Journal of Network and Systems Management, Vol. 5, No. 3, 1997

The Role of Multicasting in Managing Interactive Multimedia Distance Learning Systems

H. Abdel-Wahab,[1,2] K. Maly,[1] E. Stoica,[1] and A. Youssef[1]

This paper discusses the important role of multicasting in designing, implementing, and managing interactive multimedia distance learning systems. This is achieved in the context of IRI, an Interactive Remote Instruction system for distance learning built at Old Dominion University. IRI is an Internet-based system which integrates continuous multimedia, shared applications and a variety of multi-user collaborative utilities. In this paper, we concentrate on the process architecture and dynamic multicast group handling as they pertain to managing multimedia resources, and show how they support robustness and short response time to user actions. IRI uses raw IP multicasting for audio and video streams and reliable multicasting for resource management and data sharing. The system is scalable (uses multicast for inter-process communication) and expandable (partitioned into a set of autonomous but cooperating components).

KEY WORDS: Multimedia resource management; distance learning; multicasting; distributed systems; virtual classroom.

1. INTRODUCTION

Over the past two years, the department of computer science at Old Dominion University has developed IRI (Interactive Remote Instruction) which melds video, networking, and computing technologies [1–3]. IRIs goal is to enable new educational possibilities by providing each student with a learning environment supporting multimedia interaction (video, audio, data sharing), tools for presentations, surveys, evaluation, homework, exercises, multimedia note-taking and annotations,recording/playback of classes, personal review sessions and study groups. We have integrated video and audio into a complex, highly interactive

[1]Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529. E-mail: {wahab, maly, stoic-e, youssef}@cs.odu.edu

[2]Correspondence should be directed to H. Abdel-Wahab.

1064-7570/97/0900-0351$12.50/0

remote instruction system which uses the Internet technology as a vehicle for communication.

Through the use of computers, high-speed networks, standard software (UNIX, Motif, Internet IP) IRI creates a virtual classroom where a student can be at home, in office, or in a remote classroom, yet participate fully. The teacher's image is displayed on the screen all the time. Every student hears the teacher's voice. If the teacher asks a student a question, or if a student needs attention, the image of that student is also presented. In addition, since the virtual classroom may consist of several rooms (sites), a wide angle view of each classroom (one at a time) is also displayed. The teacher can speak at any time. Students share a group of audio channels, each controlled by a token, and only the students with the token can speak. The participants can also share X applications (tools). For example, the teacher may present his notes using Netscape. Any student can take control of the tool while the others observe. Only the person who has the token, can interact with the tool. IRI allows students to take notes of a presentation, play back a presentation at a later time, see other students regardless of location, ask questions, and make presentations. Although in the previous discourse we used the term teacher, our system is built such that any person can take on this role. For example, a student while making a presentation becomes the "teacher".

Designing IRI, we faced problems in making this system usable by non-computer science people (e.g., history majors), scalable (to 100 users) and robust. We designed an interface that hides all implementation details from the user. Moreover, all interface-related modules are included in a separate process, thus providing short response time to user actions. We use reliable and unreliable multicast communication between processes on different machines. Thus, IRI can support any number of users. To provide robustness to our system, we implemented each functionality in a separate process. In this way, one failure does not cause failure in other subsystem parts.

In this paper we focus on the process architecture and dynamic multicast group handling as they pertain to the management of multimedia resources.

Section 2 is an overview of IRIs software architecture. Section 3 is a comprehensive discussion of IRIs inter-process communication. Section 4 gives the description of the Class Information Manager (CIM). Section 5 deals with managing IRIs shared resources. Section 6 discusses the video and audio subsystems with emphasis on the dynamic multicast group handling. Section 7 gives an overview of IRIs tool sharing engine and Section 8 contains our conclusions.

2. SYSTEM SOFTWARE ARCHITECTURE

Figure 1 shows the multimedia software architecture of IRI. The IRI system is composed of two major components: the Class Information Manager (CIM) and the Remote Instruction Server (RIS). There is only one CIM for the entire

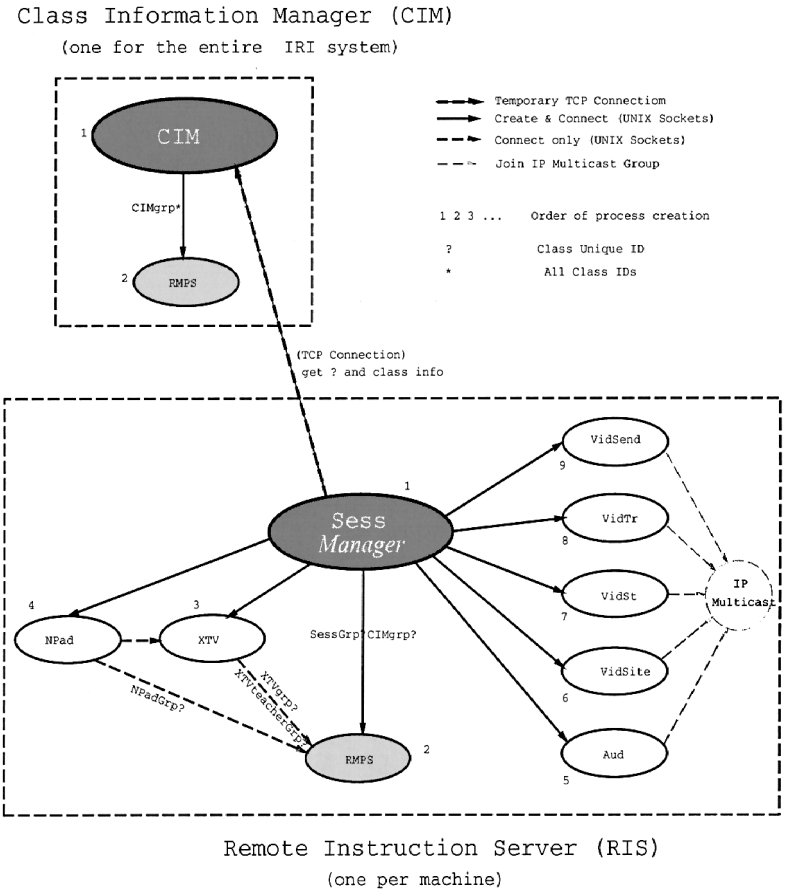

Fig. 1. System architecture.

IRI system and there is one RIS for each participant. The CIM system is the heart of IRI that glues everything together. It assigns unique IDs to classes and participants and maintains the attendance lists of all on-going classes.

RIS consists of several autonomous processes that use UNIX-domain sockets [4] for communication within the same host. On different machines, processes communicate using raw IP multicasting and the reliable multicast protocol RMP, developed at UC-Berkeley/West Virginia University [5]. RIS has the following set of processes: Session (*Sess*), X Tools sharing (*XTV*), Audio (*Aud*), Video capture (*VidSend*), Video student receiver (*VidSt*), Teacher video receiver (*VidTr*), Class video receiver (*VidSite*), Reliable Multicast Protocol Server (*RMPS*) and Notepad (*NPad*).

The Session process manages the class, multimedia resources, and the IRI interface. It also creates and initializes all the other processes.

The X Tools sharing process allows sharing X applications between teacher and students. The application usually runs at the teacher machine but can be run on any accessible machine. *XTV* intercepts X application requests and sends them to the local *X* server and to the students' X servers [6, 7]. The interaction with the application is controlled by a token. Only the person that has the token (token holder) can interact with the tool (this can be the teacher or any one of the students).

The Audio process handles voice data. In IRI there are several audio channels. One is reserved for the teacher to speak at any time, and the others are to be shared by all students. Since workstations may be in different rooms, in each room one machine is designated to play out the teacher's voice and the speaking students' voices using a software audio-mixing technique.

The Video processes capture and display the teacher's, students' and the classrooms' images. The *VidSend* process captures and sends the image of the teacher (at a rate of 30 frames/sec) or a student (a rate of 10 frames/second). The *VidTr* process displays the teacher's image in a 640*480 window if no X application is running and shifts the image into a 320*280 window if an X application is started. A *VidSt* process receives a student image and displays it on the screen in a 320*280 window. At one moment, images of two students can be displayed. Each room has a workstation designated to capture a general image of that site. On this machine runs another *VidSend* process that captures the image of the site at a rate of 5 frames/sec. The *VidSite* process receives and displays a site image in a 320*200 window. The architecture also allows to monitor network conditions and adjust the frame rate of individual video streams.

The *RMPS* process uses the services provided by the RMP protocol for reliable multicast. Whenever the *Sess*, *XTV* or *NPad* process want to communicate with their counterpart on another workstation, they deliver the data to the *RMPS* module. This multicasts to the *RMPS* modules running on all stations, which in turn deliver the data locally to the destination processes.

The Notepad process provides students with the utility of taking notes during the class. Any window on the screen can be snapped and introduced in a page on the notepad. The Notepad is used here only as an illustration for IRI utilities which support the learning environment: Presentation tool, Instant survey, and Lesson planner [2].

3. IRI INTERPROCESS COMMUNICATIONS

Each process inside IRI has a well defined and expandable interface for communication. We have the following naming convention for messages and

functions. If a process X sends message M to a process Y, the message is given the name X to Y–M. In process X the function to send the message is called ToY–M, and in process Y the function to receive the message is called FromX–M. Thus, IRI can be completely defined by listing all messages and their associated functions.

3.1. UNIX Domain Protocols

As shown in Fig. 1, CIM is composed of two processes while RIS has several processes. Processes within CIM or within the same RIS communicate with each other using UNIX Domain Protocols [4]. The UNIX domain protocols are often twice as fast as TCP/IP since the data never leave the host and, therefore, no checksum or sequence numbers are needed to guard against data corruption, data loss, or out of order arrival.

More important to application programmers is that the API (Application Program Interface) of the UNIX domain is almost identical to the popular TCP and UDP sockets. While other forms of IPC (e.g., shared memory) are universally considered more efficient than the UNIX domain protocols, they usually use completely different APIs which do not interact nicely with other forms of I/O [4]. In Berkeley-derived UNIX, a pipe is implemented as a UNIX socket pair. In the X window system if an X client determines that the X server is located in the same host as itself, a UNIX domain socket is used instead of a TCP socket.

3.2. The Reliable Multicast Protocol

RMP was developed to provide reliable multicasting over the Internet. We have adopted RMP as the basis for inter-site communications between processes that need reliable data transmission (e.g., X protocol data). In IRI three types of traffic need reliable communication:

T1: one-to-many, e.g., the output (requests) of a shared X application (tool) to the X servers of all participants.
T2: many-to-one, e.g., the input (events) to a shared X tool from the X servers of all participants.
T3: one-to-one, e.g., a teacher having a private conversation with a student.

RMP is ideal for type T1 traffic. For example, IRI creates a multicast group called *XTV-group* and each student joins this group. Any message sent to *XTV-group* will be received by all members of the group.

TCP connection is ideal for type T2 traffic. However, for the reasons explained in section 2 (scalability), we use RMP to carry T2 type traffic, by creating a multicast group called *TeacherGroup*. The only member of the *Teacher-*

Group is the teacher. Each student may send messages to this group without being member of it.

To handle the third type of traffic T3, we may consider one of the following three options:

Option 1. The teacher uses a multicast group G to send a message and explicitly specify that the message is intended for a specific student. All other students should ignore the message.

The disadvantage of this option is clearly the overhead involved in sending the message to all students since all but one will ignore it. This option is appropriate if the teacher intends to send few messages to one student.

Option 2. The teacher uses a multicast group G to inform a specific student to join another private multicast group P. The teacher also joins P and therefore both the teacher and the student use P for their private conversation.

The advantage of this option is that it can be generalized for "side chatting" between any subset of participants, not just two. A participant may create and join a private multicast group P and use the public multicast group G to inform a subset of the participants to join P.

Option 3. The teacher uses the public multicast group G to inform a specific student to "call back" and establish a TCP connection to use for their private conversation.

This is a more efficient than Option 2 if only two parties are involved in a private one-to-one conversation.

To conveniently use the services provided by the RMP library, we have implemented an RMP Server (RMPS). As an example, an XTV process may "join" an RMP multicast group G by sending join request to RMPS. The convention used to name an RMP group G is *X.cs.odu.edu* where *X* has the syntax ⟨*process–name*⟩*grp*⟨*Class ID*⟩. For example, all *XTV* processes of a class whose ClassID is *5* should join the RMP group: *XTVgrp5.cs.odu.edu*. The Class ID is a unique number assigned by the CIM upon the start of each class.

For each UNIX connection, RMPS creates a thread to handle the connection. The messages exchanged between a process P and an RMPS thread D are:

Join group. P asks D to join group G.

Leave group. P asks D to leave group G and close the UNIX connection.

Send message. P sends message to group G.

Receive message. D delivers a message to P that was sent by a process to group G.

List members request/reply. P sends a request to D for listing the members of group G. D replies by sending the required list.

Member left. D informs P whenever a member of group G leaves.

Slowdown. D informs P to slow down and stop sending messages because of a transient congestion.

Speedup. D informs P to resume sending after, say, a congestion condition is resolved.

3.3. UDP Multicasting

Because of the overhead involved in using RMP, we use UDP multicasting for sending continuous media streams. To choose IP multicast group addresses and port numbers we follow a scheme derived from Ref. [8]. Each group type (e.g., VidTrGrp used for teacher video) is assigned a unique number n (e.g., *VidTrGrp* used for teacher video is assigned a number $n = 1$). A specific group N? is mapped to an IP multicast group G according to the following formula:

$$G = 224.A.B.C : P$$

where $A.B.C.D$ is the CIM host IP number and P is the UDP port calculated according to the following formula:

$$P = 40,000 + ((? - 1) * 30) \; mod \; 10,000 + n$$

Where ? is the unique Class Id assigned by CIM upon the start of a class. Here we assume that each class may have at most 30 IP multicast groups. The number 10,000 is to ensure that P will not exceed the range of port numbers (about 50,000 on most implementations).

4. THE CLASS INFORMATION MANAGER (CIM)

Figure 2 shows the major data structures of CIM. The CIM maintains a table of ClassInfo. Each entry in this table represents either a regularly offered course or a temporary study group. A "course" is a static entity that represents a course to be offered as an IRI course. Normally the information about IRI courses are inserted into the table at the beginning of each semester and usually are deleted at the end of the semester. Therefore the course information is considered to be long lived. For example, in Fig. 2, the ClassInfo table contains two courses CS 476 and CS 250. Each course has a set of fixed attributes such as the course number, the course title and teacher information. For example, the first entry is for CS476—Systems Programming which is taught by Dr. Wahab.

Besides regularly offered courses, students (either only by themselves or with the help of a teaching assistant or a teacher) may form IRI "study groups". From the CIM point of view, a study group is considered to be a "temporary" course that exits only as long as there is at least one active participant and is deleted as soon as the last participant leaves the study group. In contrast to the regular course information, a study group information is considered short lived.

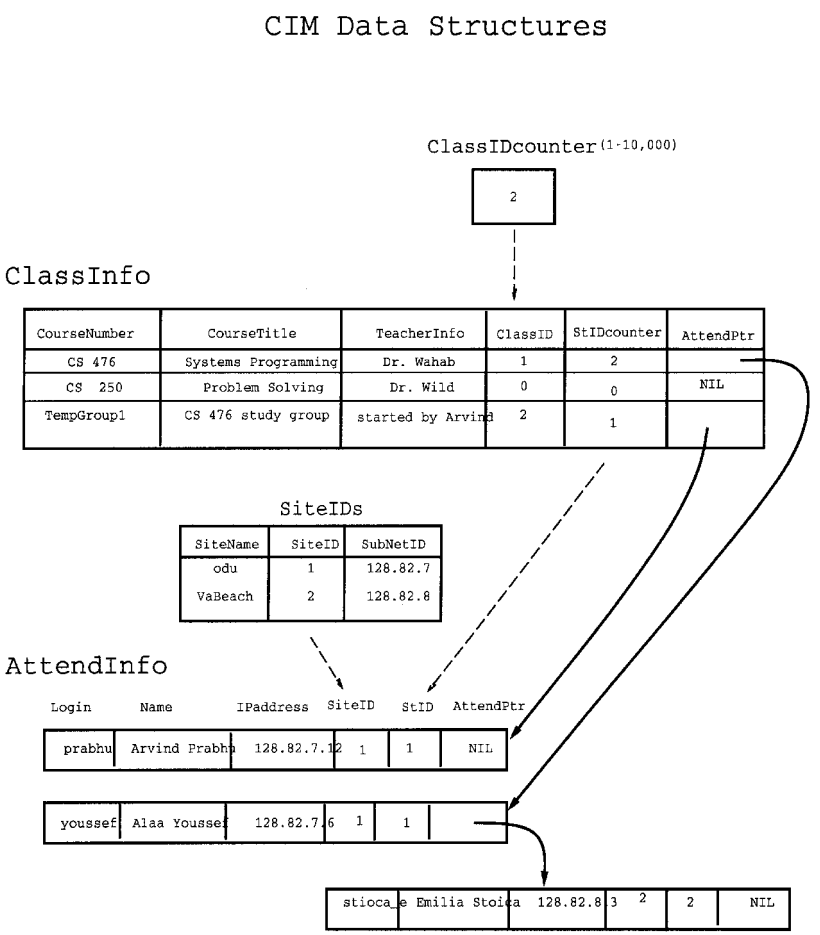

Fig. 2. CIM data structures.

In Fig. 2, the ClassInfo table has one study group called TempGroup1 for course CS476 and it has been started by a participant whose name is Arvind. In CIM, we deal with a study group exactly like a regular class except that the information about the group is deleted at the end.

Each class or study group is assigned a unique identifier called ClassID. In Fig. 2, the variable ClassIDCounter keeps track of the last assigned class numbers. The value of this variable ranges from 1 to 10,000 (this value is practically a very large number since it is unlikely to have 10,000 simultaneously active classes and study groups at any given university). In Fig. 2, course CS 476 is an active ongoing class whose ClassID is 1 while the study group TempGroup1

has a ClassID of 2. However, the ClassID of course CS 250 is 0 since there is no class held for this course at this instant of time. The classID is used as the basis for creating unique multicast groups within the IRI system.

In addition to unique ClassIDs, CIM assigns to each IRI class room a unique site ID and a set of Subnet addresses. For example, at this time, IRI have two class rooms, one is located in the main campus in Norfolk while the other is located 15 miles away in Virginia beach. The information about these two sites is shown in a table called SiteIDs in Fig. 2. The Norfolk site (odu) is assigned SiteID 1 and its subnet IP number is 128.82.7 while the Virginia Beach site (VaBeach) is assigned SiteID 2 and its subnet IP number is 128.82.8. It is easy to identify the site of each participant from the IP address of his/herworkstation. For example, in our current configuration, all workstations at the odu site have an IP address of the form 128.82.7.X and all workstations at the VaBeach site have an IP address of the form 128.82.8.X. The information in the table SiteIDs is also long lived since it rarely undergoes any changes. For example, a new record is added in this table whenever a new IRI class room is operational, an event that occurs infrequently, e.g., every few months.

The last type of information kept by the CIM is the attendants (participants) list for each class. Each attendee (or participant) is identified by a unique ID (StID) within the class. In each class or study group there is a counter called StIDcounter that keeps track of the last assigned StId. The participants of each class are kept together in a linked list of AttendInfo records. Each AttendInfo record contains vital information about each participant such as login, Name, workstation IP address, site ID and StID. In Fig. 2, the class CS 476 has two students: youssef (StId is 1 and is located at odu site) and stoic–e (StID 2 and is located at VaBeach site). Note that CS 250 currently has no active classes and therefore its ClassID is 0, its StIDcounter is 0 and its AttendPtr is NIL.

CIM always saves the changes made to its data structure in a stable storage. If for any reason, the CIM process is abnormally terminated, any new Sess process will detect this failure and will try to restart the CIM process. When the CIM process restarts after failure, it recovers the contents of the data structures from the disk files and verifies that the information it has is still current by contacting the RMPS. This is achieved by reconciling the differences between the data structures information and the RMPS information concerning the current multicast groups and their members.

5. MANAGING MULTIMEDIA RESOURCES

We define an exclusive multimedia resource to be a resource that can be held and used by at most one participant at a time. Each resource is associated with a token. Only the participant that has the token can use the resource. The following is a list of the exclusive multimedia resources mostly used in IRI:

Input to Shared Tools (*ToolToken*). Allows a participant to manipulate and provide input to the shared tools.

For simplicity, we have a single token for all tools. This is in contrast to XTV [6, 7], where each tool has its own token and thus a participant may control a subset of the X tools. From our experience in both XTV and IRI, the policy of allowing different users to simultaneously control different subsets of tools can be confusing and difficult to manage.

Global Pointer Control (*PointerToken*): Allows a participant to move the global pointer across the windows of the shared tools.

Student Audio Channel (*AudioToken*): Allows a student to speak and be heard by all class participants.

Student Video Channel (*VideoToken*): Allows a student video to be captured and sent to all class participants.

Center of Attention (*AttentionToken*): this token combines the *AudioToken* and *VideoToken* to allow the student to be the center of attention. This type of resource is considered to be a logical rather than a physical resource (like Video and Audio).

Public NotePad (*NPadToken*): Allows a participant to show his/her notepad to other participants.

Create Tools (*CreateToolsToken*): Allows a participant to run new tools locally and remotely. Usually the teacher retains this token, and it is rarely given to students.

5.1. Management of Exclusive Resources

The problem addressed here is how to manage and control access to the exclusive resources of IRI. Associated with each resource is a token and a participant must first obtain and hold the token before he/she can access the resource. Most earlier implementations of exclusive resource management are based on centralized server architecture [6, 9]. Here, we present a new implementation in which the central server functions are distributed among all participants. Hence, there is no single point of failure or bottleneck which results in a more robust and responsive system. In our algorithm, we rely on the fact that the transport layer (RMPS) has an implementation of reliable multicasting where a message sent by one participant is guaranteed to be received by all other participants.

5.1.1. Classification

Resources have *types* and each resource type has one or more *units*. Resources are classified into two categories: *primitive* and *composite*. A primitive resource is atomic in the sense that it is not composed of other resources while a composite resource consists of a set of two or more primitive resources. In multimedia collaborative applications, some participants have distinguished

roles. For example, in a desktop conferencing system such as XTV [6] one participant is designated as the conference *chairperson* and in a distance learning system such as IRI one of the participants is designated as the *teacher* and the rest of participants are *students*.

Shared Tools, *Global Pointer*, and *Slide Tool*, are examples of IRIs primitive resources. Each of these types has one unit. *Audio Channels* and *Video Channels* are also primitive resources, but each of these types has multiple units.

Examples of IRIs composite resources are the *Center of Attention* resource which has multiple units, each composed of an audio and a video channel, and the *Presenter* resource which has only one unit and is composed of slide tool, global pointer, an audio and a video channel.

5.1.2. Access Permissions

If a participant wants to speak, ideally he/she should be able to start speaking right away without first asking for permission to speak or wait for the availability of an audio channel. However, if a participant X wants to manipulate a shared application, like a spread sheet, which is being manipulated by some other participant Y, then it is appropriate to get the following permissions before this switch from Y to X is completed. First, X should ask the coordinator (e.g., conference chairperson or the class teacher) for permission to manipulate the shared application. If the coordinator approves X's request, then Y may be asked to stop what he/she is doing and be prepared to relinquish control of the shared application. Even when Y signals his readiness to pass the token, we may still have to wait for the shared application itself to reach a "consistent" state before giving the control to X. This scenario, depicts that some resources may require the approval of three entities before granting them: the coordinator (c), the current resource holder (h) and the switched resource itself (r).

Based on this brief discussion, we associate with each resource type three permission bits, if one bit is set then the corresponding permission must be obtained before a unit of this resource type can be granted. In representing these permissions we use notations similar to that used to represent file permissions in UNIX, where a file may have three primitive permissions (rwx) for read/write/execute. We use (chr) for coordinator/holder/resource. For example, (-h-) means that only the holder permission is required and (ch-) means that both the coordinator and the holder permissions are required.

5.1.3. RMP Messages

We use only two RMP messages, one to request the resource and the other to grant the resource. We assume each user has a unique identifier (called UID) and each participant maintains a list of all other participants UIDs. The RMP message *RequestResource* is sent by the participant requesting the resource. The message has four arguments: the sender's UID, the requested resource ID, the Rsrv field (for dealing with composite resources as will be discussed later) and the resource

permission bits. The current resource holder process responds to this request, after the needed permissions have been obtained, by sending a *GrantResource* message which has three arguments: the new holder's UID, the granted resource RID and the Rsrv field.

5.1.4. Free Resource Holder

One of the participant's processes, called the *free resource holder*, keeps the resource if no one else needs it. We refer to the UID of this participants as the FreeID. FreeID is always chosen to be the lowest UID of the current active participants. In order to distinguish between the FreeID and the actual participant having the same UID, the FreeID is preceeded by a minus sign. Thus if the lowest UID is 14, the FreeID is −14. Note that granting a resource to FreeID is equivalent to freeing the resource. Each participant can always compute the value of FreeID since we assume each has an updated list of the current participants. It is important to note that the FreeID is not tied to any particular participant, but it is assigned to the lowest numbered UID among the current participants. Thus if the current FreeID leaves the conference, a new FreeID takes over, and if a new participant joins the conference with a lower UID than the current FreeID, then that new participant takes over and becomes the new FreeID.

5.1.5. Requesting/Granting Composite Resources

The *Rsrv* (reserve) field of the message is used for dealing with composite resources. Assume a user is granted a composite resource unit with RID r and assume the composite resource consists of the primitive resources $r_1, r_2, \ldots, r_n$. To get one unit of each primitive resource, they should be collected, one-by-one, and none of the collected units should be released until the composite resource itself is released. In such case, the primitive resources obtained as part of a composite resource should be reserved and should not be available for reallocation until the composite resource is freed. The Rsrv field of the messages requesting and granting units of $r_1, r_2, \ldots, r_n$ should have the value r, indicating that this unit is reserved to be part of the composite resource r. If the value of Rsrv field is zero, then the resource unit is being allocated as a stand-alone primitive resource unit. Thus if a process notices that the Rsrv field of a grant message is non-zero, it marks the resource unit as reserved and should refrain from requesting these units as if they do not exist.

Accordingly, each resource unit has one of three states: *free*, *used*, and *reserved*. Both free and used resource units can be granted to users according to the resource permission bits. However reserved units can not be requested or granted by any process. If the composite resource r is being used by one user A, then switching r to another user B implies that all the components of r should be granted to B as reserved units. On the other hand, if r is freed by A (by granting it to FreeID), then its primitive units should be freed as well. Therefore, a composite resource is "assembled" with reservations if it is granted to user A

by FreeID, is "dismantled" when it is granted to FreeID, and is "preserved" if it is switched from one user A to another user B.

6. AUDIO AND VIDEO RESOURCES

It became most obvious that two features are crucial in a computer based learning paradigm: responsiveness and robustness. Responsiveness means both performance and ease of use. Robustness means that the failure of one system part should not affect the others. In this section, we will describe the services provided by the audio and video processes to support these requirements.

6.1. Audio

As mentioned earlier, the teacher can speak at any time. The teacher's voice is sent to the *AudTrGrp?* multicast group. The symbol "?" stands for ClassID, which is a unique identifier for an IRI session. Since in each room, only one workstation plays the teacher's voice, at most one *Aud* process per site joins this group. At the teacher's site, no *Aud* process joins *AudTrGrp?* since students can hear the teacher without a speaker. Students share together a group of audio channels, each controlled by a token. Thus, only students who have audio tokens can speak. The student's voice is sent to the multicast group *AudStXGrp?*, X = 1 ..., N_{aud}, where N_{aud} is the number of audio channels available to students. As for the teacher, at most one workstation plays the student voice and thus one *Aud* process per site joins this group. The number of audio channels should be sufficient in order not to block any student who wants to speak. Each room has a workstation designated to play the voice coming from all the audio multicast groups, using a software mixing technique.

6.2. Video

The video subsystem has the task of displaying multiple video streams. As presented in Section 2, it consists of the following processes: *VidSend*, *VidTr*, *VidStX*, *VidSite*. The *VidSend* process captures a video image from a local camera. This image can be a teacher's image or a student's image. The teacher's image is sent to *VidTrGrp?* multicast group. Each *VidTr* process joins the *VidTrGrp?* and thus receives the teacher's image and displays it in a window. As mentioned earlier, if a student is at the local workstation, *VidSend* captures the image of the student and sends to one *VidStXGrp?*, X = 1 ..., N_{vid}, where N_{vid} is the number of student video channels. The processing time for decompression and display of a frame limits us to display the image of only two students at a time. The image of the first student will be multicasted to *VidSt1Grp?* and the image of the second student will be multicasted to the *VidSt2Grp?*. The *VidStX*

process, X = 1, ..., N_{vid}, joins the VidStXGrp?, receives the image and displays it in a window. On each workstation, there is one *VidStX* process for each multicast group *VidStXGrp?*. In each room, there is a designated workstation where an additional *VidSend* process runs. This machine has a supplementary video board attached to a camera that captures a general image of the site. This image is sent to the *VidSiteXGrp?*, X = 1 ... N_{site}, where N_{site} is the number of sites. The *VidSite* process receives and displays the frames in a window. *VidSite* joins only one of the multicast groups *VidSiteXGrp?*. Thus, at any instant in time, all the students and the teacher will have the image of the same site on the screen. The teacher can change the image to display another site. In this case, the *VidSite* processes will join another multicast group and will display the other site image.

During the class, the teacher's image may be shifted from a big window (640*480), when no tool is running, to a small window (320*240), when an X application is started. We apply this also in the case of a student making a presentation, where he switches the role with the teacher. The *VidTr* processes will receive a message from the *Sess* process, which causes the teacher's image to be displayed in the smaller (320*240) window. The *VidStX* processes that displays the presenter's image, will receive another message, from the *Sess* process, which causes the student's image to be displayed in the 640*480 window.

7. THE TOOL SHARING ENGINE

The tool sharing in IRI is handed by the *XTV* process as shown in Fig. 1. Although XTV [6, 7] is a general purpose collaborative system designed to share X applications in a desktop conferencing environment, we retain the name XTV in IRI to refer to those parts that handle the mechanics of sharing X tools.

In XTV, a token is used to coordinate control of shared tools. Without this control, shared tools may reach inconsistent states and crash. Only the participant with the token is allowed to change the state of the tools. In XTV, a tool may be created *locally* on the same machine where the *XTV* process of the teacher is running or *remotely* on any other machine. In either local or remote execution of a tool, whenever the tool establishes a TCP connection with the teacher *XTV*, the teacher *XTV* sends an RMP message to all other *XTV* processes to add a new tool structure. The new tool structure will contain all the information needed for traffic translation between the local and teacher displays. See Ref. [7] for details of this translation.

Since the tools are connected to the teacher's *XTV* process, all tool requests are intercepted by this process and sent to all X servers. The events generated by an X server are sent to the teacher's *XTV* only if the participant has the tools token or the event is an *expose* event [10]. We always let the expose events to be sent immediately to the tools, otherwise some participants may have to wait

long times to refresh their display windows. The teacher's *XTV* process forwards the received events to the tool based on the assumption that at most one of the participants has the tools token. Every *XTV* process checks to see if it has the tool token before forwarding any packets to the teacher's *XTV*.

8. CONCLUSIONS

The advent of digital technology to support interactive, multimedia virtual classrooms at multiple, distant locations is an important step in solving the problems of delivering education and training efficiently and effectively to students who are bound to specific locations by jobs or lack of travel and relocation funds. IRI is a system which implements a virtual classroom to support such a distance learning environment. IRI is an open architecture with a specification of its services and communication protocols available; the multimedia subsystem and its impact on the new learning paradigm were the driving force behind the design of the IRI system and lead directly to the process structure described in this paper.

To manage such large and complex distributed multimedia system effectively, we implemented each functionality into a separate process. In this way, one failure does not cause failure in other subsystem parts. We use reliable and unreliable multicast communication between processes on different machines. Thus, IRI can support any number of users.

In IRI, multicasting is not only used for performance reasons, but is also used for management of the multimedia resources. The ability for any process to join and leave any multicast group proved to be one of the most valuable and powerful feature of IGMP (Internet Group Management Protocol) [11]. For example, each remote site in IRI has a process that takes the class video and sends it to the multicast group assigned to that site. Then, a process that receives a remote site video can be instructed to display the images of any desired site by simply joining the corresponding multicast group of that site.

In addition, we have described a distributed algorithm to manage and control exclusive resources among IRI participants. The algorithm is based on RMP, an implementation of reliable multicasting, in which every message sent by one participant is guaranteed to be received by all other participants.

ACKNOWLEDGMENT

This work is supported by grants from NSF, Sun Microsystems and Cox Communications.

REFERENCES

1. K. Maly, H. Abdel-Wahab, C. M. Overstreet, A. K. Gupta, Mauthu Kumar, and Rahul Srivat-

sava, Issues in scaling multimedia collaboration tools for remote instruction, *Proc. IEEE Symp. Computers and Communications*, Alexandria, Egypt, pp. 42–48, June 1995.
2. K. Maly, H. Abdel-Wahab, C. M. Overstreet, and C. Wild, Telelearning: A new paradigm for interactive remote instruction, *Int'l. Distributed Conf. Teleteaching*, Madeira, Portugal, pp. 1–14, November 1995.
3. K. Maly, H. Abdel-Wahab, C. M. Overstreet, C. Wild, A. Gupta, A Youssef, E. Stoica, and E. Al-Shaer, *Distance learning and training over intranets*, *IEEE Internet Computing*, Vol. 1, No. 1, pp. 60–71, February 1997.
4. W. Richard Stevens, *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the UNIX Domain Protocols*, Addison-Wesley, 1996.
5. B. Whetten, T. Montgomery, and S. Kaplan, A high performance totally ordered multicast protocol, *Theory and Practice in Distributed Systems*, Springer Verlag LCNS 938, 1994.
6. H. Abdel-Wahab and M. Feit, XTV: A Framework for sharing X window Clients in remote synchronous collaboration, *Proc. IEEE TriComm '91: Communications for Distributed Applications & Systems*, Chapel Hill, North Carolina, pp. 159–167, April 1991.
7. H. Abdel-Wahab and K. Jeffay, Issues, problems and solutions in sharing X clients on multiple displays, *Journal of Internetworking Research & Experience*, pp. 1–15, Vol. 5, No. 1, March 1994.
8. S. Pejhan, A. Eleftheriadis, and D. Anastassiou, Distributed multicast address management in the global Internet, *IEEE Journal on Selected Areas in Communications*, pp. 1445–1455, October 1995.
9. H. Abdel-Wahab, B. Kvande, S. Nanjangud, O. Kim, and J. P. Favreau, Using Java for multimedia collaborative applications, *Proc. 3rd Int'l. Workshop on Protocols for Multimedia Systems (PROMS'96)*, Madrid, pp. 49–62, October 1996.
10. A. Nye, *X. Protocol Reference Manual for Version 11*, Volume 0, O'Reilly and Associates, Inc., Sebastopol, California, 1989.
11. S. E. Deering and D. Cheriton, Multicast Routing in Internetworks and Extended LANs, *ACM Trans. on Computer Systems*, Vol. 8, No. 2, pp. 85–110, May 1990.
12. R. Steinmetz and K. Nahrstedt, *Multimedia: Computing, Communications and Applications* Prentice-Hall, 1995.

Hussein Abdel-Wahab is a professor of computer science at Old Dominion University, an adjunct professor of computer science at the University of North Carolina at Chapel Hill, and a faculty member at the Information Technology Lab of the National Institute of Standards and Technology. His main research interests are collaborative desktop multimedia conferencing systems and real-time distributed information sharing. He received a Ph.D. in computer communications from the University of Waterloo.

Kurt Maly is a Kaufman professor and chair of computer science at Old Dominion University. His research interests include modeling and simulation, very high performance network protocols, reliability, interactive multimedia remote instruction, Internet resource access, and software maintenance. Maly received a Ph.D. in computer science from the Courant Institute of Mathematical Sciences, New York University.

Emilia Stoica is a Ph.D. candidate in computer science at Old Dominion University. Her research interests include synchronization of heterogeneous streams in multimedia systems, multimedia architectures and shared computer-supported workspaces. Stoica received an M.Sc. in computer science from the Politechnical Institute of Bucharest.

Alaa Youssef is a Ph.D. candidate in computer science at Old Dominion University. His research interests include quality of service support for distributed multimedia systems and computer-supported collaborative work. Youssef received an M.Sc. in computer science from Alexandria University, Egypt.