Report Number: WUCS-95-28

1995-01-01

Distributed Radiological MultiMedia Conferencing

Naeem Bari

Distributed Radiological Multimedia Conference (DRMC) is a collaborative imaging/multimedia conferencing tool which allows geographically separated physicians to confer over a shared projection radiograph. DRMC utilizes the advantages of high bandwidth and scalability offered by the new Asynchronous Transfer Mode (ATM) network technology. This application is customized for the high quality of displayed images and rapid response to user requests. It allows conferees to: share a common radiograph; each possess an independently controlled globally visible cursor; be able to point to and outline areas on the image to bring it to the other conferees' attention; and see and hear each... Read complete abstract on page 2.

Distributed Radiological MultiMedia Conferencing

Naeem Bari

Complete Abstract:

Distributed Radiological Multimedia Conference (DRMC) is a collaborative imaging/multimedia conferencing tool which allows geographically separated physicians to confer over a shared projection radiograph. DRMC utilizes the advantages of high bandwidth and scalability offered by the new Asynchronous Transfer Mode (ATM) network technology. This application is customized for the high quality of displayed images and rapid response to user requests. It allows conferees to: share a common radiograph; each possess an independently controlled globally visible cursor; be able to point to and outline areas on the image to bring it to the other conferees' attention; and see and hear each other via an independent video conferencing package.

Distributed Radiological MultiMedia Conferencing

Naeem Bari

WUCS-95-28

December 1995

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis, MO 63130-4899

Master's Project advisor: Jerome R. Cox, Jr.

DISTRIBUTED RADIOLOGICAL MULTIMEDIA CONFERENCING

Master's Project

by Naeem Bari

with Dr. Jerome R. Cox Jr.

Applied Research Laboratory and Electronic Radiology Laboratory

---

ABSTRACT:

Distributed Radiological Multimedia Conference (DRMC) is a collaborative imaging/ multimedia conferencing tool which allows geographically separated physicians to confer over a shared projection radiograph. DRMC utilizes the advantages of high bandwidth and scalability offered by the new Asynchronous Transfer Mode (ATM) network technology. This application is customized for the high quality of displayed images and rapid response to user requests. It allows conferees to:

- share a common radiograph
- each possess an independently controlled globally visible cursor
- be able to point to and outline areas on the image to bring it to the other conferees' attention
- see and hear each other via an independent video conferencing package

Key Phrases: collaborative imaging, multimedia conference, ATM, graphics techniques

Contents

Abstract **1**
Contents **2**
Figures **4**
Brief Introduction **5**
Benefits offered by DRMC 6

Existing Components **7**
Introduction to ATM 7
Introduction to the MMX 8
Introduction to Programmer's Playground 9
Introduction to Vaudeville 10

Project Components and their Relationships **10**
Construction of a Clinical Workstation 10
Overall Application Structure 12
Registration Server 12
Image Server 15
Alert Server 18
Conference Setup Program 18
Image/Cursor Sharing Program 19
Virtual Switch 21
Image Previewing/Registration Program 21
Vaudeville 23

Solutions to Select Problems **24**
Conference Scheduling Mechanism 24
Floor Control Scheme 25
Efficeint and Visible Overlay of Cursors over Image 25

Update and Future work **27**

Appendices **29**

Appendix A 29

Appendix B 33

Appendix C 37

Appendix D 40

Appendix E 42

Appendix F 46

Bibliography **52**

Figures

Figure 1 11
Figure 2 13
Figure 3 14
Figure 4 16
Figure 5 17
Figure 6 22

Brief Introduction

Distributed Radiological Multimedia Conference (DRMC) is a collaborative imaging/ multimedia conferencing tool which allows a physician sitting at his workstation to set up a multimedia conference among his colleagues. This multimedia conference is comprised of a switched video channel, a multiple source multicast audio channel for all participants and multiple point to point image channels displaying a radiological image to all the participants. Each doctor possesses his or her own cursor that they can use to point to or outline specific areas on the shared image to aid in discussions among the group.

The number of high bandwidth channels necessary to provide these capabilities can be supported only by the emerging Asynchronous Transfer Mode (ATM) network technology. DRMC utilizes this technology to its fullest extent and provide a means of investigating various issues regarding its use in distributed applications.

A package called Programmer's Playground is used for the communication between the end points. This package, developed at Washington University, allows all the details of across-the-network communication to be hidden from the programmer, allowing him to concentrate on the design of the project at hand. DRMC is also the first application to exploit the new virtual switch feature of Playground.

This DRMC application can be run from within the confines of the Image Viewing Application (IVA) or on its own, and uses TCP/IP connections over ATM (implemented by an Efficient Network, Inc. (ENI) host adapter card) or Ethernet.

Setting up such a conference requires the participating physicians to have a Sun Sparcstation, ENI

card, MultiMedia eXplorer (MMX), camcorder and an audio system.

DRMC is geared towards handling an arbitrary number of conferees.

Multiple conference scheduling mechanisms and floor control techniques were studied and the most user friendly ones were chosen and implemented. A lot of care was given to the development of a very user friendly interface for the physicians to use as well.

Benefits offered by DRMC

Three main benefits justify the development of this application:

- Cost efficiency
- Speed
- Customization

Cost Efficiency

When physicians want to consult over a patient's radiograph they currently need to be present physically at a common location with a copy on film. This is not a cost effective solution for doctors primarily because of their possible geographical separation, even if by only a few miles. The DRMC project solves this problem by displaying radiographic images on a window in a workstation. This also enables physicians to avoid the expense of a specialized radiological display.

Speed

Admittedly there are some good multimedia conferencing packages that are commercially available, even in the shareware and freeware domains. However, they all have the disadvantage of limited bandwidth with respect to this kind of an application. Even for LANS the usual data rates

available are between 1.5 and 10 Mb/s. Furthermore we envision multimedia conferencing between physicians separated by large distances. Current wide area technology provides extremely low data rates, insufficient for a reasonable multimedia conference involving medical images. The emerging ATM network would support the required data rates, even across the continent. A tool with the functionality of DRMC does not yet exist for ATM.

Customization

The demanding requirements of radiology are best served by an application that is optimized for rapid response to image requests and to high fidelity of the displayed images. Since the use of ATM lifts bandwidth constraints it is feasible to transfer images without any kind of compression, lossy or otherwise. This ensures image quality and also relieves the machines of the decompressing work, allowing cheaper computers to be used.

Existing Components

The following pre-existing applications and equipment were exploited to implement this project:

Asynchronous Transfer Mode (ATM) network
MultiMedia eXplorer (MMX)
Programmer's Playground
Vaudeville

Introduction to ATM [1]

ATM (Asynchronous Transfer Mode) is a virtual channel oriented cell switching network. It is an internationally standardized form of packet switching in which user data is carried in small 53

byte cells, where the first five bytes of each cell include a label which identifies the user channel to which the cell belongs. Communication over an ATM network takes place over virtual channels, which are typically established when some user application is initiated. The route for the channel is determined at setup time and remains fixed for the duration of the communication. Since data transmitted by users is typically much larger than 53 bytes, it is fragmented into 53 byte cells by the source's network adapter and reassembled by the receiver's network adapter. Host software is typically free to work with data units of arbitrary size. All cell level processing can be left to the network adapter.

A key objective of ATM network technology is to ensure consistent performance to users in the presence of stochastically varying traffic. This is necessary to provide adequate performance for many high speed applications (such as video) that require guaranteed throughput. This objective is achieved by selecting virtual channel paths according to the anticipated traffic and allocating the necessary network resources. For such applications users must specify, at virtual channel setup, the amount of network resources they require. It also means that if the required resources are not available a user's request could be refused by the network.

Introduction to the MultiMedia eXplorer (MMX) [2]

The Washington University MultiMedia eXplorer is a complete, host independent multimedia system capable of transmitting and receiving JPEG-compressed video, CD-quality audio, and high resolution radiographic images over the Washington University broadband ATM network. If the host is equipped with an ATM adapter card, normal network traffic can be supported via an ATM extension port on the MMX. The major components of the MMX are an ATMizer and three multimedia channels.

For transmission of video the MMX does the following steps:

- Digitizes an analog NTSC video signal into a 196Mb/s digital stream.
- Compresses this digital stream using JPEG into a 5 to 20Mb/s JPEG stream.
- Passes this stream to the ATMizer, which breaks the stream into 53 bytes cells.

- Transmits these 53 byte cells to the ATM network.

For reception of video the MMX carries out

- The above steps in reverse order to obtain an analog NTSC video signal.

Similar steps are performed for audio.

The MMX also offers an extension connection to the ATM network to support the transmission of data to and from the host machine.

Introduction to the Programmer's Playground [3]

I/O abstraction is offered as a new high-level approach to interprocess communication. Functional components of a concurrent system are written as encapsulated modules that act upon local data structures, some of which may be published for external use. Relationships among modules are specified by logical connections among their published data structures. Whenever a module updates published data, I/O takes place implicitly according to the configuration of logical connections.

The Programmers' Playground is a software library and run-time system supporting I/O abstraction. Design goals include high-level communication among programs written in multiple programming languages and the uniform treatment of discrete and continuous data types. Support for the development of distributed multimedia applications is the motivation for the work.

For our purposes, Programmers' Playground is a C++ support tool that supports transparent communication between our different modules running at various sites. We make extensive use of the PGaudio and PGvideo data types. We also use a new feature of Playground, the Virtual Switch, to simplify the connection making process.

Introduction to Vaudeville

This is the videoconferencing application developed at Washington University. It utilizes an MMX connected to the ATM network to transmit and receive JPEG compressed video and CD quality audio between all conferees. It features voice-switched floor control and the ability to join pre-existing videoconferences as well as launch new ones.

Project Components and their Relationships

The DRMC application consists of several components:

- Clinical Workstation
- Registration server
- Image server
- Alert server
- Conference setup program
- Image/Cursor sharing program
- Virtual Switch
- Image previewing and registration program
- Vaudeville

Construction of a Clinical Workstation (Client)

It is assumed that every physician will have a Clinical Workstation setup on his/her desktop. This setup is referred to as a Client and consists of the Image Viewing Application (IVA) (from whose confines DRMC can be launched), the DRMC conference setup module, the Image/Cursor sharing program, the Alert server and Vaudeville. The only optional component at this time is the Image previewing and registration program. The Client can be seen in greater detail in fig. 1.

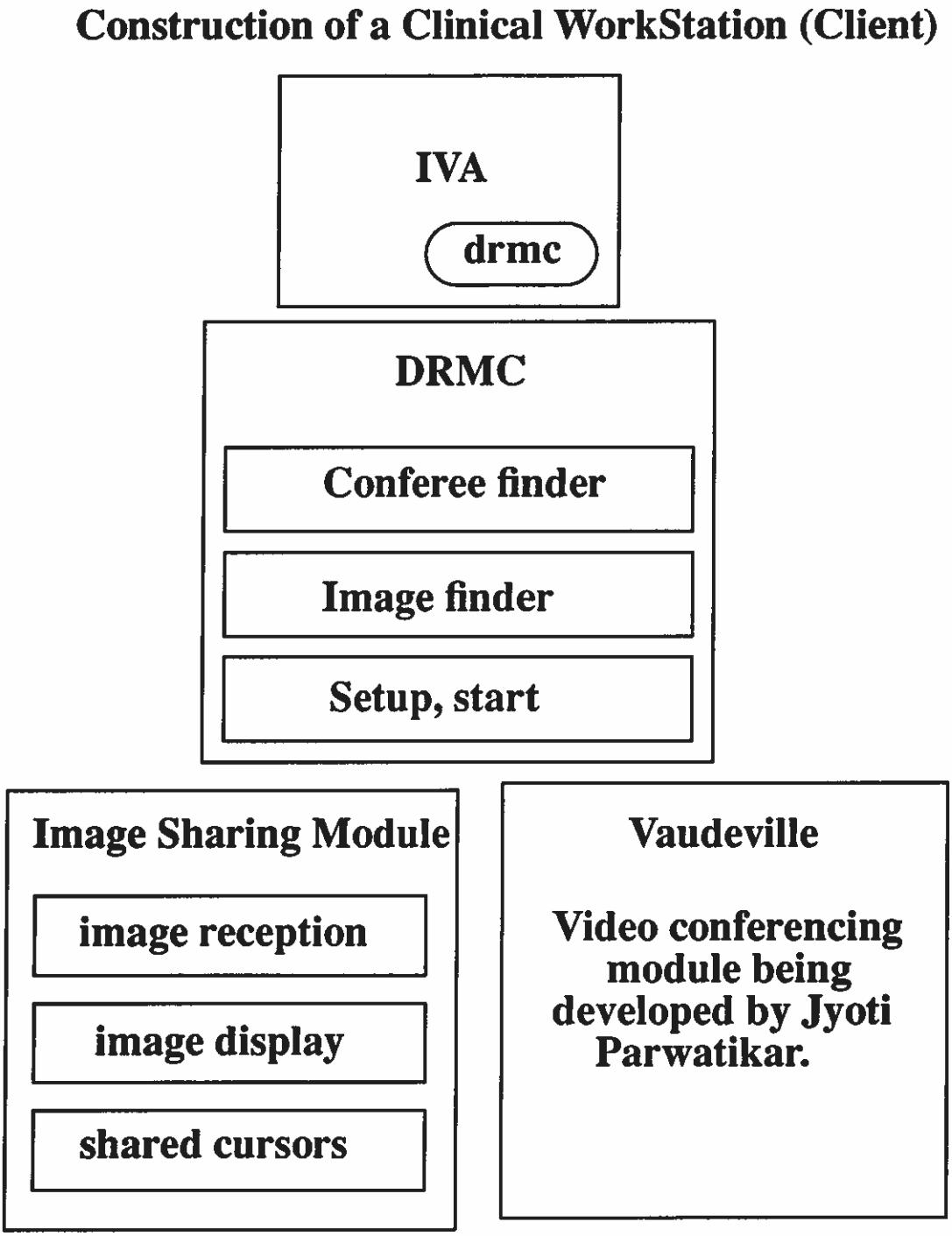

Construction of a Clinical WorkStation (Client)
IVA
drmc
DRMC
Conferee finder
Image finder
Setup, start
Image Sharing Module
image reception
image display
shared cursors
Vaudeville
Video conferencing module being developed by Jyoti Parwatikar.

Figure. 1

Overall Application Structure

Fig. 2 presents an overall view of this application. It outlines briefly the steps taken by one client to actually start a conference.

Registration server

This is a daemon that runs constantly at a universally known location. Its job is to maintain a list of clients able to take part in a conference. It keeps the following information about every client:

Name of user
Occupation
Employer
Machine Name
Email address
Status

The above record is referred to as a Conferee Details Structure (CDS). When a physician acquires the software and hardware capability for taking part in a conference he has to register his CDS with the Registration Server. Upon receipt a CDS is stored in a local file called "registeredconferees".

The Status field is a flag which is set to different values depending on the conferring capability of the client.

A query to the server consists of the first letter of the last name in a CDS. Upon receiving this letter all the CDS which match are sent back to the querying party.

See fig. 3 for the relationship between clients and the registration server. See Appendix B for implementation details.

Overall Application Structure

Here we see Client A initiating a conference

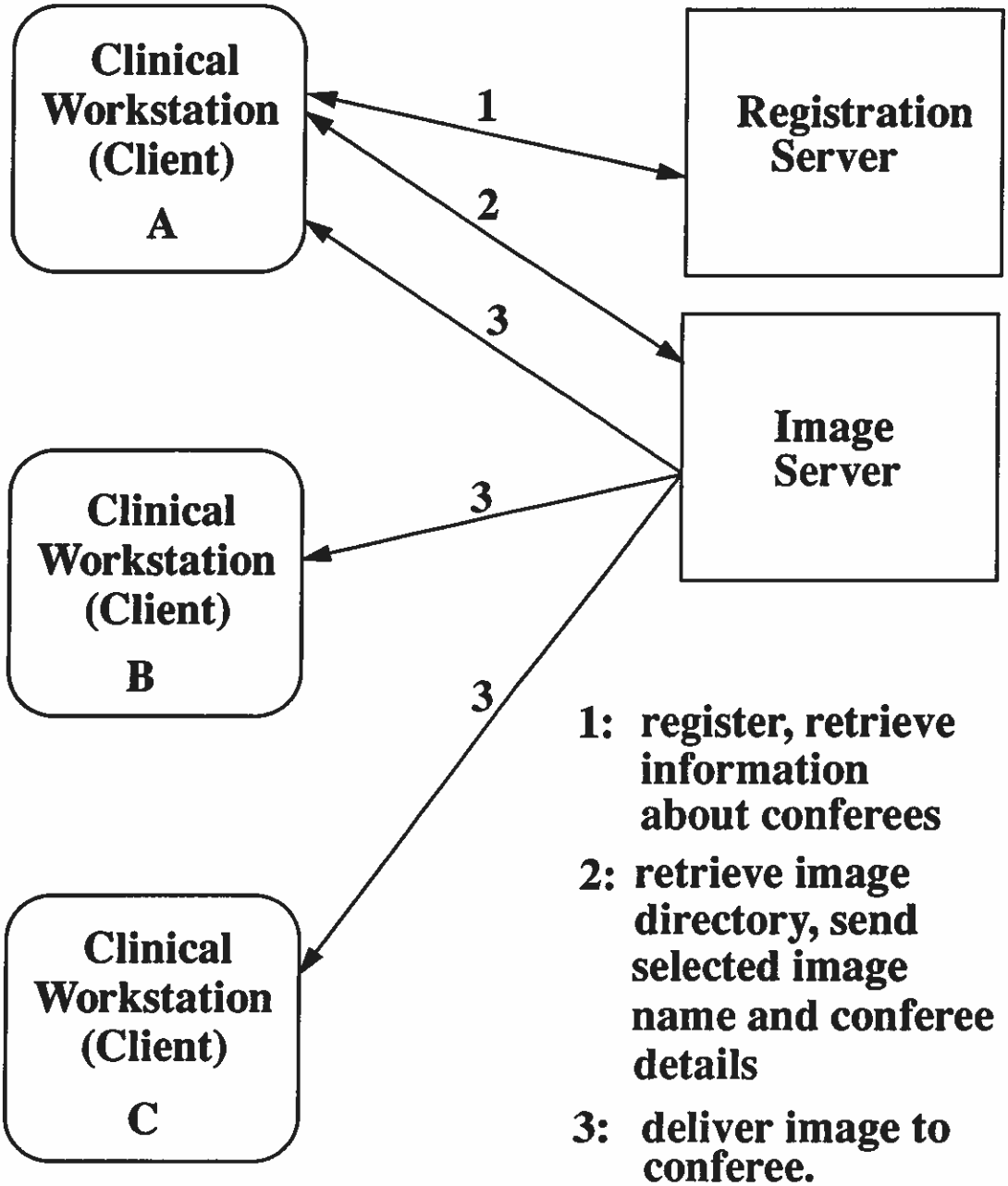


Figure. 2

Interaction of Client A with Registration Server

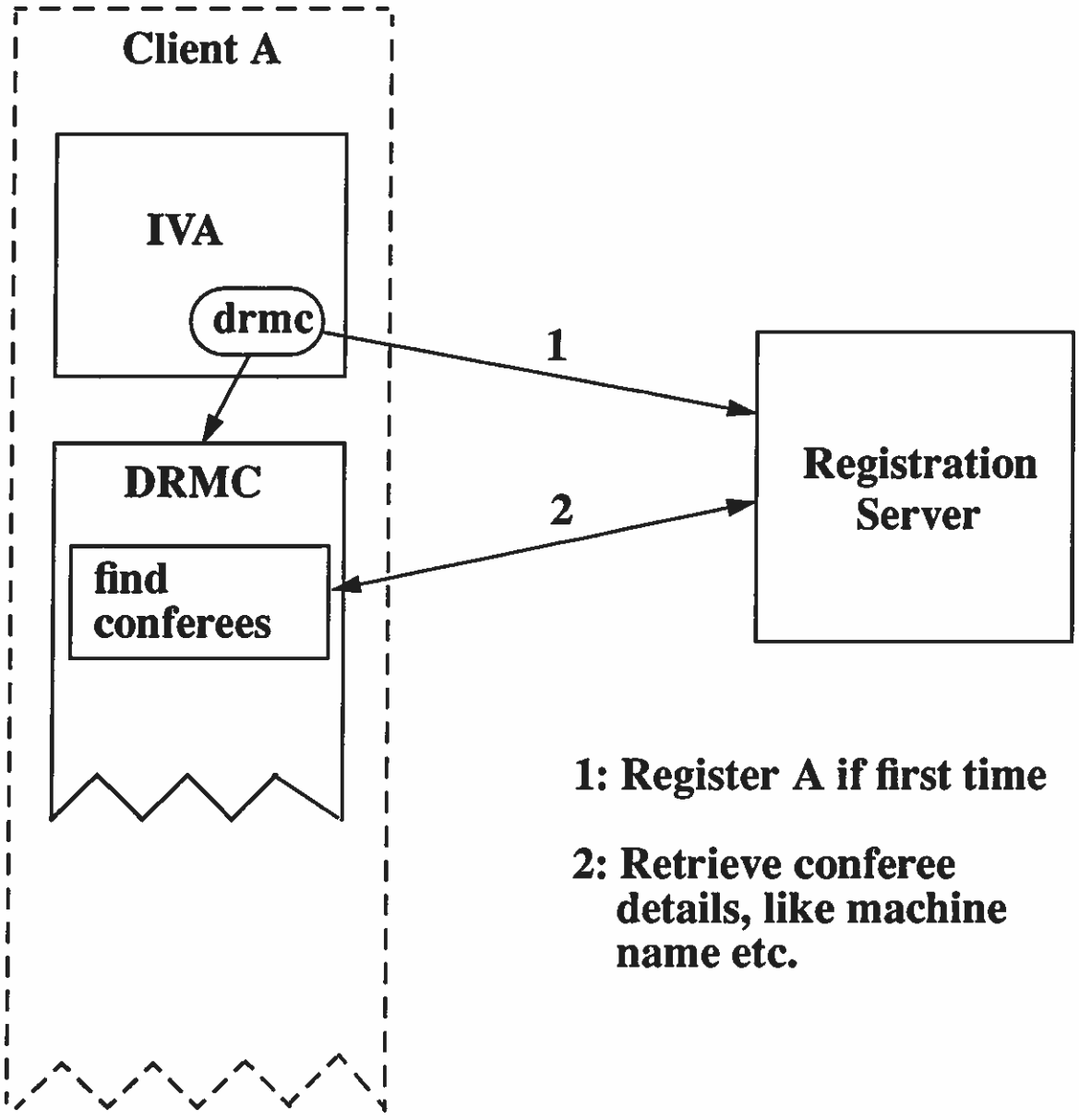


Figure. 3

Image Server

The Image Server is also a daemon that runs in a universally known location. Its job is to store images in a local directory and send them to machines when needed. A local file called "ImageDirectory" is maintained by the server. This file contains entries on every line which take the following form:

ImageName width height bits/pixel

This information is necessary for the recipient of the image (to be discussed later). This server accepts three kinds of requests:

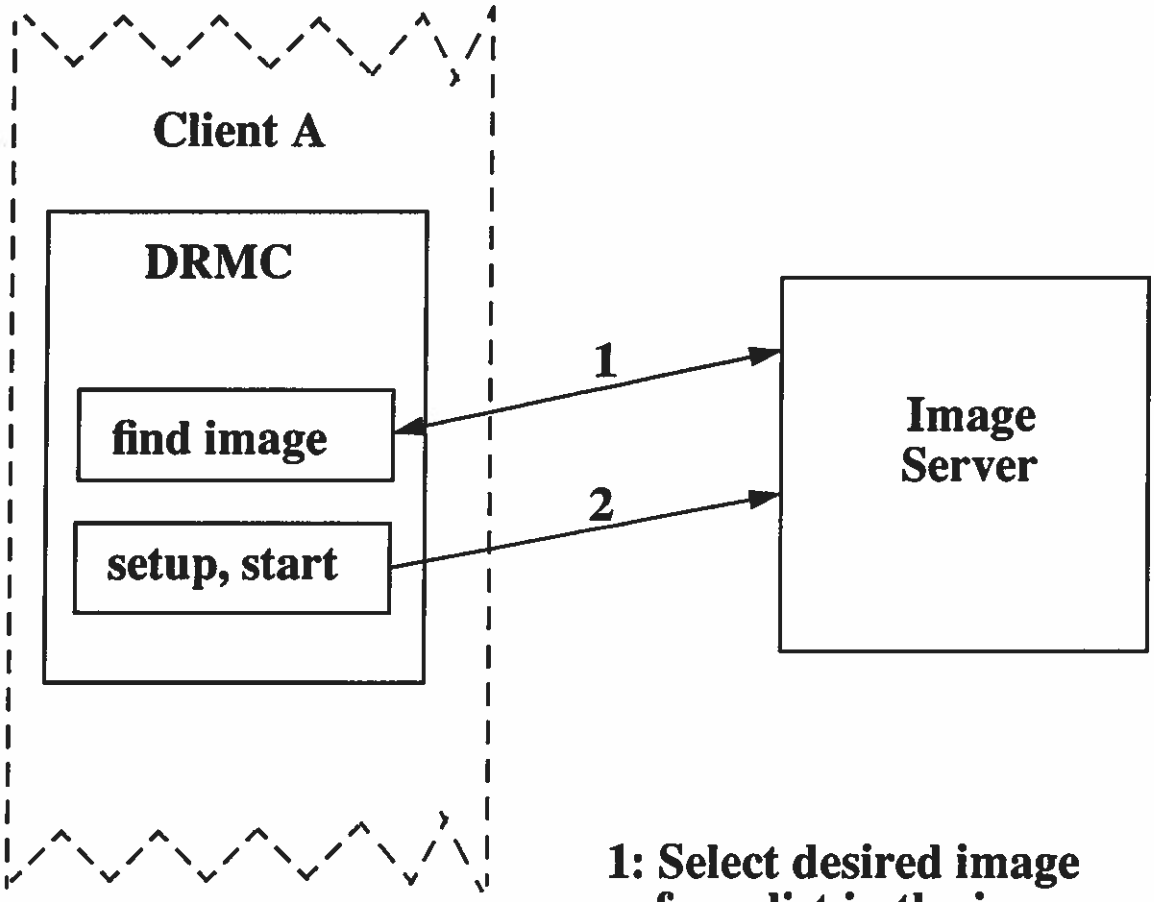
1) Directory retrieval: On receiving this query the image server simply returns the image names in the "ImageDirectory" file to the requesting party.

2) Image broadcast: The query structure consists of an image name followed by one or more machine names. On receiving this query the daemon loads the image from its local filespace and sends it out to the machines. In the current implementation this broadcast is done in a point to point fashion over any network which supports TCP/IP (LAN, N-ISDN, ATM in LAN emulating mode).

3) Image registration: Every physician has the ability to register an image (store the image in the central image database) with the Image Server from his/her local filespace so that the image may be used in conferences.

See fig. 4 for the interaction between a Client and the Image Server and fig. 5 for the interaction of the Image Server with a Client. See Appendix C for implementation details.

Interaction of Client A with Image Server



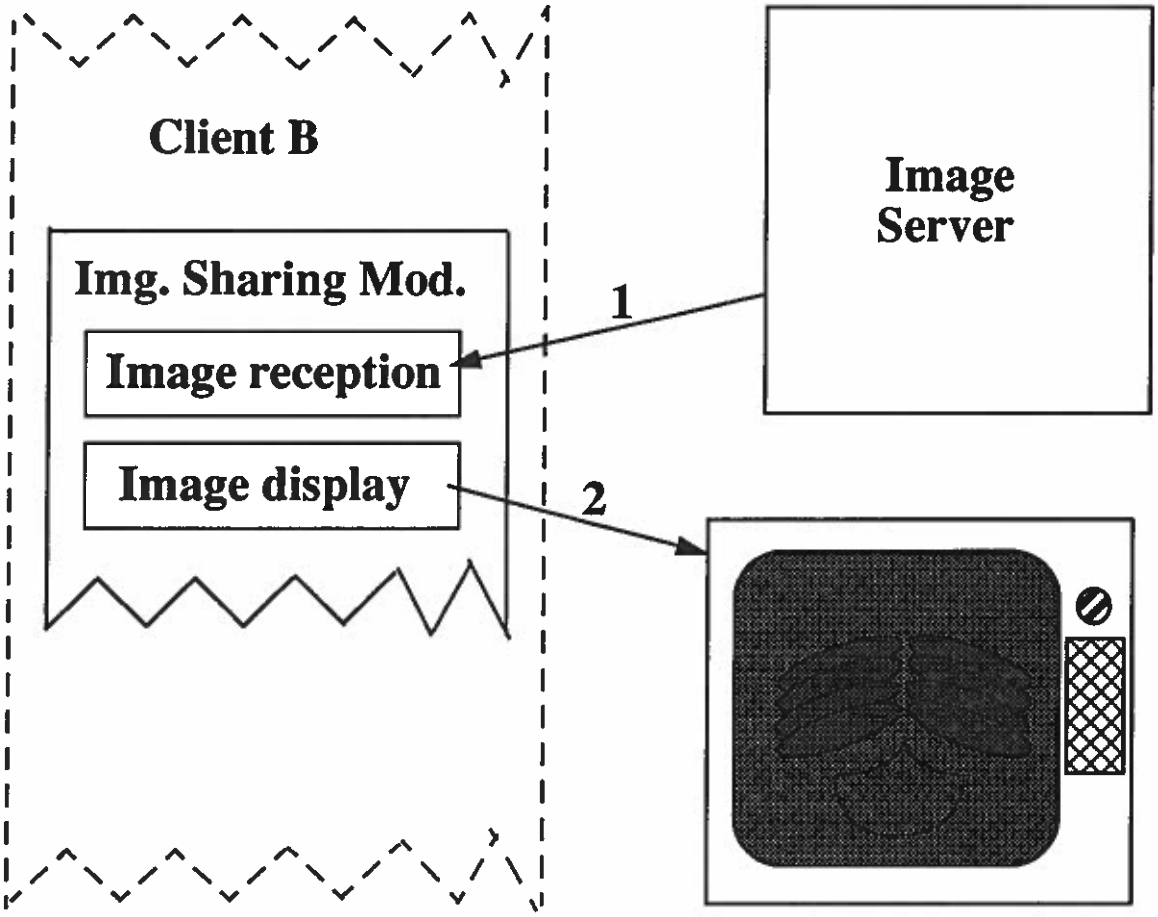
1: Select desired image from list in the image server

2: Supply image server with the machine names of conferees. These were obtained from the registration server.

Figure. 4

Interaction of Image Server with Clients

This is the interaction taking place between the image server and Client B. Similar steps are taken with all the conferees.



1: Image and Id sent.
2: Image displayed, either on a separate monitor (EVB) or in a window on the same monitor

Figure. 5

Alert Server

This is a simple daemon running in all the clients. It is attached to a specific machine and the physician using it. When a conference is called the alerter asks its associated physician whether he wants to participate or not. The reply is conveyed to the person trying to convene the meeting who may then take appropriate steps. If the physician accepts participation then it is the alerter's job to ready the local system, that is, to start up the required programs at the correct time and take other high level measures. Thus this daemon acts as the personal agent of the physician it represents. See Appendix D for implementation details

Conference Setup Program

This program is available at every client. Its job is to start a conference between physicians. The following steps are taken are taken by the setup module to accomplish its goal:

The physician types a letter in the "Name" field and presses return. The setup module constructs a request and sends it to the Registration server. The registration server returns all the entries which have the same first letter. The "hits" are displayed to the physician so that he may select one or more conferees. This process may be repeated till all the desired conferees have been selected.

Similar steps are taken to list the images available at the Image Server. One image is selected from this list.

Next, the desired conference time is entered. The current implementation supports a 24 hour wraparound time, that is, if the current time is 9:00 AM, and the desired conference time is entered as 8:00 AM, then the setup module will deduce that the conference is being scheduled for the next morning. On the other hand if 4 PM had been entered, the setup module would have understood that the conference was being scheduled for the same day.

Once the conferees, the image and the time have all been selected, the physician can now press the "Start" button. The setup module immediately contacts the alert daemons of all the clients, requesting the participation of their associated physician. The alert server pops up a small window with the message "You are being invited to a conference being scheduled by so-and-so at such-and-such time" along with Accept/Decline buttons. If the physician declines, no further action is taken with this person. If he/she accepts, the alert program stores a local flag and sends an acceptance message back. If no button is pressed, a built-in timer automatically expires (the duration of this timer can be set when the alert daemon is launched) and a default of 'Accept' is assumed. This default may be changed to 'Decline' based on feedback.

After the conference setup module has resolved all the accept/decline messages received from all the alert daemons it then goes to sleep till sixty seconds before conference time. The same steps are taken again, except this time when a physician selects "Accept" or his timer expires, the image/cursor sharing program (to be discussed next) attached to this physician is started.

Once this is done the Setup module again goes to sleep till conference time. When the time arrives it sends an image broadcast request to the Image Server. The Image Server takes the image and machine names and sends the image to all the clients.

See Appendix E for implementation details.

Image/Cursor Sharing Program

This is the collaborative imaging program which is started up by the alert daemons at every client when the associated physician 'Accepts' for the second time.

Each module allocates a customized 256 entry private colormap and initializes its local cursor database (both these topics will be addressed separately later). Once the appropriate initializations are done all the modules go into an idle state and wait for the Image Server to supply them with an image.

As soon as the image is received it is converted from a grayscale intensity map to an internal format and displayed on the screen.

An important part of the sharing module is the 'coord' data structure. This is a Programmer's Playground data type called a PGtuple. Every module possesses two variables of this type, an 'in' variable and an 'out' variable. The 'out' variable of every module connects to the 'in' variable of all the other modules. This tuple consists of the following fields:

Id number:

A unique number allotted to the module by the Image Server. Every module sets this field in its 'out' variable so that when it is received at the 'in' variable of some other module, the receiving module is able to uniquely identify the sender.

name:

User name of client. Serves almost the same function as above. The difference is that the receiving module can display the name to its associated physician.

x1, y1:

first set of position data

x2, y2:

second set of position data

Action:

Instruction code. This code can have the following values:

0 - Draw Cursor number *Id* at position *(x1, y1)*

1 - Draw Line from *(x1, y1)* to *(x2, y2)*

2 - Move Image Up

3 - Move Image Down

4 - Refresh Portion of Image *(x1, y1)* to *(x2, y2)*

5 - Marker Captured

6 - Marker Released

When a physician takes an action at his end, this action and related fields are set in the 'out' variable and reported to all other modules via the 'in' variable. The other modules then take the appropriate action.

See fig. 6 for the relationship between these modules. See Appendix F for implementation details.

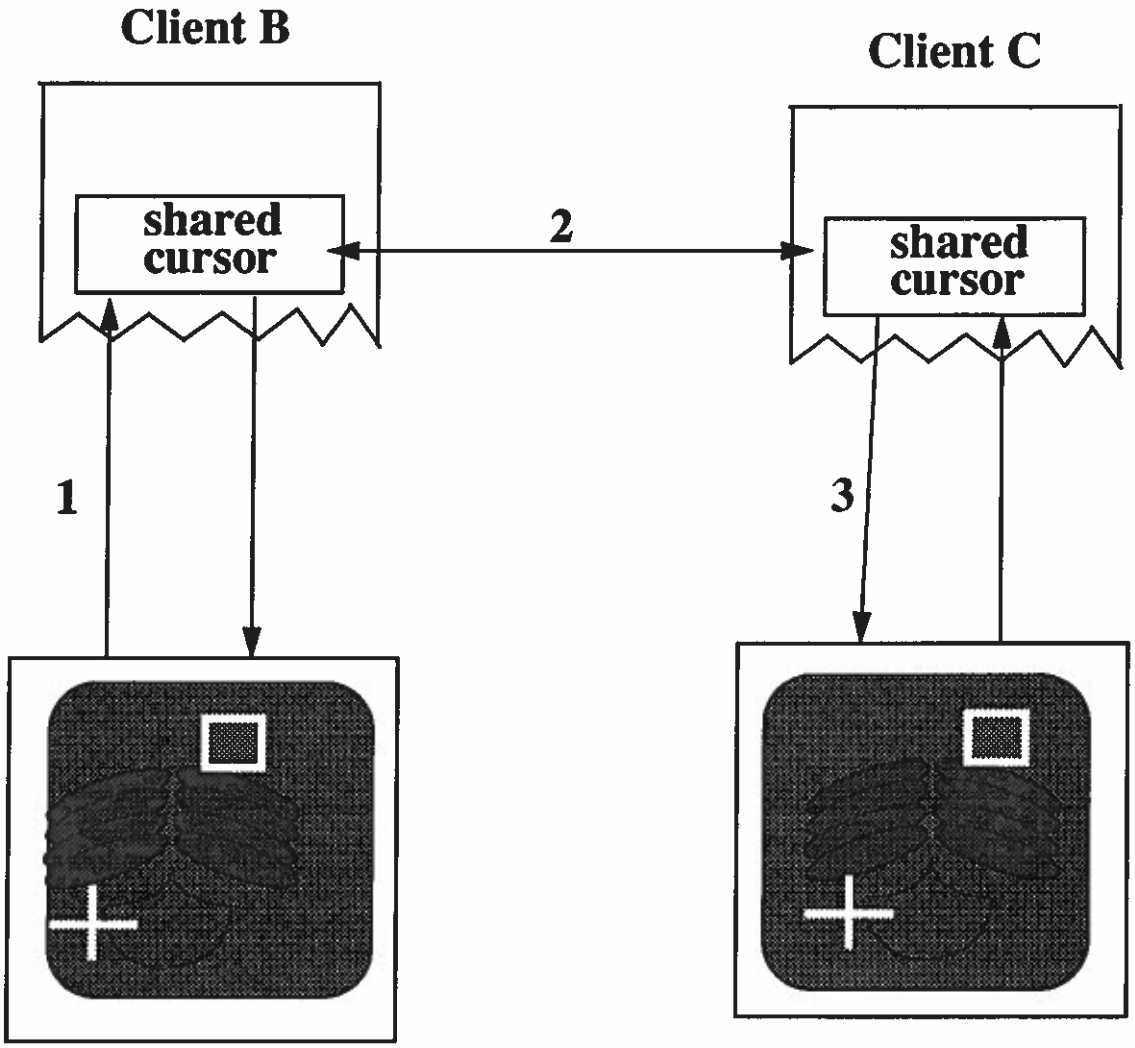
Virtual Switch

This module is responsible for automatically making connections between all the Image/Cursor Sharing modules. It contains two Playground variables of the type PGhandle. These variables are called 'In' and 'Out' and are 'short circuited', that is, the 'Out' variable is internally connected to the 'In' Variable. Every Sharing module establishes a connection between its 'Out' variable and the Switch's 'Out' variable and between its 'In' variable and the Switch's 'In' variable. The new Virtual Switch mechanism in Playground automatically makes the appropriate connections. This process ensures that all the Sharing modules have to make only two explicit connections for total connectivity.

Image Previewing/Registration Program

This program is available at every client. It is used by a physician to preview locally stored radiographs. The image can be registered at the central database as well. A single button on the program's user interface can simultaneously register the image and also invoke the conference setup program so that a conference may be convened by the physician.

Interaction between Image sharing modules of Clients



1: x,y coordinates of B's cursor reported

2: coordinates sent to other conferees (A and C) using playground

3: C receives coordinates and places B's cursor in proper position

Fig. 6

Vaudeville

This is the videoconferencing application which is launched by the Alert daemons when a conference starts. It consists of a single executable called 'Part' which is run in every Client. The Part program utilizes a MultiMedia eXplorer (MMX) connected to an ATM network to transmit and receive JPEG compressed video and CD quality audio to all other conferees. This module was written by Jyoti Parwatikar and was assimilated into the DRMC application. It currently uses Permanent Virtual Circuits but this will be changed to utilize Switched Virtual Circuits for multicast purposes when it is available.

Solutions to Select Problems

Some amount of research was also conducted during the course of this project. The topics which came under scrutiny were:

- An acceptable conference scheduling algorithm
- An acceptable floor control scheme
- Efficient and visible overlay of global cursors over the shared image

Conference Scheduling Mechanism

A successful scheduling mechanism has to be easy to use and require a minimum of intervention from the user. After studying several options the following mechanism was designed and implemented in this project:

Every physician has a Conference Setup module through which he/she can specify the desired time for a conference. When the start button is pressed the Setup module forms a message 'You are invited to a conference being convened by so-and-so at such-and-such time' and sends this message to the Alert daemons of all the potential conferees. The Alert servers display a simple 'Accept/Decline' window with the message to the physician. If the physician accepts, the alert daemon writes a unique conference id into a special 'confs' directory.

One minute before the conference is supposed to start the setup module formulates a message 'The meeting is about to start in one minute...' and sends it to the alert daemons. The alert server checks to see if the physician had accepted participation earlier. If so, it opens a similar window as before and waits for input. If no action occurs for a specified number of seconds a default of 'Accept' is assumed by the daemon. The response is reported to the setup module. Accepting also starts up the Image/ cursor sharing module and the video conferencing module. The sharing module goes into a listening state, waiting to be supplied with an image.

When the conference time arrives the setup module sends the image name and the physician's machine names to the Image Server, which transmits the image to the waiting sharing modules in each machine.

Floor Control Scheme

An acceptable floor control algorithm for drawing operations by multiple physicians has to allow users a fair chance of winning and make sure that every conferee gets roughly the same amount of time to draw. The following method was found to be satisfactory:

We have the concept of a 'marker'. Only the person in possession of this marker can draw. This eliminates the confusion that would result if all participants were allowed to draw at the same time.

A person gains control of the marker by clicking on a special icon. Once the marker is successfully selected the icon changes to a busy icon on all the conferees' user interfaces and no one else has access to it. The successful conferee can then proceed to draw with the marker by keeping the left mouse button pressed. As soon as the button is released the marker is lost and the busy icon reset to the usual marker icon. At this point any one can compete for it.

Efficient and visible overlay of global cursors over the shared image

Since every client displays multiple independently controlled cursors over his image in real time it is very important to do the cursor drawing as quickly as possible. This was accomplished by assigning every Client a unique Id and maintaining a cursor database at every site. This way, only the Id's have to be sent across the network and the corresponding cursor can be located in the

local database and drawn to screen.

An X-ray image is represented as a stream of intensity values which range from 0 (black) to 255 (white). To accommodate this color gamut a private colormap is allocated along the same lines. The problem with this colormap is that the conferees' cursors tend to disappear over certain areas of the image, no matter what color they are. The solution to this problem was to draw the cursors with an Exclusive OR function. This causes the pixels below the cursor to be mapped to the opposite end of the color spectrum. Thus when the cursor is moving over a dark area the area turns light and vice versa.

The problem with the above approach is that the cursors disappear whenever they move over a neutral gray area, since the inverse of the color is a similar neutral gray. To overcome this the colormap was allocated like this:

Colors 0, 2, 4 ..., 254 range from pitch black to neutral gray.
Colors 1, 3, 5 ..., 255 range from stark white to neutral gray.

This ensures that the inverse of a color always maps into the opposite array, that is, if a color index is even, its inverse will be an odd index color. Since the two arrays are 128 shades apart this creates an excellent contrast over all areas of the image.

Update and Future Work

Current state of the DRMC application:

- All the software components have been completed and are stable
- All the connections between modules are point to point
- One form of Conference Scheduling has been implemented
- One form of Floor Control has been implemented
- Pre-Clinical trials will begin shortly at the Electronic Radiology Lab

Documented problems:

- There is a small bug in the sharing module which results in 'ghost-cursors' being left behind in certain situations.
- Another bug in the sharing module results in the erasure of more area than expected when the third mouse button is pressed.
- The Video Conferencing module does not have the self-mute mode activated at start-up.

All of the above problems are non-threatening and do not interfere with the proper operation of DRMC. A more serious problem is that of Sun Microsystem's Operating System, Solaris 2.3. A bug in this OS causes any Playground based application to fault after an arbitrary number of successful runs. This problem does not exist in Solaris 2.4. Currently DRMC is being run on 2.3 based machines because the Dome Display drivers are incompatible with 2.4. As soon as the upgraded drivers are available the switch can be made to Solaris 2.4. It is important to note that Clients who do not possess Dome Displays can safely run Solaris 2.4.

Future Work:

Apart from the bug fixes a lot of improvements can be made to DRMC to make it a better application:

- Connections should be made point to multipoint. This will result in a dramatic increase in speed, especially in large conferences.
- The conference scheduling and floor control mechanisms should be modified based on feedback from pre-clinical and clinical trials. It may also be desirable to implement multiple schemes and allow the user to select an appropriate one at run-time.

Appendix A

Application setup for System Administrator:

In DRMC we have a concept of a central machine and client machines. The central machine is the one which runs the Playground Connection Manager, the Virtual Switch, the Registration and Image Servers and a session directory for Vaudeville. It is not at all difficult to launch all these modules in separate machines but our setup is much easier to maintain if we do it on one machine, preferably a fast RAID equipped Sparcstation 10 or greater.

The Clients all run just the Alert daemon. These servers start the image sharing and video conferencing programs when it is necessary to do so.

To facilitate the launch of the central modules there is a simple shell script called setupDRMC.center. It looks like this (this setup is customized for the Electronic Radiology Lab):

```
# home of the DRMC directory
setenv DRMCHOME ~naeem/DRMC/
# home of Playground executables
setenv PGPRODIR ~naeem/bin/
# home of the Vaudeville directory
setenv MHOME /pkg/megaphone/
# home of Vaudeville executables
setenv MHOMEBIN $MHOME/bin.sun/
# home of extra files necessary for Vaudeville
setenv RHOME /project/gbn_sw/switch/
# modify the PATH variable to point to all the bin directories
set path=( $DRMCHOME/bin $PGPRODIR $MHOMEBIN $path )
# Ports the Registration and Image Servers will use, respectively
setenv RSPORT 2002
setenv ISPORT 2003
```

```
# start up extra files required by Vaudeville
cd $RHOME/etc
rbatch &
# start up Registration and Image Servers
cd $DRMCHOME/RS
rs &
cd $DRMCHOME/IS
is &
# increment playground port
$PGPRODIR/incpg
# start up Playground Connection Manager and wait a little while
$PGPRODIR/PGcm &
sleep 3
# start the connections monitor. Optional.
$PGPRODIR/PGcmgui &
# start up Virtual Switch
$DRMCHOME/bin/logswitch -pgreq &
# start up a Vaudeville sessions directory
$MHOMEBIN/Dir &
# Backup old Resources file and build new one with the new variables
$DRMCHOME/bin/BuildResources
# additional actions
cp $HOME/.pginitrc $DRMCHOME
xhost +
```

Once this shell script is run successfully it build a Resources file which is required by all the possible Clients.

At the Client side there is a script called setupDRMC.client. It looks like this (this setup is customized for doc1 using blofeld.wustl.edu at the Electronic Radiology Lab):

```
setenv HOST blofeld
setenv DRMCHOME ~naeem/DRMC/
setenv MHOME /pkg/megaphone
setenv MHOMEBIN $MHOME/bin.sun
set path=( $DRMCHOME/bin $MHOMEBIN $path )
# set the display variable
setenv DISPLAY blofeld:0.0
# set the Dome display variable. If Dome is not available comment this out
setenv DOMEDISPLAY blofeld.wustl.edu:0.1
# start the alert daemon with the default timeout value in milliseconds. If none is specified
# it will assume a timeout of 1 minute (60,000 milliseconds)
$DRMCHOME/bin/StartAlert 10000
```

The following configuration files have to be maintained at all the clients:

$DRMCHOME/Resources, $HOME/.pginitrc, $DRMCHOME/.pginitrc, $DRMCHOME/switch and /pkg/megaphone/.mm_cfg

It is important to realize that if all the Clients are using the same physical DRMC and Vaudeville directories then there is no need for copying these files anywhere. They will automatically be available to everyone.

It is up to the site administrator to customize the shell scripts to suit the directory structure of the environment.

Important Points and Warnings:

The LD_LIBRARY_PATH variable should be set to point to where Motif, XToolkit Intrinsics, Xlib and special Vaudeville libraries are kept. Currently this can be done by sourcing the ~naeem/.cshrc file. The administrator can add the modifications to a Client's .cshrc files as well.

The Registration and Image Servers should be started in their own directories, since they make references to their local directories.

Multiple copies of the Registration, Image and Alert daemons should IN NO CIRCUMSTANCES be started by the same user in the same machine. The Alert daemon is particularly dangerous as it can potentially crash the machine. And the rule of thumb is: the faster the machine, the faster it will succumb to the ravages of the Alert daemon. This is the reason the Alert Server is not started on its own, but by a separate program 'StartAlert'. This program first checks if a prior copy of Alert is running. If that is the case, it kills it and starts a fresh Alert process.

Application Setup for physician:

No setup is required. A conference may be started by launching the Conference Setup module whenever so desired. The operation of the 'ConfSetup' program can be found in Appendix E.

Appendix B

Registration Server:

The registration server keeps a list of records subgrouped by the first letter of the record in a file called 'registered people'.
The records have the following structure:

LastName#FirstName#Occupation#Employer#MachineName#Email#Status

Below is a sample from this Database file:

b:
bari#Naeem#Student#N/A#wedge.arl.wustl.edu#naeem@arl#A
c:
charles#Rob#Radiologist#Barnes Hospital#blofeld.wustl.edu#rob@blofeld#A
cox#Jerome#Professor#Washington U. In St. Louis#hobbs.arl.wustl.edu#jrc@hobbs#A

It is important to note that the first letter is always lower case. The Status field can have the following values:

A - Client has ATM and MMX.
B - Client has ATM only
C - Client has ATM with LAN emulation and MMX
D - Client has ATM with LAN emulation only
E - Client has LAN and MMX
F - Client has LAN only
G - Client has N-ISDN and MMX
H - Client has N-ISDN only

The Registration Server handles three kinds of requests:

Registration:	Client supplies a string which contains all his/her information
Query:	Client supplies a string like 'a#=#=#=#=#=#='. The server returns all records which begin with 'a'.
Replace:	Client supplies two strings. Server replaces first string with second one.

Pseudocode for the Registration Server:

```
Main()
        Open socket and listen.
        Do Forever:
          When contacted:
              Fork program.
              Read Request_Type and Request_Body_1 from socket
              If Request_Type is 2 then Read Request_Body_2 from socket
              If Request_Type is 0 then InsertInFile (Request_Body_1)
              If Request_Type is 1 then GetFromFile (Request_Body_1)
              If Request_Type is 2 then ReplaceInFile (Request_Body_1, Request_Body_2)
              If Request_Type none of the above then report error
              Close Socket
End.


InsertInFile (String)
        LowerCase (String)
        temp <= First_Character (String) + ':'
        Open Database file
        Read till temp located in file
        Insert String at this point
End.


GetFromFile (String)
        temp <= First_Character (String) + ':'
        Open Database File
        Read till temp located in file
```

```
        Read till EOL into Request_Reply
        Write Request_Reply to socket
End.


ReplaceInFile (String_1, String_2)
        LowerCase (String_1)
        LowerCase (String_2)
        temp <= First_Character (String_1) + ':'
        Open Database File
        Read till temp located in file
                Locate String_1
                Overwrite String_1 with String_2
End.
```

Appendix C

Image Server:

The Image Server keeps a list of records in a file called 'ImageDirectory'.
These records have the following structure:

ImageName bits/pixel width height

Below is a sample from this file:

chest 10 1024 1024
hand 8 512 512

Apart from this name file the Image Server also keeps the images which correspond to these names in its local directory.
These radiographs can be either 8 or 10 bits 'deep', that is, they can contain information for either 256 or 1024 shades of gray respectively.
The Image Server should run on a very fast machine, preferably equipped with RAID technology as well.

This server handles three kinds of requests:

Registration: Client sends Image. The server adds it to its local database and updates the 'ImageDirectory' file.

Directory: Client requests names of available images. The server responds by sending the contents of the 'ImageDirectory' file.

Broadcast: Client sends an image name followed by one or more machine names. Server responds by reading the specified image from its local directory and sending it to every machine.

Pseudocode for the Image Server:

```
Main()
        Open Socket and listen.
        Do Forever:
          When contacted:
                Fork program.
                Read Request_Type from Socket
                If Request_Type is 1 then SendImgNames ()
                If Request_Type is 2 then I_Serve ()
                If Request_Type is 3 then I_Reg ()
                Close Socket
End.


SendImgNames ()
        Open ImageDirectory file
        Till EOF
                Read Record from file
                Write Record to Socket
End


I_Serve ()
        Read Image Name from Socket
        Open ImageDirectory file
        Locate record corresponding to Image Name
        Read Depth, Width, Height of image from file
        Determine size of image file (Width x Height)
        Read Image from local directory
        Read Machine Names from Socket
```

For all Machine Names

Open Socket

Write Image and Image Info (Depth, Width, Height) to Socket

End.

I_Reg ()

Read Image and Image Info (Name, Depth, Width, Height) from Socket

Open ImageDirectory file

If Image Name is already present then report error and End

Write Image Info to file

Write Image to local directory

End.

Appendix D

Alert Server:

This simple daemon is responsible for:

- Receiving an invite message from some Conference Setup module
- Asking the physician for the reply
- Conveying the reply to the setup module
- Remembering locally that the physician has accepted participation. This is done by creating a file in the DRMC/confs directory. This file is named after the machine the daemon is running on.
- Receiving early startup message from conference setup module
- Reminding physician of impending conference (if evidence was found of the earlier acceptance)
- Launching image/cursor sharing module if Physician accepts.

The alert daemon can take two command line parameters - a timer value and a default reply. When the daemon pops up a message window and asks the physician for an 'accept' or 'decline' reply the timer is started. If the timer expires without the physician having answered the program assumes the default reply and proceeds as such. The timer value is entered in milliseconds. If no value is given for timeout and default reply, a timeout of 1 minute and a reply of 'accept' is assumed.

Also to be noted is that this program is never started on its own. A small program called 'StartAlert' is entered at the command line. StartAlert makes sure that there are no orphan alert processes running in the machine. If it does find one or more orphan processes it kills them and starts a fresh one. It also makes sure that the required environment variables HOME and DRMCHOME are set. The timer value is given as a command line parameter to StartAlert, which in turn starts up the alert process with the timer parameter.

Pseudocode for the Alert Daemon:

Main()

Open Socket and listen.

Do Forever:

When contacted:

Fork program.

Read Request_Type from Socket

If Request_Type is 1 then Get_Reply ()

If Request_Type is 2 then Warn_And_Start ()

End.

Get_Reply ()

Read Invite_Message from Socket

Open Window with Invite_Message and Accept and Decline buttons

If 'Accept' then write file dummy in DRMCHOME/confs directory and Accept to Socket

If 'Decline' then write Decline to Socket

End

Warn_And_Start ()

Read Warn_Message from Socket

Read Connection Manager Info from Socket

Open Window with Warn_Message and Accept, Decline buttons

If 'Accept'

build HOME/.pginitrc file with Connection Manager and ATM Info

Start Image/Cursor sharing module

End.

Appendix E

Conference Setup Module:

The Conference setup module is used to set up and start a conference. On being launched it presents a Graphical User Interface to the user. In the 'Name' field a partial name can be entered by the doctor. As soon as the field loses focus (return is pressed or field is deselected) the setup module formulates a query string and sends it to the Registration Server. The reply is parsed and the Last and First names in all the records are displayed to the user. Similar steps are taken with the 'Image' field, which displays the images available at the Image Server.

From the conferee name and image name lists one or more conferees and one image is selected by double clicking. The results are shown in separate lists.

There is one more field which need to be filled. The time field takes an hour and a minutes value. This is the time that the conference is desired.

Upon filling in all the fields, the Start button is pressed. This initiates an Invite loop in which the setup module sends Invite messages to all the machines whose conferees were selected. The alert daemons at the remote machines respond with accepts and declines. Based on the replies the setup module compiles a final version of the conferees list and goes to sleep.

A few minutes before conference time the module wakes up and sends reminder messages to all the conferees, where the local alert daemons start up the image/cursor sharing modules. The setup module then goes to sleep till a few seconds before conference time. At this point it awakens and sends the image name and the conferee machine names to the Image Server, which goes ahead and broadcasts the image to the waiting image/cursor sharing modules.

Pseudocode for the Conference Setup module:

```
Main ()
        Create_GUI ()
        Whenever Name_Field activated get_details_from_RS ()
        Whenever Image_Field activated get_details_from_IS ()
        Whenever Name_List activated Select_Conferee ()
        Whenever Image_List activated Select_Image ()
        Whenever Hours_Field activated add_hours ()
        Whenever Minutes_Field activated add_minutes ()
        If Start_Button activated Start_Conference ()
End.


Create_GUI ()
        Create Text Entry element 'Name_Field'
        Create Text Entry element 'Image_Field'
        Create List element 'Name_List'
        Create List element 'Selected_Conferees_List'
        Create List element 'Image_List'
        Create Text Entry field 'Selected_Image_Field'
        Create Text Entry element 'Hours_Field'
        Create Text Entry element 'Minutes_Field'
        Create Button element 'Start_Button'
End.


get_details_from_RS ()
        Read entry in Name_Field
        Formulate Query
        Send query to Registration Server
```

```
        Read Reply
        Parse Reply into set of records
        Display records in Name_List
End.


get_details_from_IS ()
        Read entry in Image_Field
        Formulate Query
        Send Query to Image Server
        Read Reply
        Parse Reply into set of records
        Display records in Image_List
End.


Select_Conferee ()
        Read selected record from Name_List
        Display selected record in Selected_Conferees_List
End.


Select_Image ()
        Read selected record from Image_List
        Display selected record in Selected_Image_Field
End.


add_hours ()
        Read entry in Hours_Field
End.
```

```
add_minutes ()
        Read entry in Minutes_Field
End.


Start_Conference
        Read entry in Selected_Image_Field
        Read entries in Selected_Conferees_List
        Get Connection Manager and Virtual Switch Info from local filespace
        Conference_Time <= hours x 3600 + minutes x 60
        Send Invite messages to Alert daemons
        Sleep for (Conference_Time - 60) seconds
        Send Reminder messages to Alert daemons
        Sleep for 60 seconds
        Send Image name and Machine names to Image Server
End.
```

Appendix F

Image/Cursor Sharing Module:

This module is started up in every Client by the local Alert daemons when the associated physician's participation in a conference is confirmed. This module is the heart of the DRMC application. It is responsible for receiving an image file from the Image Server and conducting a collaborative imaging conference by allowing the physicians to view the same radiograph and giving them globally visible cursors and drawing abilities.

This module needs to know where the Playground Connection Manager and the Virtual Switch reside. This information is provided by the Conference setup module to the Alert servers, which in turn pass it to the sharing modules when they bring them up. It also needs to know where the Image Server is located. This information is provided by the Resources file which was constructed by the central setup script 'setupDRMC.center' discussed earlier. The sharing module makes use of the Dome display if it available, otherwise it uses the standard monitor to display the radiograph. The Dome display is recommended since it has a separate set of colors which the sharing module can use, instead of interfering with the standard colormap.

This module has the capability for displaying two kinds of images, 512 x 512 x 8 bpp and 1024 x 1024 x 10 bpp. It can scroll an image up and down if it too large to fit on the monitor lengthwise (the width always fits, since a standard Sun monitor is 1152 pixels wide).

Currently the sharing module works with an 8 bits per pixel grayscale colormap. If the image has a greater color gamut than that (10 bits per pixel), the intensity is mapped down to fit in 8 bpp (by dividing by 4, or rather simply right shifting each pixel value by 2 bits). This restriction can be easily fixed if the underlying hardware has a color gamut capable of supporting 10 or more bits

per color component (e.g, SGI Extreme and RealityEngine graphics support 12 bits per color component with a 12 bit overlay plane, for a total of 48 bit graphics).

Since there are multiple independently controlled cursors which have to be drawn by every sharing module, efficiency is of prime importance. Just is important is the visibility of the cursors - since we work with a hardware system capable of supplying only 256 colors (all of which we use to represent our 8 bpp grayscale colormap).

Some innovative techniques were devised to accomplish the above objectives as discussed in the last section of the report.

Pseudocode for Image/Cursor Sharing Module:

```
Main ()
    Check if Dome_Display available
    Open Display
    Connect In and Out variables to Virtual_Switch
    Allocate_Private_Grayscale_Colormap ()
    Read Image from socket
    Open Parent and Image windows
    Create Cursor Database
    Create Marker cursor and icons
    Initialize_Image_Display ()
    Put Image in Image Window
    Draw control buttons
    Do Forever
        If any activity on In variable Update ()
        If Mouse_Moved
            set Out to 0 and broadcast position
        End
        If Mouse_Button_3_Pressed
            Clear own outlines
            Redraw erased cursors
            Set Out to 4 and broadcast
        End
        If Up_Button_Pressed
            Move Image up
            Redraw erased cursors
            Set Out to 2 and broadcast
        End
        If Down_Button_Pressed
```

```
                    Move Image down
                    Redraw erased cursors
                    Set Out to 3 and broadcast
                End
                If Quit_Button_Pressed
                    Exit
                End
                If Marker_Icon_Clicked
                    If Marker_Not_Selected
                        Select marker cursor
                        Draw Marker_Selected icon
                        Set Out to 5 and broadcast
                    End
                End
                If Mouse_Button_1_Clicked_In_Image_Window
                    Do While Mouse_Button_Pressed
                        Draw Line
                        Set Out to 1 and broadcast
                    End
                End
                If Mouse_Button_1_Released
                    Set Marker_Not_Selected
                    Draw Marker_Not_Selected icon
                    Set Out to 6 and broadcast
                End
        End
End


Allocate_Private_Grayscale_Colormap ()
        For i = 0 to 254, increment = 2
```

```
                Color i = i/2
        End
        For i = 1 to 255, increment = 2
                Color i = 255 - i/2
        End
End


Initialize_Image_Display ()
        If Image_Depth = 10
                Shift each pixel right by 2 bits
        End
        Initialize_Display_Structure
End


Update ()
        Read In variable
        Case 0:
                Draw appropriate cursor
                End
        Case 1:
                Draw line
                End
        Case 2:
                Move Image Up
                Redraw erased cursors
                End
        Case 3:
                Move Image Down
                Redraw erased cursors
```

```
            End
        Case 4:
            Erase lines
            End
        Case 5:
            Draw Marker_Selected icon
            End
        Case 6:
            Draw Marker_Not_Selected icon
            End
End
```

Bibliography

1. Adapted from "Project Zeus", IEEE Network, March 1993, pp. 21-22 by Jerome R. Cox, Jr., Michael E. Gaddis, and Jonathan S. Turner.

2. Adapted from "Production Quality Video Over Broadband Networks: A System Description and Two Interactive Applications", pp. 1 by William D. Richard, Jerome R. Cox, Jr., A. Maynard Engebretson, Jason Fritts, Brian L. Gottlieb and Craig Horn.

3. Adapted from "The Programmer's Playground: I/O Abstraction for Heterogenous Distributed Systems", WUCS-93-29, June 1993, pp. 1 by Kenneth J. Goldman, Michael D. Anderson and Bala Swaminathan.