THE RAPPORT MULTIMEDIA CONFERENCING SYSTEM: A SOFTWARE OVERVIEW

J. Robert Enor  S. R. Ahuja  David N. Horn  S. E. Lucco

AT&T Bell Laboratories
Holmdel, NJ 07733

The Rapport multimedia conferencing system supports interactive, real-time, distributed conferences among two or more people. Executing on personal workstations interconnected by data and voice networks, Rapport provides basic mechanisms to create, manage, and terminate conferences. Our system provides an environment in which many types of meetings can take place, including telephone conversations, discussion among colleagues, and lectures. Existing workstation programs can be used during a conference to produce and edit data and displays for the conferences. Rapport is designed to help people emulate face-to-face conferences as closely as possible with their workstations.

This paper focuses on the Rapport software, which is a three-level hierarchy comprised of the user interface, conference server, and system levels. The user interface level provides humans and other programs with access to the Rapport system. Maintaining consistent views of the conference for all the participants, the conference server level coordinates the activities of the conferences and provides basic protocols for exchanging information and control during a conference. The system level includes window management service and drivers for the hardware required to transmit data in the various media.

1. INTRODUCTION

Rapport is a multimedia conferencing system which executes on a collection of workstations connected by data and voice networks. Through its user interfaces, this system provides communication protocols and control functions that allow geographically distributed users to conduct conferences. Conferences interact with a real-time exchange of information in several media, including voice, graphics, images, and text. They talk with each other and use existing programs to produce and edit common displays on their workstations. Rapport also supports conventional telephone service as a conferencing activity.

A primary goal of the Rapport design is to provide a framework for systems that give remote conferences approximately the same opportunities for exchange of control and information as they have when meeting face-to-face. We have informally studied meetings, focusing on the ways in which people communicate and the protocols that they use to coordinate their communications. Our intent has been to identify basic communication methods, such as speaking and pointing, and basic means of effecting typical protocols, such as raising a hand and speaking to interrupt another speaker. Many important behaviors, such as the use of body language, are not available to us through workstations.

(In our current implementation, we do not support the video medium.) We hope that Rapport's communication and control features support common meeting behavior, and hence will be relatively easy to use. Studies of remote, computer-based conferences (e.g., [Jo 79]) have not yet found a definitive set of features for remote conferencing systems; thus we plan to use Rapport to gain some of the data required for determining these design parameters.

Most people are familiar with the conduct of face-to-face meetings. Many techniques for conducting meetings have been discussed (e.g., [Do 76] and [Go 85]), and some have been used as the bases for computer support. Two well-known computer-based systems—the Colab [St 87] and Nick [Be 86] projects—are intended to support face-to-face meetings, and each provides direct support for a specialized meeting activity. The Cogniter system [Po 86] within Colab is designed to help people with brainstorming activities, and the Liza agent [El 87] of Nick is designed to serve as a meeting facilitator [Do 76]. In contrast, Rapport does not try to support face-to-face meetings or any particular meeting management methodology. However, we believe that specialized support of these activities or management techniques can be readily built using Rapport.

Another primary goal of Rapport is to use existing application programs with little or no modification as part of multiple-workstation conferences. We feel that one should be able to use a favorite editor, VLSI CAD system, financial planning expert system, etc., during a computer-based conference. We call user-level tools that are associated with a conference application programs. Other multimedia conferencing projects (e.g., [La 86], [Le 87], [Su 86]) also allow existing application programs to execute within a conference. The difficulty in allowing these programs to execute within a conference is providing the necessary input/output environment for them. For example, Lantz has described a solution to this problem through use of a communication system implemented in the V operating system [Ber 86] that allows appropriately built application programs to be used without modification by a single user or in a conference [La 86]. Similarly, in Rapport we use the X window system [Sc 86] executing on UNIX as a standard input/output environment. Thus, many programs based on this environment can be run without modification by conference participants.

One of the most common conferencing activities is the telephone conversation, and we support this type of conference with Rapport. We view the telephone as a ubiquitous communication interface and one that represents the simplest interface to a Rapport conference. A Rapport works-

* UNIX is a trademark of AT&T Bell Laboratories
tion may establish conferences with a heterogeneous mix of communication devices, including telephones. Thus the user interface to Rapport can vary at the different sites within the conference, each incorporating as much sophistication as possible. Rapport supports voice transmissions and conventional telephone service within conferences, though we should note that we are not building a general purpose voice storage and editing system like the Etherphone project [Ad 86].

This paper focuses on the facilities provided by the Rapport software. Section 2 describes the model upon which the Rapport design is based. Section 3 outlines the overall execution environment and software architecture of Rapport. In section 4 we briefly introduce our initial user interface. Section 5 contains a more detailed look at the conference protocols, and section 6 describes the system level. We summarize our design and its impact on our workstation requirements in section 7.

2. Virtual Meeting Rooms

During the design of Rapport we have had to ask many times whether a given feature should be supplied by a conferencing system. For example, a lively topic is whether or not a conferencing system should supply its users with a multimedia editor. We base our answers to such questions upon a model we have developed. We view Rapport as a provider of virtual meeting rooms (see Figure 1). This perspective highlights important characteristics of face-to-face meetings that we think should be reproduced in the Rapport conference environment. It also allows us to omit rationally features that are supplied in some other systems. This section introduces the representation of the virtual meeting room that is presented to users on Sun workstations, while section 4 contains an example of our initial user interface for these machines.

According to our metaphor, one starts a conference by going into a meeting room and then inviting others to join him or her there. Similarly, when a person wishes to join a meeting room, he or she will be notified of requests for new conferees and to post signals, i.e., “raise their hands” for attention.

Rapport provides a few other services which we also associate with the virtual meeting room metaphor. Each conferee has a “beeper” or message service. While a person is in a meeting, he or she will be notified of requests for new meetings. These requests, which include incoming telephone calls, may be annotated with a description. A receptionist may be associated with each conference room to check conferee identification, to “pass out literature,” etc. Finally, Rapport supports a recording service, which could provide the basis for the minutes of a conference.

3. System Overview

Rapport currently executes on VMEbus-equipped Sun workstations. A 10Mbit Ethernet is used to transmit data among the workstations, and a separate network is used to transmit voice. Figure 2 illustrates a collection of workstations and indicates some architectural features of each machine. Each Sun has been augmented with a voice interface, which is linked through serial ports to the main processor board. The voice interface also connects to a headset and to the voice network. The voice network is managed by a custom-designed switch that provides hook-up to conventional phone lines (hence giving each workstation voice access to any phone).

Currently, we support the complete Rapport functionality only over a cluster, those workstations connected by a local area network. The only mechanism we have to connect clusters is the standard voice-based phone connection, and hence our wide-area conferencing is still limited to voice
conferencing. We hope to support Rapport fully over wide-area networks when data transmission is supported in that domain.

As illustrated in Figure 3, Rapport's software is divided into the system, conference server, and user interface levels. Each level provides a set of services for the clients within the higher levels. A level is defined by its operations and events. An operation, corresponding to the common notion of procedure, provides the means for a higher level to request action of lower levels. For example, a client in the user interface level may request that a server in the conference level take some action by invoking the appropriate operation. An event is a message passed from a lower level to a higher level. Events notify interested clients of certain changes, such as the addition of a conference participant, that occur at the lower level. Thus, events occur asynchronously with respect to the activities at the higher level.

The user interface level provides other programs and humans access to Rapport. The conference server level maintains consistent views of the conference for all conferees. This is accomplished by coordinating activities of the conferees and providing basic protocols for information and control. The system level controls the hardware that we have developed to transmit data in the various media. This level also uses the UNIX operating system and Network File Server from Sun. We use a modified version of the X window system as a basis for our input/output support.

This software structure has been implemented through a specific process configuration. Each workstation executes exactly one conference server. For each conference, each workstation executes exactly one user interface process and exactly one process for each application program that has been initiated locally.

4. User Interface Level

At this writing we have implemented only a primitive realization of the virtual meeting room concept in our user interface. A snapshot of this interface is captured in Figure 4. Our user accesses Rapport through a conference interface window. This window represents all the virtual meeting rooms associated with the various conferences in which our user is a participant. A person can participate in many conferences concurrently; however, a user can actually be in only one conference at a time. Metaphorically, this corresponds to being able to move among several meeting rooms, but being in only one at a time. In the figure, our user is a member of three conferences—bob.0, sid.0 and dave.0—as indicated by their names appearing near the top of the conference interface window. This user is presently active in the conference sid.0, which has two participants, Sid and Dave.

Each conference can have several associated application programs. sid.0 has two shared applications (xterm and bitmap), their displays representing two "walls" of the current meeting room. Each program has an associated window. The person initiating an application program is responsible for positioning the associated window, but all windows have the same appearance on each conferee's workstation.

The input control of a particular application resides with only one person at any given time. Only the person with the input control can provide input for the program. This control is explicitly transferred from one conferee to another. In our example Sid has control of xterm, while Dave has control of bitmap.
The figure also shows Dave’s pointer on Sid’s screen. Dave is using it to draw attention to a command in the bitmap editor. This pointer is provided by Rapport and is not part of the bitmap program. Each conferee can have a pointer, which is labeled by the conferee’s name (except on the pointing conferee’s local screen). As done with the “telepointers” in Colab [St 8G], we display all active pointers.

5. Conference Server Level

The conference server level is the heart of Rapport; its events and operations define the conference abstraction. This level has two main responsibilities. First, the conference server supports interactions among conference participants. It provides mechanisms for rendezvous and communication among people that permit initiation, termination, and conduct of meetings. Of particular interest in this regard is that the server supports pointing and signaling methods. Second, the conference server supports the execution of application programs and maintains their consistency among the workstations in the conference.

5.1 Supporting Conferee Interactions

The conference server supports protocols used to initiate, terminate, and conduct meetings. These protocols use messages that are distinct from those used to transmit shared information in the various media during the conference. Our experience with telephone conversations, the most common form of remote conferencing, has shown us that these protocols are even distinct from transmissions in the voice medium. Thus the various protocols implemented by this level have been designed to function with messages from different types of devices. As a result, Rapport is able to support conferences involving heterogeneous workstations and to incorporate conventional telephone service as a subset of our general conference.

We elaborate on these points through an example—the initiation of a conference involving two conferees, each using a Sun workstation. Figure 5 illustrates the messages necessary to set up the conference sid.0 between participants Dave and Sid (the example used in Figure 2). Sid decides to begin the conference, and his user interface sends message h1 to conference server A indicating that Sid and Dave are to be the participants. Sid’s conference server, A, then sends message h2 to the local voice interface to enable his voice connection and message h3 to Dave’s conference server, B, to build a new conference named sid.0, with A as its owner. The message M3 includes the list of desired participants. At this stage both conference servers know about the potential conference. In our conceptual model this corresponds to having a reservation for a particular virtual meeting room. At this point conference server B posts event M4 for Dave’s user interface. The interface may tell

1 Advanced telephone protocols, e.g., call forwarding and call waiting, require a control channel in addition to the voice channel.

2 Server A uses its unique machine identifier to generate a unique conference identifier and distributes it to B. A separate, generalized name service would be desirable.
Dave of the meeting request. In our interface, Dave may forward such requests to another workstation where someone else may deal with it as desired. At any rate, to continue with this example, Dave now indicates that he wishes to participate in the conference sid.0. As a result, his interface sends message M5 to the server B. This is conceptually equivalent to entering the associated meeting room (and is also equivalent to answering a telephone call). Conference server B sends message M6 to conference server A, notifying it of Dave's decision. A then sends event M7 to Sid's interface. Concurrently, B sends message M8 to Dave's voice interface, enabling the voice connection. The conference has been established with a voice and control connection. The participants may now initiate shared applications.

The messages sent among the conference servers and the events the servers generate for higher levels of software represent the conference protocols of Rapport. These protocols are meant to serve as a mechanism supporting various protocols at the human level, not as a policy for conference conduct. An example should clarify this distinction. When a person wishes to join a conference, several issues of policy are important, including who may join a conference and who may admit new participants to a conference. One might decide that only a person whose name is on an access control list or knows a password may join a given conference. Similarly, one might decide that a new conferee may be admitted by the conference chair, by a sponsoring participant, or through a majority of voting conferees. Rapport makes no such policy decisions in its conference server, but the server's protocols do provide the opportunity for higher level software to make or to aid such decisions. In our present example, suppose the user associated with conference server A wishes to join a conference currently involving servers B and C. Server A sends a join-request message to B and C. Servers B and C then post locally corresponding events for higher level software. The user interface of the initial Rapport prototype notifies each conferee that the join request has been issued. The conferees can then act on the request as they wish. If any user issues an add-member request, each conference server is notified and the new conferee is added to the conference. Other user interface levels of Rapport could be built, enforcing particular policies for conference conduct. For example, when receiving the join-request event, a user interface level program could require that conferees vote on the request and issue an add-member operation only if the majority said to do so.

Our experience has indicated that pointers are very useful during remote meetings; in fact, they seem almost essential at times. These pointers allow each conferee to direct the attention of the other conferees to a particular part of their shared display, providing a point of reference for the discussion. Signals are another control mechanism provided by the conference server level of Rapport. Each conference server can send a "signal" message to other servers, which then locally post the event for higher level software to use in some appropriate manner. In our present user interface level we use these signals to indicate that the sending user has "raised a hand."

5.2 Supporting Application Programs

Coordinating the input and output of application programs is a principal responsibility of the Rapport conference server level. Two ways to coordinate these activities are obvious. First, each workstation in the conference can execute each application program under some constraints of synchronization and input control. The advantage of this approach is that it tends to generate relatively little network traffic, since only the application program input commands are transmitted among the conference workstations. The major drawback of this technique is that each conferee must have the same software executing in the same environment. Some programs, written to utilize local state, are not suitable for use with this technique. For example, a bitblt program might expect to receive as an argument a pointer into its local machine's memory. Giving this command, with its argument, to each conferee would not preserve the consistency of the conference. The maintenance of this environmental consistency is not generally possible for a conference server that does not completely control the workstation.

In the second approach, a single site can execute each application program and broadcast its output commands to the other conferees' workstations. The major advantage of this approach is that it allows the various conferees to see the results of programs without executing them. Thus people can participate in conferences in spite of some differences in software and hardware. The disadvantage of this technique is that it might generate significant network traffic, since all the window level commands and arguments must be transmitted for display generation. (See [La 86] for additional discussion on these techniques and hybrid approaches.)
In the initial implementation of Rapport we tried the multi-site execution technique (termed the "fully-replicated" approach in [La 80]). We used Sun's Network File System (NFS) to provide the workstations of a cluster with storage for data and programs. All participants in a conference used a single directory tree to access the common files that permanently stored their shared data and application programs. Local files were used to store data during the conference. We tried to provide additional support for consistent execution of programs by using the same "shells" at each workstation. However, we found this approach unsatisfactory for any but the simplest conferences. In more realistic trials we simply could not maintain consistency among the conferees.

As a result of our experience with the first prototype, we believe that the fully-replicated approach does not provide a suitable environment for associating arbitrary application programs with a conference. Without constraining application program execution, the conferencing system cannot maintain the required consistency. An arbitrary application program can use local state, which may vary at each workstation. Additionally, the conferencing system must coordinate input and output of application programs without understanding the programs; thus the conferencing system cannot even realize that inconsistencies have occurred. In our second implementation of Rapport we are using a single-site execution scheme (termed the "centralized" approach in [La 80]), which is also used in [Le 88] and [Su 86].

6. The System Level

Much of the support described in this section is not actually part of Rapport, but it is necessary for our system's implementation. We mention this required support and our modifications to these existing systems to indicate the needed environment for Rapport software. The system level of Rapport provides the higher levels with multimedia transmission, network management, and display management.

6.1 Device Drivers

We have used Sun's resident operating system, UNIX, to provide support for processes. The conference server and the user level interface run as processes on each workstation. In addition, we have modified existing and installed new device drivers to support our voice and phone interfaces. We are currently adding a touch screen and its associated driver to the workstation. We are also building a special interface board, to be installed on the VMEbus of the Sun workstation, that will give local support to voice and video networking. This board will be handled through a special device driver.

6.2 Network Support

The effectiveness of a distributed conference depends on support of real-time communication. If the media transmissions used in a conference are not delivered to the participants in real-time, the interactions necessary for a conference are not possible. Often information flow within a conference is bidirectional, and this puts severe demands on the delay characteristics as well as the throughput of the underlying network.

The networking requirements for different media vary according to the intended interactions within a conference. Voice is always supported in full duplex mode, just as it would be in a face-to-face conference. We have implemented a separate voice network and voice switch to support conferencing.

An important characteristic of the Rapport conferencing system is that conferees may freely move among the conferences in which they are participating. The ability of a conferee to move between conferences places noteworthy demands on the voice network. Since each conference may have different participants, different collections of voices have to be combined and given to the user at each workstation. Thus each workstation must be able to transmit and control voice signals for all the conferences in which its user is a participant. This is done in our current prototype by separate networks, one for voice controls (the voice switch) and one for voice transmission. Each conference server has a separate control path to the voice switch allowing it to effect easy movement among several conferences. Eventually the interface board on each workstation will be able to locally conference voices [Ah 88].

All data and conference protocol control paths among conference servers are established using sockets from Sun's UNIX. The performance of our system is quite limited by large latency encountered in using sockets. This is especially true for the movement of pointers.

We currently do not support video transmissions within Rapport. Later, video will be supported on a separate network though software controlled by the conference servers. The major distinction between the voice and video networks is that video is single ended, i.e., only one video output is displayed on a workstation at a time.

The important point to note here is that the software requirements for control differ for each medium; thus we have modeled these connections as separately managed entities. This is not the same as supporting a "generalized data connection" that adds other media to an existing data path (e.g., [Le 88]).

6.3 Window Management

Rapport is responsible for coordinating the displays on the workstations of conferences, and uses the X window system [Sc 80] to interface to these devices. We chose to use the X system because it is widely-used and we could modify it as necessary. Two modifications are required in the single-site execution scheme. Input to each application program must be filtered because we have only one input at a time for each application (this is also true in the multi-site execution scheme). The other modification allows a conference server to multicast the output of applications to remote X servers, where they are executed locally.

7. Summary

We have implemented a single-cluster prototype of Rapport using a widely available environment of Sun workstations, connected by an Ethernet and providing a software base of UNIX and the X window system. We have demonstrated that it is possible for such an environment to serve as the partial basis of a conferencing system. We have, initially, concentrated on providing the low level mechanisms
that allow users to establish and conduct real-time, remote conferences. Furthermore, we have succeeded in keeping the conferencing software independent of application programs, allowing existing applications to be used within any conference.

Many of the capabilities of Rapport were made possible by the environment in which it executes. However, our initial experience with Rapport has convinced us that even more powerful, as well as versatile, workstations are required to provide both a user’s communication and computing needs. Most important in the list of desired workstation features is better input/output interfaces. A keyboard and mouse are not the easiest or most pleasant controls for conducting conferences! We are exploring the use of touch screens and remote controllers as additional interfaces. Better networking support for both data and voice is also needed. Reduced overhead in the low level software supporting network protocols and better communication hardware will improve our support for real-time interaction. Finally, we recognize the need for good database support for creation, storage and sharing of electronic documents.

We plan to perform some experiments with new user interfaces, including comparisons of specialized interfaces and automatic support for user customization of conference control functions. We also anticipate the construction of various programs, such as calendar servers [Sa 85], that will use Rapport.

References.


