[Twachtmann 92] B. Twachtmann: The Heidelberg Transport System: Data Link Layer. IBM ENC Heidelberg, Internal working paper, October 1992.

[Vogt 92] C. Vogt: The Heidelberg Resource Administration Technique: Design Fundamentals, QoS Calculation, and Resource Reservation. IBM European Networking Center Heidelberg, Technical Report No. 43.9213, October 1992. Also: to be presented at Kommunikation in verteilten Systemen. March, 3-5 1993, Munich.

Figures

Figure 1: Hardware Environment.

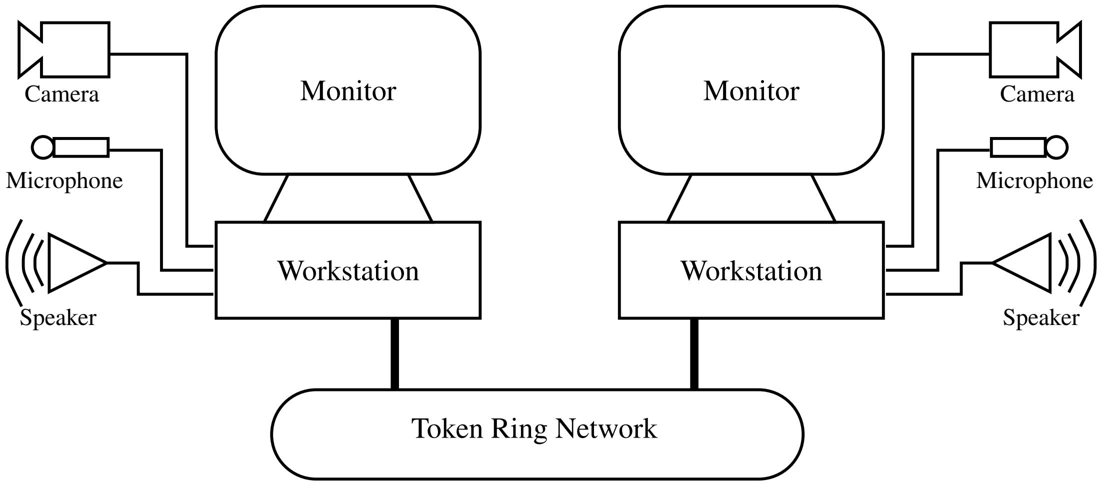


Figure 2: Separation of User Interface and Multimedia Handling

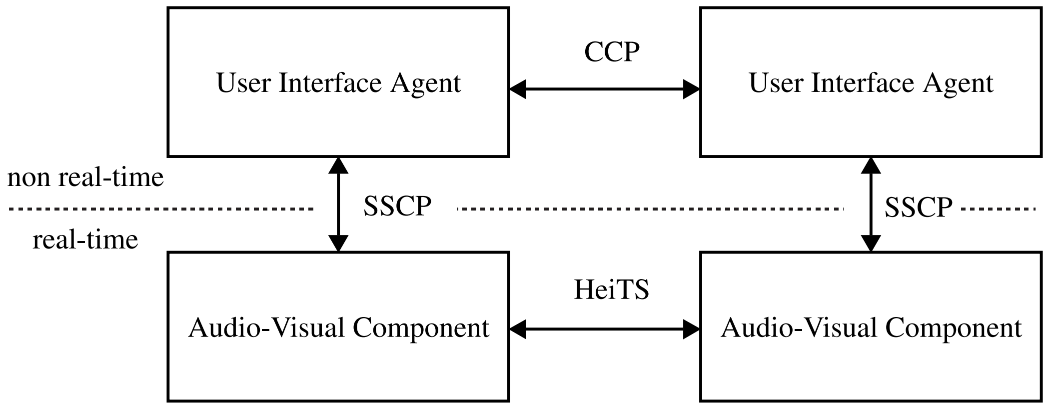


Figure 3: HeiPhone Main Window — Users Phone Book on the Left Side

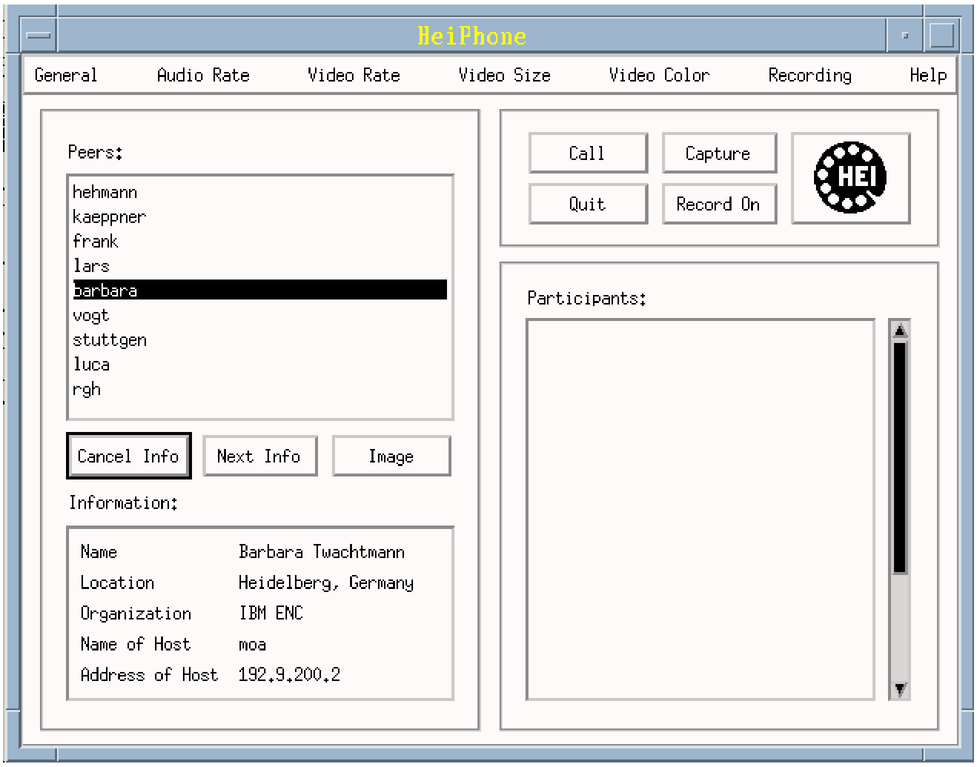


Figure 4: Invitation to a Conference

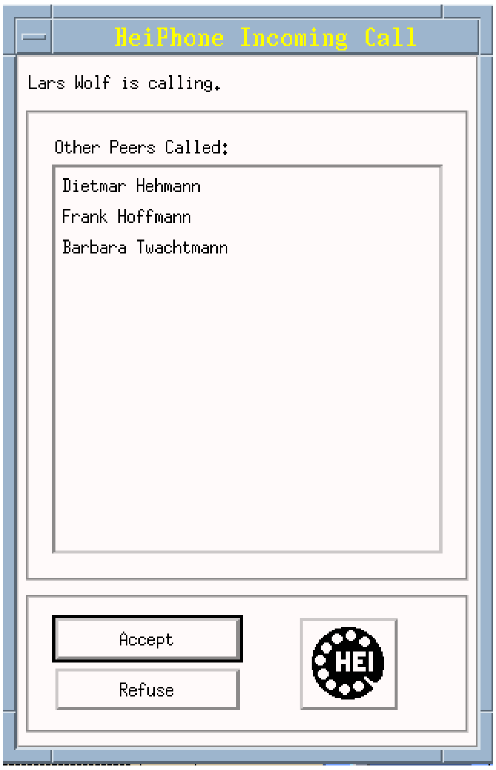


Figure 5: Conferencing States

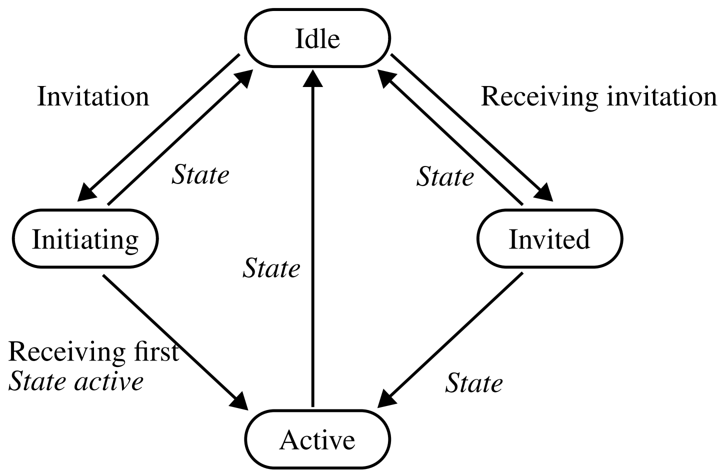
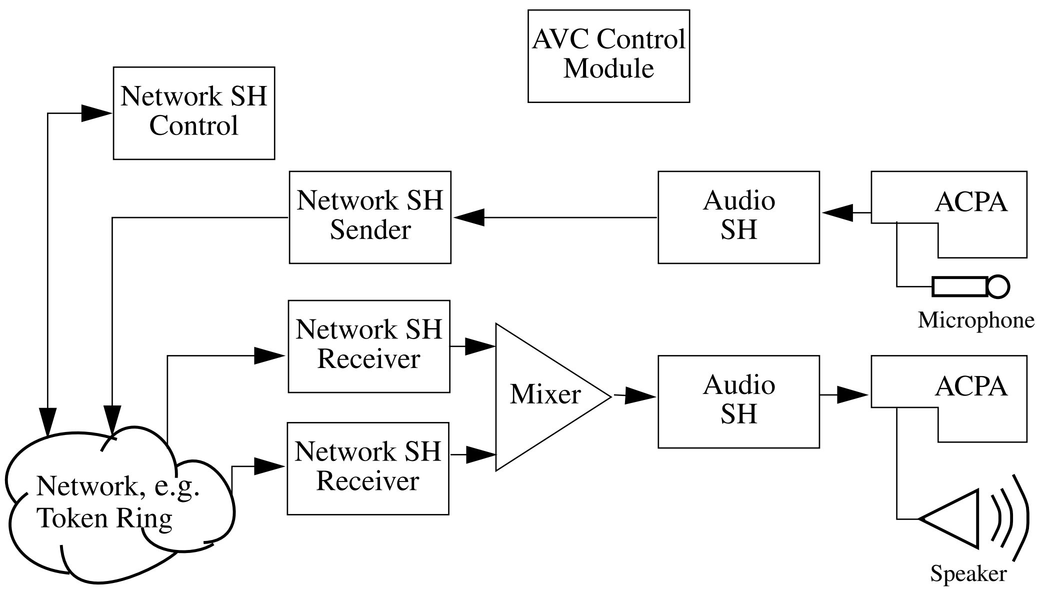


Figure 6: Structure of stream handlers during 3-way audio conferencing



Tables

Table 1: Conference Control Protocol Primitives

Primitive	Content
Invitation	Initiator Time stamp List of Invitees
State	Initiator Time stamp Participant New state

Table 2: Source and Sink Control Protocol Primitives

Request	Parameters	Return Values
ListEndpointAVTypes	EndpointType AVMode	AVTypeList
OpenSource	AVType	Endpoint
OpenSink	AVType	Endpoint
CloseEndpoint	Endpoint	none
AddSinks	SourceEndpoint SinkEndpointList	none
RemoveSinks	SourceEndpoint SinkEndpointList	none
SetTransmission	SourceEndpoint Mode	none
GetAVOption	Option	OptionValue
SetAVOption	Option OptionValue	none

Event	Parameters
EndpointClosed	Endpoint
SinkDropped	SourceEndpoint SinkEndpoint Reason