NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-56319-2

A Document Architecture and Conferencing System for a Network of Multimedia Medical Workstations

by
Pierre Henri Gross

A THESIS
submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
M. A. Sc.
in
Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
OTTAWA, ONTARIO

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

à Sylvie.

Abstract

The information handled by radiologists and attending physicians in a radiology department is by nature composed of different media types: images (eg. x-rays), voice (eg. dictated reports) and text (eg. typed reports and clinical patient data). Large benefits are expected from the integration of all these information elements on computer-based multimedia information systems. Such systems provide services for the creation, archiving and communication of the multimedia information handled in a radiology department.

In this thesis, we investigate the data management aspects of such a system. We propose a hierarchical organization of the different documents related to a radiological examination in patient folders. Furthermore we propose to use the current international standard for Office Document Architecture from ISO as a base for the definition of multimedia radiological reports combining x-ray images, text and voice sections.

We have also investigated the realization of a real-time computer-based conferencing facility which allows the discussion of cases between a radiologist and an attending physician located at distant sites. We used the general model proposed in the X-400 standard for message handling systems from CCITT as a base for our protocol architecture. We have implemented a conferencing prototype integrated with the user interface software on the radiological workstations which compose the tesbed facility.

Acknowledgements

First of all I want to thank my thesis supervisor Dr. N. Georganas for his continuous help and the confidence he showed in my work. I thank him also for having made my stay in Canada possible by his financial assistance.

I wish to express my gratitude to Dr. Luis Orozco Barbosa for his technical help and moral support throughout this thesis work and to Dr. Ahmed Karmouch for his valuable help. I want to thank also the students and staff members of the MUSIC project team for having made this Master's thesis an enlightening social and multi-cultural experience.

Finally I wish to acknowledge the financial support from the School of Graduate Studies and Research of the University of Ottawa.

Table of Contents

List of Figures

List of Abbreviations

ACR	AMERICAN COLLEGE OF RADIOLOGY
CCITT	INTERNATIONAL TELEGRAPH AND TELEPHONE CONSULTATIVE COMMITTEE
CMA	CONFERENCE MANAGEMENT AGENT
CSMA/CD	CARRIER SENSE MULTIPLE ACCESS WITH COLLISION DETECTION
DBMS	DATABASE MANAGEMENT SYSTEM
DMA	DATA MANAGEMENT AGENT
ECMA	EUROPEAN COMPUTER MANUFACTURER ASSOCIATION
HIS	HOSPITAL INFORMATION SYSTEM
I/O	INPUT/OUTPUT
ISO	INTERNATIONAL STANDARDIZATION ORGANIZATION
MCCP	MULTIMEDIA CONFERENCE CONTENT PROTOCOL
MCMP	MULTIMEDIA CONFERENCE MANAGEMENT PROTOCOL
MIT	MASSACHUSSET INSTITUTE OF TECHNOLOGY
MMTP	MULTIMEDIA MAIL TRANSFER PROTOCOL
MTA	MAIL TRANSFER AGENT
MUSIC	MULTIMEDIA SYSTEM FOR INTEGRATED COMMUNICATIONS
NEMA	NATIONAL ELECTRONIC MANUFACTURER ASSOCIATION
ODA	OFFICE DOCUMENT ARCHITECTURE
PACS	PICTURE ARCHIVING AND COMMUNICATION SYSTEM
PBX	PRIVATE BRANCH EXCHANGE
RTCAL	REAL TIME CALENDAR
UA	USER AGENT

Chapter 1
Introduction

In the past years computerized information systems have been developed in order to solve the different document handling problems in the office. The trend is toward the integration of services in an office automation system [COE 87]. An integrated system gives the users access to the different services for the processing of office documents from one workstation. Several types of services can be distinguished: production services (eg. document editing, formatting, printing), archiving services (eg. document filing and retrieving) and communication services (eg. electronic message handling and conferencing).

The word *integration* covers another important aspect, namely, the integration of different data types inside the basic object of an office environment: the multimedia document. Whereas the early office information systems could only handle textual documents, new systems have appeared [UED 87], [FOR 86] which can manage documents composed of text, voice, graphics and images.

There is a strong need for automation in the hospital environment as well. Already, so called Hospital Information systems are used to automate the patient registration process and other administrative tasks, such as billing operations. But the needs of modern radiology with its huge generation of x-ray films and digitally acquired images through new computerized

modalities (eg. imaging equipment like nuclear medicine) are more demanding. New Picture Archiving and Communication Systems (PACSs) have been introduce to solve the communication problems of modern radiology departments. The requirements of such systems are demanding: huge data storage needs, fast reliable communication of images in real-time.

In this thesis, we propose to see how this multimedia technology, when allied with the open system architecture which has been successfully introduced in the field of office automation, can be applied to the medical environment and specifically to radiology. This thesis work has been pursued in the framework of the MUSIC project. This project originated in the department of electrical engineering of the University of Ottawa.

In this work we have been studying two aspects of the problem. One was the development of a data architecture for the management of the multimedia information produced inside the radiology department, the other was the research of a solution for the realization of a shared consultation service between two workstations. The presentation of this research is organized in 7 chapters.

In chapter 2 we give a general introduction of the MUSIC project. After a description of the current manual operations in the radiology department, we explain the benefits which are expected from the use of a multimedia communication system and the services it should provide. Then we present the system architecture and user interface of the experimental setup for the project.

In chapter 3 we present the basic concepts used in multimedia systems for multimedia data management and communication and for the conferencing services. We present some representative systems which provide multimedia document management services or conferencing services.

Chapter 4 describes the design of the multimedia filing system used to organize the different data objects (ie. x-ray images, voice reports and text information) in a hierarchical fashion. We then describe the data model for the multimedia document structure which can be used to represent documents in which the different media elements are linked together.

The last chapters describe the conferencing application for our medical communication system. In chapter 5 we present a general system model and protocol layering for an integrated conferencing service based on current standardization work in the area of message handling systems. Some issues relevant to our application are discussed here.

Chapter 6 describes our design of a conferencing service implementation. We discuss the requirements, the user interface and the protocol architecture. Finally, in chapter 7, we explain the implementation aspects of the prototype that was integrated with the early experimental testbed facility system.

The conclusion emphasizes the different results that we obtained from our research and the possible future developments. The appendices contain the descriptions of the programs which were written for the prototype.

Chapter 2
Medical Application of Multimedia Systems

In this chapter we introduce the motivation for the introduction of multimedia communication system in the hospital environment. We will present an overview of a system which was developed at the University of Ottawa in the framework of the MUSIC project.

2.1 Communication Needs in the Hospital

Much as in the area of office communication, it has been noted that computer-based communication systems could represent a solution for the problems which arise in the operation of a large hospital and especially at the radiology departments. Such a system should provide services for the creation, archiving, communication and management of the huge amount of information handled in such an environment. We present here the existing operations in an hospital radiology department and show how the problems which arise can be solved by a communication system.

Hospital activities generate, for each patient, a large number of data sets of different types and formats [MUS 87]. The need exists for both individual and shared access to these data sets from different locations in the hospital and by different users. To illustrate this and to explain

the different operations and data sets involved, we show the key steps involved in a typical examination process in a radiology department [ROG 86], [OOS 87].

•**Scheduling:** At the request of the referring physician, personnel work through the patient database on the existing Hospital Information System (HIS) to enter or retrieve patient demographic information, identify the patient with a unique identification number, assign a procedure or examination number and schedule the examination.

•**Examination preparation:** Before an examination, the patient examination folder which contains all images and diagnosis related to one specific examination is completed with information from previous examinations folders retrieved from the main library.

•**Examination:** A technician generates new images and includes them in the patient folder. The patient folder is sent to the radiologist.

•**Interpretation:** The radiologist reviews the images and dictates a report on a tape recorder. This report must then be typed by a secretary and approved by the radiologist.

•**Archiving and reporting:** The final patient folder must then be sent to the requesting referring physician and eventually stored in the main library.

In addition to this main information flow, other collateral activities take place: conference/teaching activities during which a group of physicians discuss a set of cases, consultation between a referring physician and a radiologist concerning a specific patient examination, and emergency case handling.

Some features in the existing manual system of handling radiographic information are inefficient and could benefit from a computer-based communication solution. Since there is only one copy of the patient folder, it is impossible to have two physicians consult it at the same time in different locations. Furthermore, it is not uncommon that a patient examination

folder is not in the main archive when needed. Time is another important factor, especially in emergency cases. In the study of the operation in the radiology department associated with the research we have realized that a lot of time is lost by a physician searching for information, finding radiologists to get specific diagnosis information or waiting for the results of a specific examination. Even moving the patient files around the hospital takes time.

Two answers have been brought to these problems. Hospital Information Systems (HIS) have been introduced to keep track of patients, examinations, billing and various type of management data. We showed earlier how those systems are used to retrieve patient information. However they can only handle textual information. To answer the specific needs of radiology departments, in the beginning of the eighties, so-called Picture Archiving and Communication Systems (PACS) have been studied.

As stated before, PACS offer a solution to the communication and archiving problems in a radiology department. The purpose of these systems is to store images from different types of diagnosis equipment in an image database, which is accessible from different workstations. They become even more justified by the fact that recently developed digital techniques, such as Computer Tomography, Digital Subtraction Angiography and Magnetic Resonance Imaging already generate large quantities of digital image data. Nowadays these digital images are temporarily stored on magnetic tapes and printed on film when a radiologist wants to make a diagnosis [BIJ 87].

2.2 Multimedia Communications in Medicine

The MUltimedia System for Integrated Communications (MUSIC) project has been initiated at the Department of Electrical Engineering of the University of Ottawa to investigate the application of multimedia technology in radiology. Such a multimedia communication system

would provide the services of a PACS, not only for image information but for various types of information such as voice, computer data, image and commands. This would allow all the information produced in the examination processes described in the previous section to be fully integrated on a communication system. Such a system can help radiologists provide better services in the following ways:

- **The remote access** of patient information by a physician could reduce the waiting time and the search for documents from days to minutes.
- The generation of **multimedia reports** in which radiographs and radiologist voice reports are tied together to provide automatic synchronized play-back could enhance the communication of information to the attending physician.
- **A computer based conferencing** facility could allow discussion of cases between radiologist and physician without the need of them to have a face to face meeting.

From the presentation of the current operations in a radiology department that we have done in the previous section, we see that the patient examination folder is the basic data set that a multimedia communication system should manipulate. Such a patient folder contains the different x-ray images related to a specific examination as well as the request forms, patient data sheet and diagnosis report. A multimedia system would handle this information in electronic form, providing services for communication, archiving and production of this medical data.

To show some of the capabilities of such a system, let us go through a simple sequence of operations. When a patient arrives at the hospital, all previous medical data related to him is retrieved from the permanent archiving node and a new examination is prepared. Once x-ray images have been taken from a patient at an acquisition device, they are stored in the patient folder with the clinical data relevant to his case. The radiologist can then retrieve the patient

folder and produce a voice report which will also be stored in the folder. The attending physician, in his turn, can access the folder and get the report made by the radiologist. Furthermore, if the patient comes from another department, the folder can be sent to a physician on a remote cluster. Also, for difficult or urgent cases, a computer-based conference may be established between two physicians. They can have simultaneous access to the patient folder on their workstations without having to meet in person.

The requirements for such systems regarding storage capacity, display quality, communication capacity, database management and user interface are very high. In this thesis we examine some aspects of data management and conferencing. In the next section we present the architecture of a hospital-wide communication system and that of a testbed facility used in the MUSIC project.

2.3 Hospital Communication System Architecture

Figure 2.1 shows a global view of a hospital-wide communication system. Display workstations are connected to a multimedia file server by a local area network, forming a cluster. Each of those clusters can be seen as the local network of one department in the hospital. Such an architecture reduces the global data throughput since the main information flow is local to each department.

Each cluster has its own acquisition devices for medical images. Due to the storage requirements of digital images, the local multimedia file server is only used for short term storage of the generated documents, typically during the time a patient stays in the hospital. A permanent storage system is connected to the backbone network and provides long term archiving (several years). All of these storage facilities constitute a distributed database. The

backbone network may also be connected to a wide area network through a gateway so that documents can be sent to systems or workstations in remote sites.

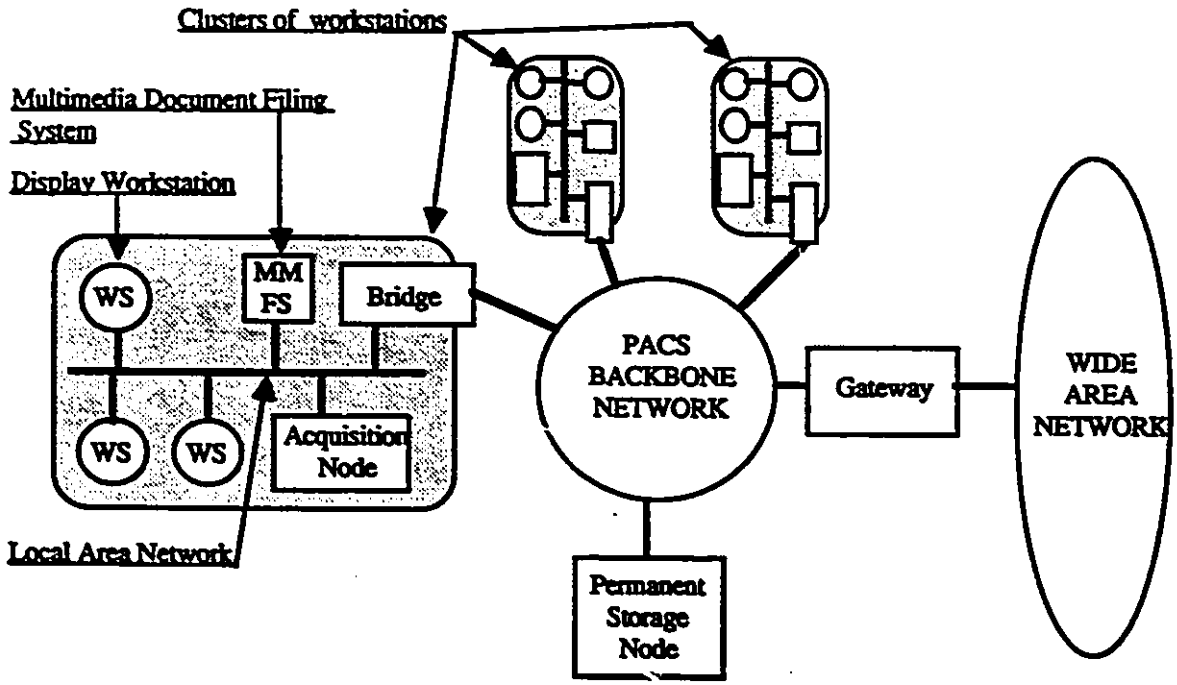


Figure 2.1. A Hospital Communication System

The system described hereafter can be seen as an example for a local cluster of workstations. Such a cluster would naturally serve one department of a hospital.

2.4 MUSIC Testbed Facility Architecture

In the framework of the MUSIC project a testbed facility has been developed. The purpose of such a system is to be installed in a radiology department. The system would convey x-ray images which are digitized by scanning existing x-ray films.

2.4.1 Hardware Architecture

As shown on figure 2.2, this facility is based on workstations interconnected by a local area network which is currently a broadband CSMA/CD type network (Sytek 6000). Different components appear on the figure:

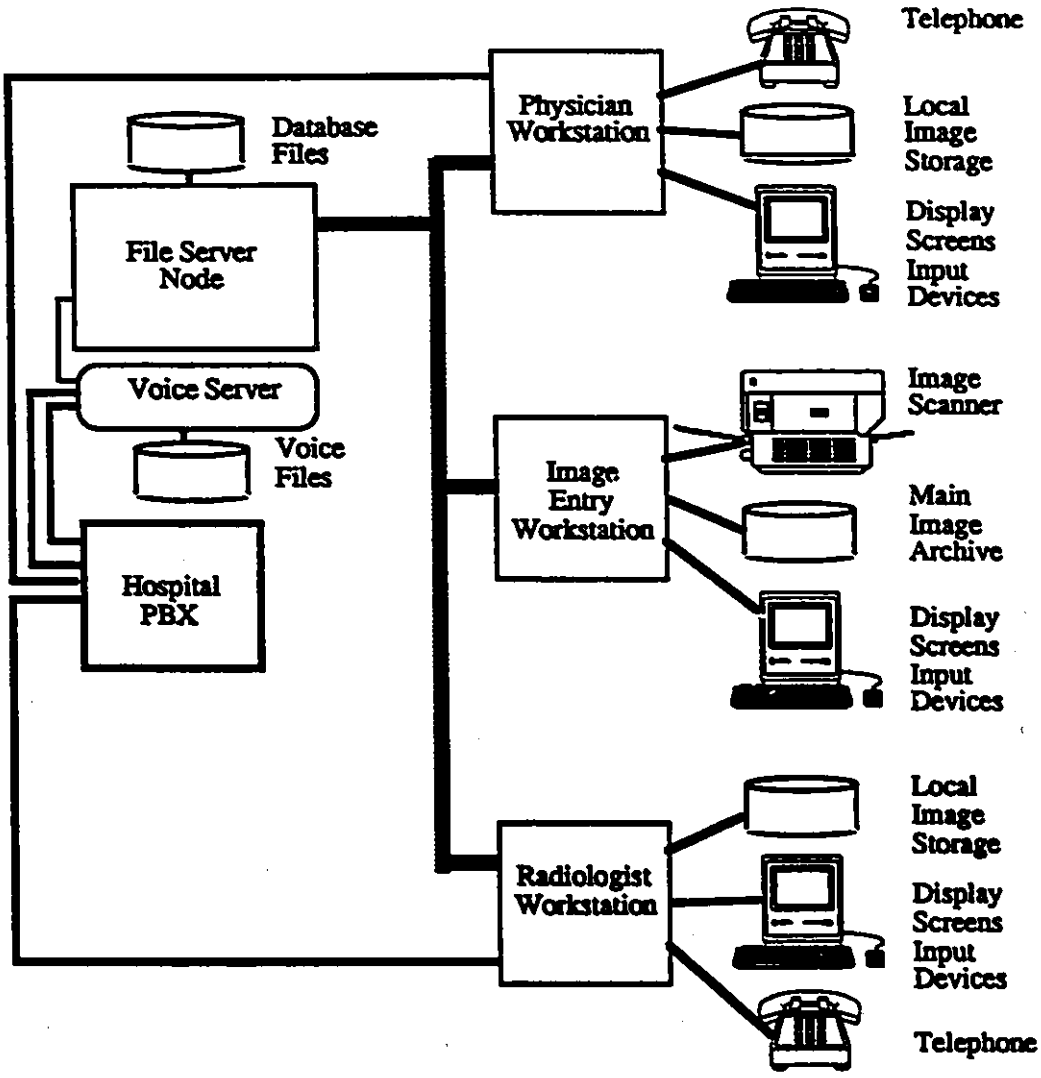


Figure 2.2. Music Testbed Facility

•The image-entry workstation is where the x-ray films are stored onto the system under electronic form. Typically, the image-entry station contains a mass storage device such as an optical disk to store the huge amount of data generated by the scanning device (1 MBytes for a resolution of 1024x1024 pixel image with 8 bit coding). The station serves the request for images of other stations. However, since the response time seen by the user of each workstation may not be acceptable if the images had to be loaded from the central file server, each display workstation has local storage where images can be downloaded in advance.

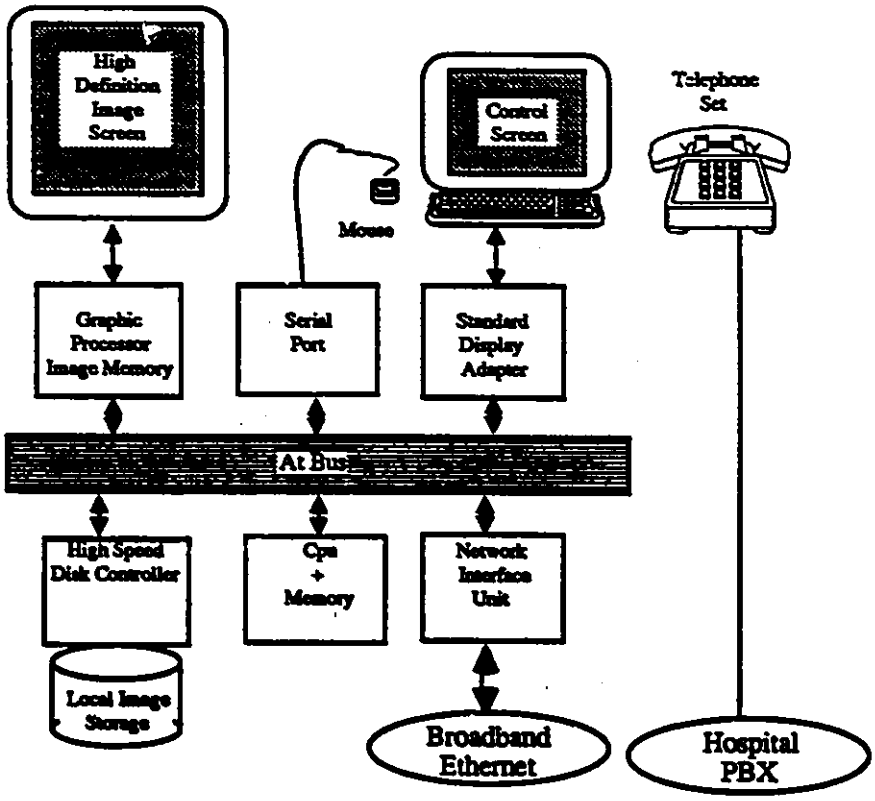


Figure 2.3. Hardware Architecture of a Workstation

•The consultation workstations are used by physicians to access all the information available for a patient. This information can be in textual form on the low-resolution monitor, a high-resolution x-ray image on the high-resolution monitor or as voice output on a speaker

connected to the telephone network. The user can issue commands by moving a mouse, typing on his keyboard or speaking in his microphone. This architecture is summarized in figure 2.3. This workstation, much alike the one of the digitizing station is based on an IBM™[1] AT compatible computer with fast processor, large memory and fast disk access.

•The voice server provides the voice capabilities in the system. As we see, the voice does not use the local area network since ISDN (Integrated Services Digital Networks) networks are not yet available, but it is provided to the user under analog form. The connection to the workstation goes through the analog phone lines but the voice information is stored in electronic form on magnetic disk. The voice server is controlled by messages from each workstation which are received by a program residing in the file server node.

•The file server stores all the information necessary to keep track of the patient folder database as well as the text documents. It is a commercially available file server. As shown, the file server controls the voice server.

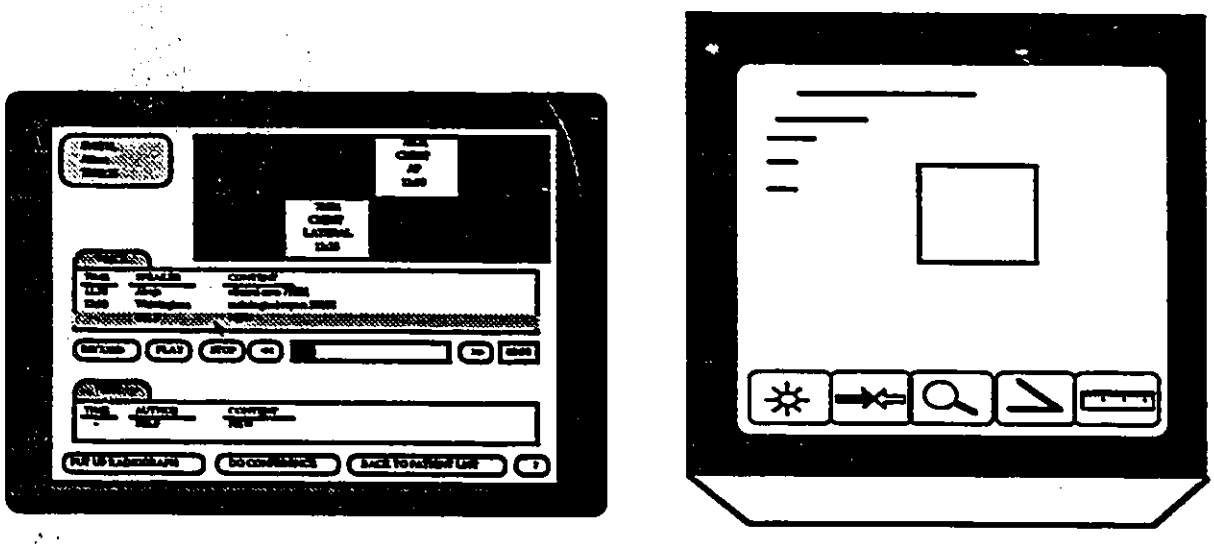
2.4.2 User Interface for a Radiological Image Workstation

The purpose of the user interface is to allow the physicians to interact with the different *objects* managed by the system. Those are mainly, the patient folder, the reports and the images. The patient folder is an electronic model of the folder used in the manual system which contains all information pertaining to a patient for a given examination. The basic information objects are the x-ray images and the voice or text reports generated by the radiologists after the analysis of the images. In this section we describe the user interface that has been built to support the various multimedia tasks required for our application [MUS 87].

---

[1] IBM is a trademark of International Business Machines Corporation.

When a radiologist comes to a workstation, he sees that some patient folders require his analysis. He then selects a patient folder, looks at the radiographic images that are of interest to him and dictates or types his comments. 'Looking at the image' actually refers to the task of performing the analysis of the image using all available enhancements and manipulation operations. On the other hand, a physician at a workstation selects the desired patient folder, listens to or reads the report, looks at radiographic images, or does both at the same time.



CONTROL SCREEN IMAGE SCREEN

Figure 2.4. User Interface

Patient folders and their contents are presented as lists from which an item can be selected; all commands available at any given moment are also presented as lists from which the user can select the appropriate command.

The visual workspace of the user is divided into two separate CRTs: a high-resolution colour monitor, the Image Screen, and a low-resolution monochrome monitor, the Control Screen (figure 2.4).

The **Control Screen** is dedicated to control operations for the selection of an image to be displayed on the Image Screen. The dialogue management between the user and the workstation is handled by menus displaying all functions available at any given moment. The mouse is used on this screen to highlight and select the items (images or text and voice reports). The dialogue is organized in a succession of screens. One of those screens is shown figure 2.4.

The Patient List Screen. As soon as a user has identified himself to the workstation the system displays the patient list. The purpose of this screen is to allow the user to select a patient folder for examination or conference. Each line of a list gives the patient name and patient unique identification number of the folder. The line of an active folder shows as well whether there is a need for his attention in a folder, even before having selected it. From this screen, the examination of the selected patient folder is initiated by selecting the box labeled 'EXAMINE PATIENT'.

The Examine Patient Folder Screen. At this point, the patient folder has been selected and it is time to examine it. In order to do so, the user must be informed of the folder's content. Two screens are used to achieve this and to allow the addition of information to the folder.

The purpose of the first screen is to present a list of all the images contained in the folder by grouping them by areas of the body (eg. CHEST) and by relating these groups to the sketch of a human body. Each of the images in a group is identified by the name of the view (eg. LATERAL), the time of day at which the radiographic film was taken, and the examination number.

At any time, clicking on the 'RETURN TO PATIENT LIST' closes the patient folder and returns to the list of patients. Clicking on 'VOICE' or 'TEXT' displays the second screen (figure 2.4).

The second screen is similar to the previous one except that the sketch of the human body has been replaced by a list of the voice and text reports contained in the folder. The top half of the area is for voice reports and the bottom half for text reports. Existing reports can be read or heard by clicking on the corresponding list item. New reports are created by clicking on the 'NEW' list item.

A user (radiologist or attending physician) may create a new voice report or listen to and modify an existing voice report. Once a voice report is open, the user will have the same facilities as if he was using a tape recorder (record, play, rewind, fast-forward) (figure 2.4). This is to replace the actual system (tape recorder) used in the Ottawa Civic Hospital.

The Image Screen is dedicated to the display of radiographic images and their manipulations. Image manipulation operations are initiated via a narrow band of icons at the bottom of the screen. The operations are, in left-right order in the bottom of image screen of figure 2.4: ***contrast*** and ***brightness control, inverse video, measure angle, zoom on window,*** and ***measure distance.*** Each of the operations above can be initiated by clicking on the appropriate icon.

2.5 Conclusion

We have presented in this chapter the motivations of the MUSIC project and an overview of the testbed facility which has been developed in the framework of this project.

A first prototype of the system will be deployed in the Ottawa Civic Hospital by beginning 1989, initially linking the Radiology and Emergency departments, and will serve as a testbed for conducting limited clinical trials. One of the primary purposes of the trials will be to assess the clinical utility and acceptance of multimedia radiology report generation and reviewing. Moreover, this trial will permit us to evaluate the performance of the system within a real-user environment.

Chapter 3
State of the Art

We propose to use some of the ideas and results obtained in the areas of multimedia data management and multimedia conferencing to the medical environment. In this chapter we present existing multimedia systems which have been developed for the office environment. We will first present examples of multimedia document handling systems and in a second part examples of multimedia conferencing systems.

3.1 Multimedia Data Management: State of the Art

3.1.1 Multimedia Filing Systems and Document Structure

There are two distinct levels in a multimedia data management system: the filing hierarchy and the multimedia document structure. We can take the example of the current operation in a radiology department. The central library holds patient folders which contain all the documents corresponding to one examination. These documents are of different types: images or text reports and patient information sheets. The folders and the library form the filing hierarchy which is used to gain access to the documents. In a computer-based multimedia information system, documents can themselves be composed of several media objects.

A filing system is used to organize the multimedia documents in order to allow their retrieval. Multimedia documents are units inside a filing system. As shown in figure 3.1, the multimedia database is actually composed of two important components: one responsible for the management and retrieval of documents and the other with the editing of the documents themselves. This includes the editing of each single media content element, the editing of the logical structure (the internal organization of the document) and the control of the physical rendition of the document (layout). A document retrieval system can be implemented easily using a standard relational database management system since the information involved at that level is only textual information.

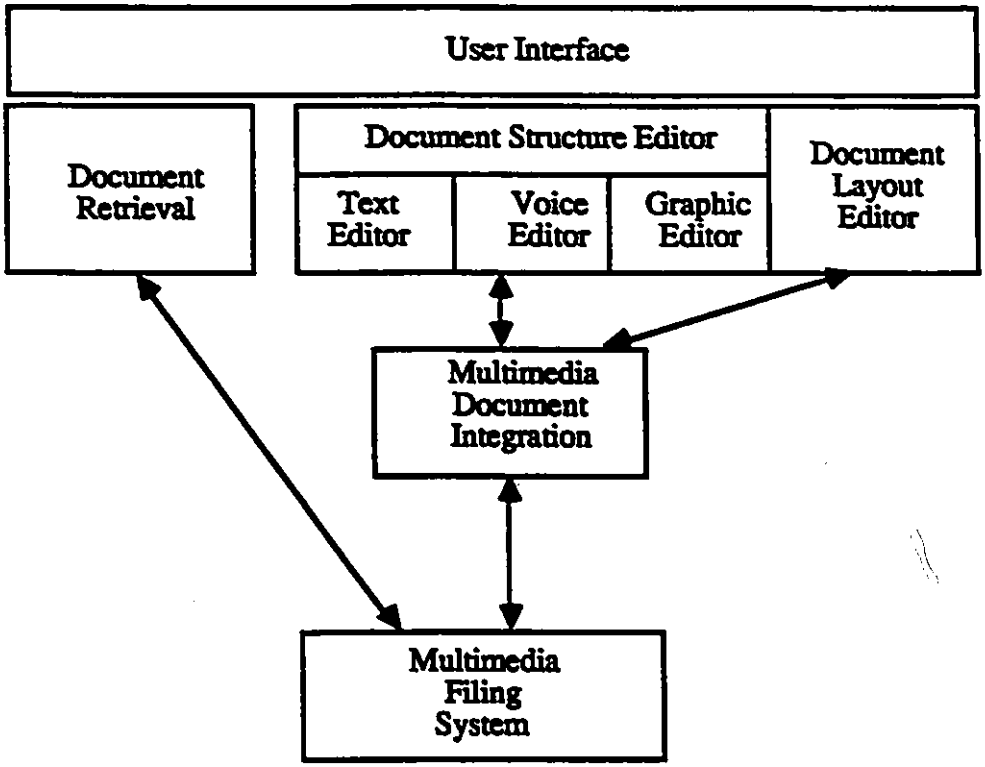


Figure 3.1. Multimedia Document Handling System Components

At the user interface level, this means that we have two kinds of modes: the user can either search for documents and browse through a certain number of documents, or, having chosen a

given document, he can browse through this document. In the Minos multimedia database developed at the university of Toronto [CHR 86], these two modes are called browse-through-object and browse-within-object. In our application for instance, physicians could browse through a list of x-ray images (each image can be seen as a document) or they could consult a multimedia report, turning pages of this report as they advance in their reading.

3.1.2 Database Services

A complete database management system (DBMS) in a professional environment must provide the following services:

- Authorization control: The access and modification of data must be in accord with a user's privileges.
- Concurrency control: The database management system (DBMS) must ensure that the concurrent access of shared data does not result in deadlocks or data incoherence.
- Crash recovery: The DBMS must provide a mechanism which guarantees the consistency of the data when user workstations or the network fails.
- Query specification: The DBMS must allow the user to specify queries to retrieve specific documents.
- Distributed architecture: On the architectural level, the DBMS must provide a service for handling distributed data. In our application, this is necessary to reduce the delay when a user requests an image.

3.1.3 Two Systems

A new generation of office automation products integrating the handling of text, graphics, images and voice is now being developed. We shall take two examples of such systems and point out some of their original features.

•The Diamond System

Diamond is a computer-based system for creating, editing, transmitting and managing multimedia documents [THO 85] and [FOR 84]. A Diamond document may contain text, graphics, images and speech as well as spreadsheets. The user interface is built using a window oriented display on a personal workstation.

The document database is implemented as a database distributed on several servers. The system offers also powerful mail handling capabilities. The architecture supporting the Diamond system is a group of clusters interconnected by the DARPA network. Each cluster includes a high speed local area network, personal workstations, shared server hosts and an internet gateway.

The main features of Diamond are the powerful document editing it offers (cutting and pasting between documents, integration of spreadsheets, mixed layout of text, voice and graphics) and its distributed architecture.

•The MINOS System

MINOS is an object oriented multimedia information system that provides integrated facilities for creating and managing multimedia complex objects [CHR 86]. The system is mainly a multimedia database without any mail handling capability. However it presents powerful features for the handling of multimedia documents:

•The user interface of this system has basically two operation modes: the browse through documents mode and the browse within documents modes. In the first mode the user can select various filters to select a group of documents and can view them in miniature.

•Browsing within a document can have several modes. The user can turn pages of the document as he would for a paper document, or he can be presented with an automatic play-back of a sequence of graphic objects. Also the data model used in this system allows small annotations to be attached to various parts of the document.

This system is a good example of a rich multimedia document handling and editing system. It shows many features which can be applied to our problem in the medical environment (for instance the play-back of a physician report synchronized with image and pointing information).

Most of the research done in the world of office automation cannot be applied directly to a hospital communication system. For instance, the document architectures used in office automation are defined to produce printed pages of text and low resolution graphics whereas in a radiology department, multimedia reports can be defined using high resolution digitized images and digitized voice segments. The output format of such documents is far from a page format.

3.2 Multimedia Computer-Based Conferencing

Studies have shown that, at the manager level in an office, people spend approximately 40 percent of their time attending conferences and meetings. Also the decision process at this level is often highly dependent on fast access to information. For this reason teleconferencing systems have begun to spread in the office since the 70's [WAT 85]. First, only voice

conferencing using the telephone network was available but now, with the development of personal computers, multimedia conferencing based on voice, text and graphic data have been proposed.

The same communication needs apply to the medical community. A multimedia medical conference system should provide real-time conferencing capabilities with high definition radiological images and voice services. Such a system would provide a fast and efficient way for physicians to arrange meetings and discuss the clinical situation of a patient, having access to all relevant x-ray images.

3.2.1 The Concept of Shared Workspace

The media supported in our system are text, graphics, voice and high-resolution x-ray images. The multimedia workstation hardware is composed of a high-resolution screen for computer output, a keyboard, pointing devices, a microphone and a speaker.

The multimedia conference is implemented by means of the shared workspace. In most systems, the concept of shared visual workspace is used to describe the part of the display screen which is replicated on each workstation. In the Diamond MMConf implementation of the conference (figure 3.3), the shared visual workspace is all the data shown in the lower part of the display. In fact, the display is divided in two functional areas, the private workspace and the shared workspace. Also, there are two different cursors: the shared cursor (large arrow) which is controlled by the user owning the floor and the private cursor which is used by each user to control its own workstation.

In the case of multimedia workstations there is a shared voice communication channel. When the multimedia documents contain voice sections, they may be part of the shared workspace. Thus, we can speak of a shared audio workspace.

Moreover, in systems which support collaborative action on shared data like in RTCAL [SAR 84] and EMCE [AGU 86], the shared workspace also contains the data. Since the users can create and modify the documents they use during the meeting, these data objects can be seen as part of the shared workspace.

3.2.2 Level of Conferencing Services

We can distinguish between three kinds of conferencing systems among those existing today:

•Asynchronous Conferencing Systems

The first computer-based collaboration between geographically dispersed users was electronic mail. With the combination of inexpensive personal computers and the widespread use of data communication technology, electronic mail has become relatively common.

Some conferencing capabilities have been build on top of electronic mail services. In these systems, users register in a conference, send messages to a central computer and have access to all the messages of the participants of the same conference.

Such capabilities are important for several reasons:

- There is a need for long term interaction between people at the same time as there is a need for meeting oriented systems.
- Electronic mail is becoming more and more standard. Standardization work such as that of CCITT in the X-400 series, will help the world-wide interconnection of systems. At the same time, this standard promotes multimedia content types, whereas most electronic mail systems today only support text-only messages.

Some authors [GAR 86] have therefore proposed architectures for real-time conferencing which are based on the Message Handling System functional architecture and protocols proposed by the X-400 standard series.

•Real-Time Conferencing Systems

In order to handle rapid group decisions, real-time conferencing systems are required. However we distinguish between those providing a simulation of face-to-face meetings, providing for instance audio discussion over a displayed document, and those which extend this possibility, supporting collaborative work during the conference.

Typical examples of the first type are the Audio Document teleconferencing systems presented earlier. Although they allow a very good interaction between the users, they do not allow any modification of the documents.

In some office automation products [SAK 85] distinction is made between final form documents and editable documents. The whole point is to support the exchange of documents in non-final form and to allow users to work together on the editing these documents. This point has also been emphasized in the development of the ISO office document architecture standard definition [HOR 84]. An integrated conferencing system should allow users to work together on documents.

•Computer-Supported Cooperative Work

Collaborative Work systems allow group interaction not only to discuss a problem but to actually create something together. Such systems are not necessarily based on real-time conferencing capabilities. For instance, at MIT a collaborative editing system has been tested [GRE 87]. It allows users to work on different parts of the same document in an asynchronous mode and does not use real-time interaction.

However the RTCAL system is an example of collaborative work in real-time [SAR 84]. Another example is the COLAB experiment [STE 87] at Xerox PARC. The COLAB is a meeting room for small working groups of two to six people using personal computers interconnected over a local area network. The users are located in the same room, but they can use all the power of the personal computer during the meeting and have access to the shared data.

Such issues as collaborative document editing and multi-user interfaces are beyond the scope of this work. However our prototype conferencing system is more a collaborative system then a document based conferencing system. Although the goal of our system was to provide a computer-based meeting between physicians, which mainly means displaying the same document on two workstations, it provides access to the same tools as when a workstation is used in autonomous mode.

A computer-based real-time conference is not intended to simulate or replace all face-to-face meetings. In such a case, video teleconferencing would be more appropriate. However it has to provide the interaction modes necessary to allow two users to discuss a specific document without leaving their desks.

3.2.3 Conferencing in "Non Integrated Systems"

The beginning of teleconferencing is very old. As early as the 1930's, a form of teleconferencing was implemented with the introduction of multiparty telephone communications [WAT 85]. Of course these early systems were voice based. Further developments have made it possible to combine voice with video or graphic information to enhance the efficiency of the conference.

•Overview

Table 3.1 shows a classification of the conferencing systems by the media types they support. Audio-conferencing systems provide a good conversational capability at a relatively low cost. A meeting is set up by having different attendees call a specific number corresponding to an audio bridge. This bridge then adds all the incoming voice signals and distributes the resulting signal to all the users. The users need only a standard telephone set. However, even if the audio medium is fundamental to a meeting, it conveys only transient information and often graphics are necessary as a base for the discussion of complex problems.

Systems		Media		Conferencing Terminal	Features
		Basic Media	Optional Media		
Audio Conferencing		Audio	Character data	• Speakerphone • Audio conferencing terminal	• Low Cost • Conversational capability
Visual Conferencing	Freeze-frame video conferencing	Still picture (Half tone picture)		• Freeze-frame conferencing terminal	• Relatively low cost • Drawbacks in conversational capability
	Audio-graphic conferencing	Telewriting	Still picture	• Telewriting tablet	• Relatively low cost • Conversational capability
	Audio-document conferencing	Document pattern	(Half-tone picture)	• Document conferencing terminal	• Relatively low cost • High resolution display
	Full-motion video conferencing	Full-motion video	Telewriting	• Video conferencing terminal	• Conversational capability • Expensive
Computer conference		Character data	Still picture	• Data terminal	• Low cost • Drawbacks in conversitional capability

Table 3.1. Classification of Teleconferencing Systems [WAT 85]

On the other extreme, full-motion video conferencing provides probably the best simulation of a face-to-face conference. The cost of the terminal equipment and of the underlying network are very high, yet the type of information transmitted is transient in the sense that documents cannot be transmitted and shared between the conference rooms. Typically such systems require that people go to specially equipped conference rooms.

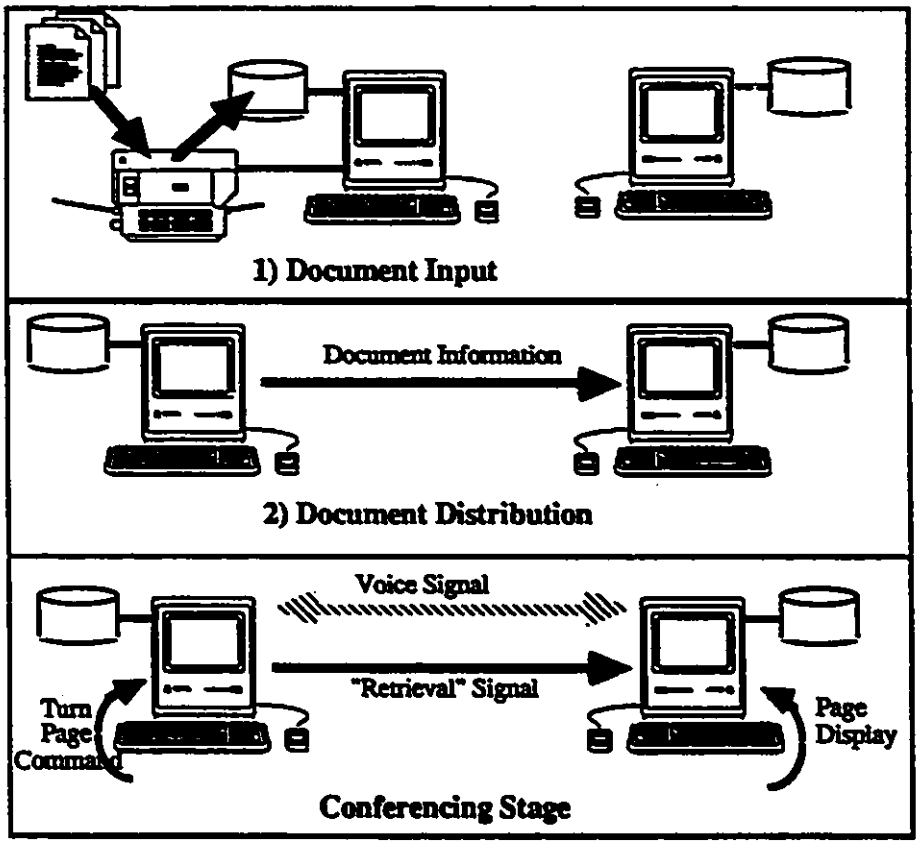


Figure 3.2. Outline of an Audio-Document Teleconferencing Service

Computer-based conferencing systems have been developed by several companies. Most of these systems are based on text-only mail services and use the most current type of terminals; they are therefore very cost effective. The discussion bandwidth is limited compared even to voice conferencing systems and the use of the keyboard makes them less

user-friendly. This type of service is provided by many electronic billboards together with mail and general information services. In this sense, the conferencing system is integrated with the mail service.

The most promising kind of services described in [WAT 85] are audio-graphic or audio-document conferencing systems based on high resolution still images. They constitute a low bandwidth alternative to video-conferencing systems still providing all the users with a common view of a document and with an audio link.

Figure 3.2 describes the features of a document oriented teleconferencing system [KOT 87]. Each node has storage facility, a scanner to enter new documents, a display workstation with audio and possibly tele-writing capabilities. Graphic documents can be scanned and stored in the workstation. They can be distributed in the preparation stage or on demand during the conference. Of course the access of preloaded documents is faster since only the name of the document has to be sent to display it. During the conference the participants interact by changing the displayed images, talking to each other using the common voice channel, by pointing to specific areas of the document and by making annotations on the screen using the writing pad.

•Observations

The systems we have described until now share the following features:

- •they are services offered by a specific manufacturer and therefore are part of a closed environment.
- •they do not provide extended manipulation of the documents being shared, either because the documents being shared consist only of digitized graphics under a static and none revisable form or because the information being exchanged consists of transient information like voice or video images.

•they are intended to be used through public networks on a nation-wide basis.

They have two drawbacks:

•None of these systems are integrated in an office automation environment. The documents which might be produced by a computer in an office must be printed on paper in order to be exchanged during the conference. For instance, they are inputted to the scanner in the case of the document oriented conferencing system.

•Even if they sometimes allow advanced interaction between participants of the conference like annotation of the image, tele-writing, pointing, the document still cannot be modified. All the work produced during the conference is lost when it ends.

In the following section we will examine systems which integrate multimedia document handling and conferencing. Some of them allow cooperative working on shared documents.

3.2.4 Conferencing in Integrated Systems

Conferencing capabilities receive their full meaning when they are integrated with all the services provided in an office automation environment. Conferencing capabilities in systems which already have the facilities for handling and editing multimedia documents enhance the services provided to the users. The users not only have the possibility to view and discuss documents, but they can modify these documents during the conference.

•Conferencing in the Diamond System

Diamond is a computer-based system for creating, transmitting, editing and managing multimedia documents developed at the BBN Laboratories [FOR 85]. A document may contain text, images, voice sections, spreadsheets and charts. Diamond is a distributed system implemented as a cluster of workstations and servers interconnected by a local area network. The network handles digitized voice as well as data.

A prototype multimedia conferencing system called MMConf has been implemented in Diamond. We will show some of its features.

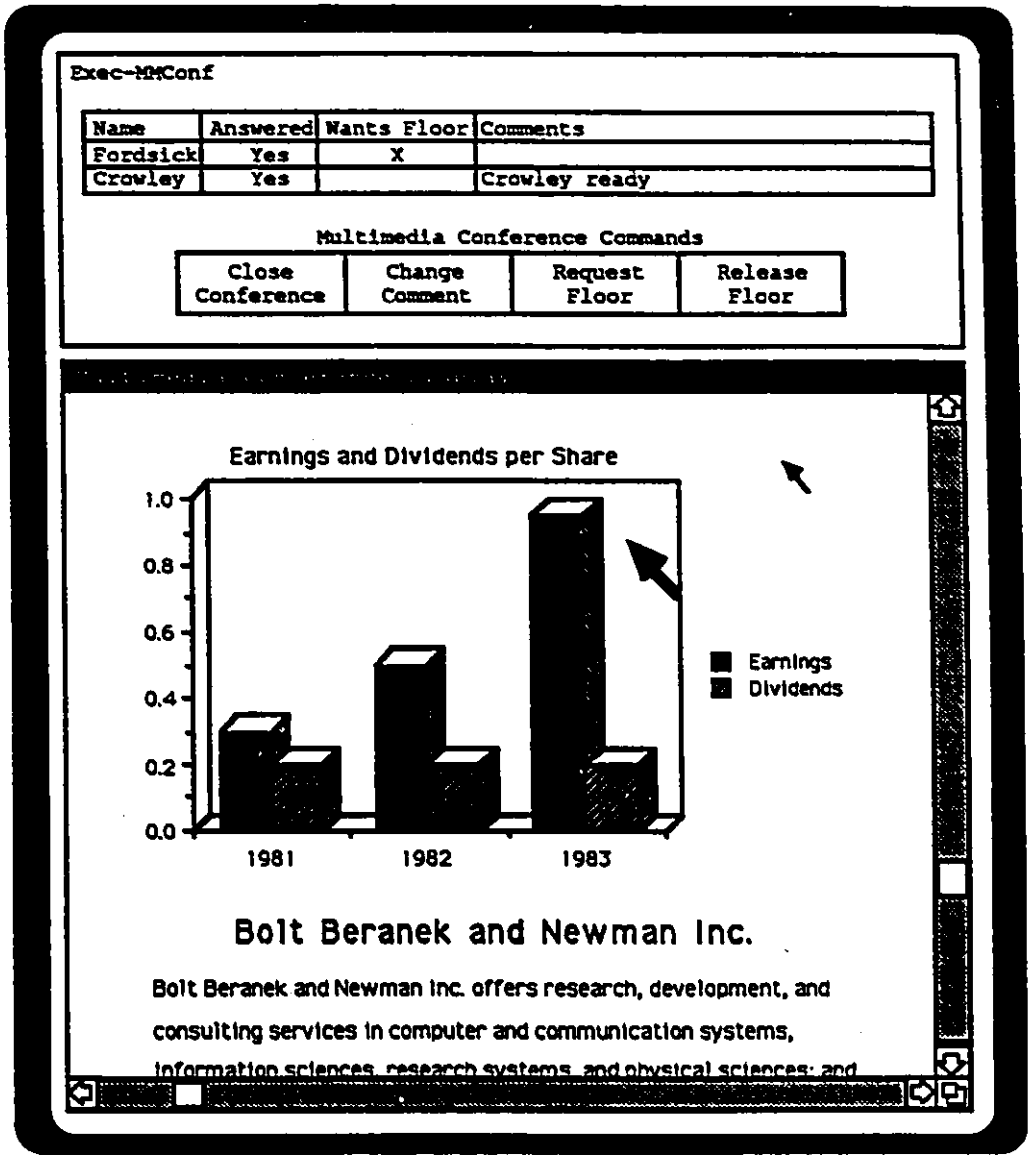


Figure 3.3. The Conference User Interface in Diamond [FOR 86]

User Interface: Each participant of a conference uses a computer with keyboard, mouse and multi-window display. The display has two windows (figure 3.3): a conference status window and a multimedia conference display window. The display window represents the

shared visual workspace and is reproduced on the screen of all the participants. The display is specific to the conference in the sense that it is not a multimedia document but only a succession of multimedia objects which are extracted from documents and pasted into the shared conference window. A shared pointer, distinct from the private mouse pointer of each user, is used to modify the document.

The status window allows each user to request the control of the conference. At one time there can only be one controller of the conference floor and the participants must explicitly request and release the floor to exchange the role. Also, since the voice transmission is done by sending digitized voice samples on the network, audio interaction is only half-duplex and only the controller can speak. This awkward control mechanism and unnatural audio interaction have been recognized as important shortcomings in the use of the system.

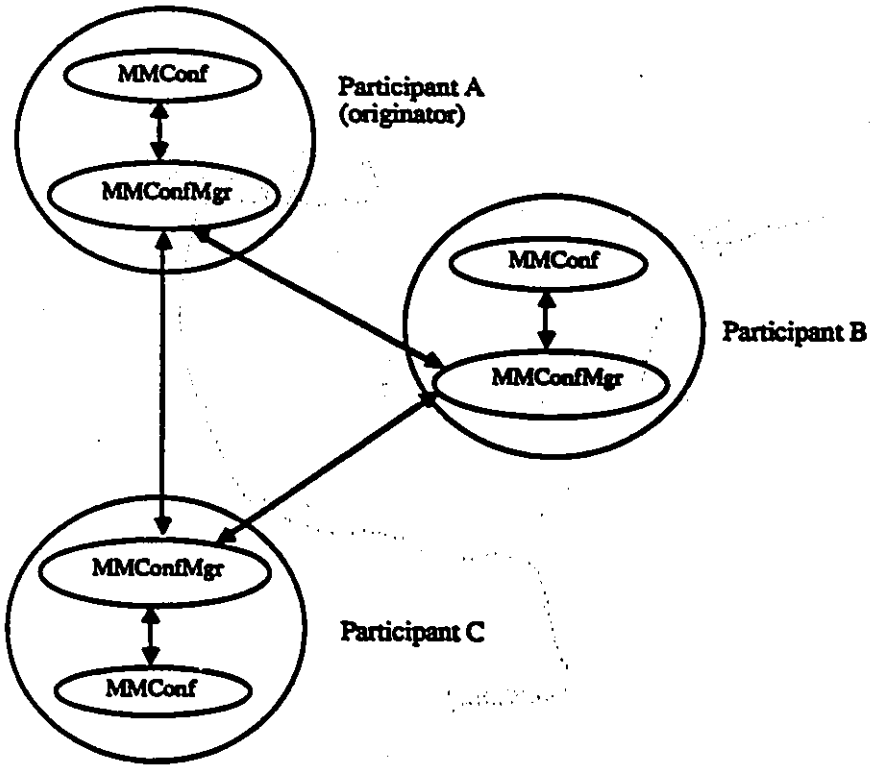


Figure 3.4. MMConf Software Architecture [FOR 85]

Architecture: The conference is implemented by two processes in each workstation (figure 3.4). One, MMConf, manages the user-interface of the conference whereas the other, MMConfMgr, handles the control of the distributed interactions. The MMConfMgr have equal capabilities and are symmetric. The one exception is the MMConfMgr of the originator of the conference: It receives the *Request_Floor* and the *Release_Floor* commands and plays the role of a centralized controller for distributing the control of the floor between participants.

There are two logical streams of data in a MMConf conference:

- The voice/pointer stream originated by the participant controlling the floor carries (X,Y) pointer coordinates and 20 ms frames of digitized voice. This data type does not need to use reliable transmission and is multicasted[1] to the workstations of all participants using a communication protocol which tolerates occasional data errors.
- The command/object stream carries the commands issued by any of the participants and uses a reliable transmission protocol.

One deficiency of the MMConf system was that the multimedia objects entered into a conference could not be modified. One of the goals of computer-based conferencing system is however to provide not only communication between participants but also cooperative editing capabilities of the documents shared during the conference. This area of research is called Computer Supported Cooperative Work.

•RTCAL

RTCAL (for Real Time CALendar) is a prototype for real-time conferencing implemented at the MIT Laboratory for Computer Science ([SAR 84] and [GRE 87]). Although this system

---

[1] A multicast protocol allows a data packet to be sent to a group of receiver stations. Using a point to point communication service, the data packet must be sent repeatedly to each station.

does not provide integrated multimedia services, in [SAR 84] an architecture for real-time conferencing and a common set of functions which can be used to support conferencing in many applications are given.

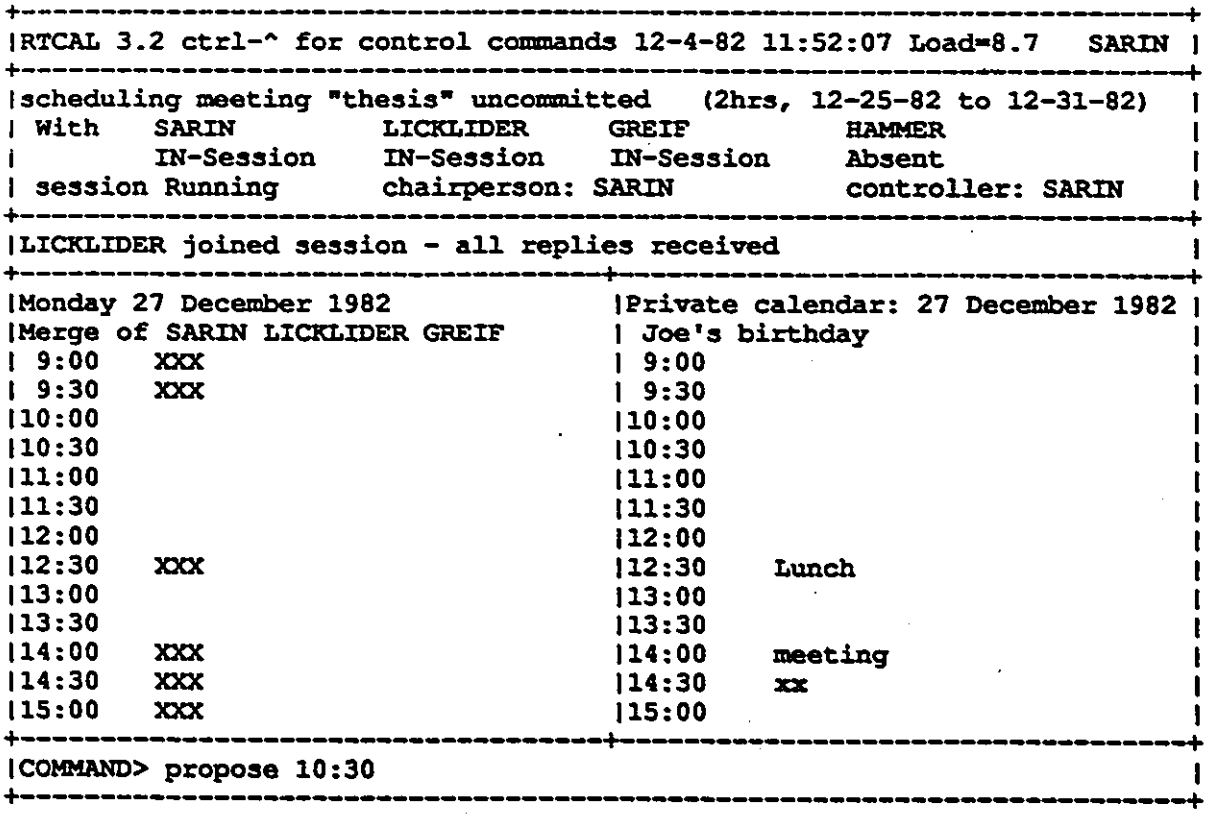


Figure 3.5. RTCAL User Interface

User Interface: The screen in RTCAL (figure 3.5) is divided in a status window showing information relative to the conference, a shared window, on the left, showing a merge of the information of the participants private calendar and a private window, on the right, displaying the private calendar. The terminal allows only textual information, there is no pointing device.

As it is the case in other systems, only one participant has the control of the conference at any time.

Architecture: In [SAR 84] a system architecture is proposed (figure 3.6) which is based on a two main layers:

- The *Ensemble* layer provides the conference control facilities.
- The *Application* layer uses the Ensemble facilities to implement a specific conference.

In fact these two layers may be further subdivided into layers in agreement with the ISO Open Systems Interconnection reference model. The Application layer may have a Presentation layer performing window management and command parsing, while the Ensemble layer will include a transport layer and lower layers.

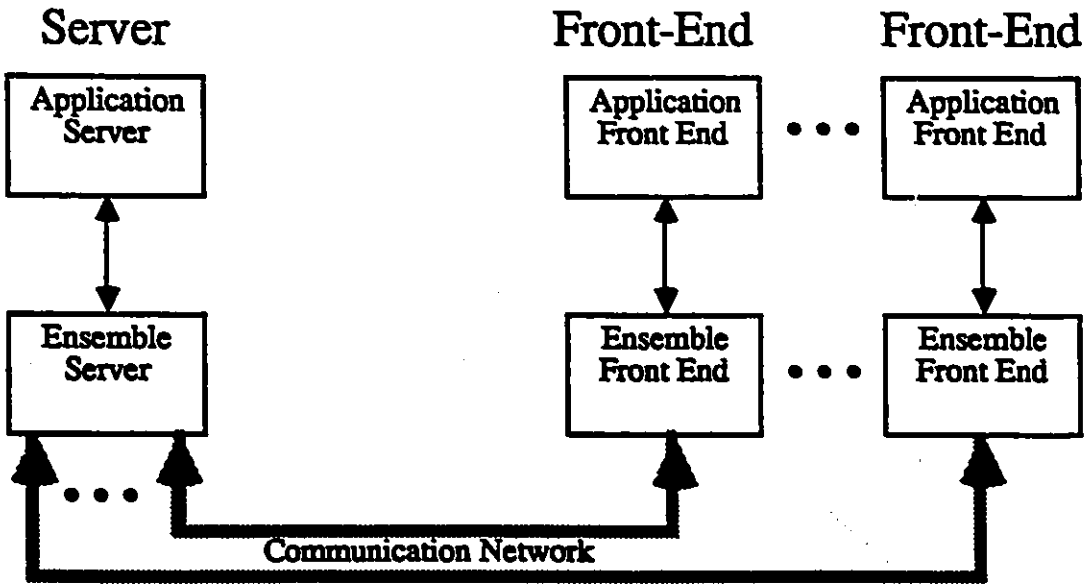


Figure 3.6. Ensemble Conference Architecture [SAR 84].

The author assumes a centralized architecture for any conference (figure 3.6). He defines two types of nodes in the system:

- The Front-End node, which interfaces with the user's output devices and manages the set of objects the user is working on at a given time.

•The Server node, which provide file storage, mail functions and conference control.

In this framework, the author defines the conference as an agreement among a collection of front-end nodes, representing the participants of the conference, and a server node, that controls the conference, to share a specific collection of objects and to allow the manipulation of some or all of these objects via specified activities.

One important concept introduced in this system is the concept of objects. The Ensemble layer manipulates general objects without knowing their content. The main features of the object oriented approach in RTCAL are the following:

•Objects can represent many different things in the system. The most common type is an image which can be shared (in this case it will be displayed) between the participants. There may be other kinds of objects such as high-level application documents or objects used to facilitate participant interaction in the conference such as agendas, minutes, a queue of requests for permission to enter commands.

•Objects are controlled and updated by the conference server node only. This facilitates the synchronization between the users states. Typically an object is updated by the application server in response of a participant command and all local copies of the object are updated by a command from the application server to the Ensemble server.

•Ensemble is not aware of the semantics of an object. It provides service calls like, *Give_Object* (to the participants of the conference), *Update_Object*, *Remove_Object*.

Ensemble provides also the services for creating a conference, adding participants and other conference management functions.

This architecture has been used to implement a shared bitmap facility. The shared bitmap is an advanced form of virtual terminal software. Instead of managing the exchange of only textual information between an application and a user terminal, it also provides graphic operation services. The Bitmap application was divided between a Bitmap Server and a Bitmap Front-End. An application, written for a single terminal could work without modification in multiuser mode using the shared bitmap facility and the Ensemble layer. The shared bitmap facility provides the necessary bitmap and cursor replication on the connected workstations transparently.

3.3 Conclusion

In this chapter we have presented features of two types of services which can exist in computer-based multimedia information system: Data management services and Conferencing services.

Chapter 4
Multimedia Data Management for the MUSIC Project

This chapter describes our propositions for the management of multimedia document in our environment. We distinguish between two aspects: the filing hierarchy and the multimedia document structure.

4.1 Multimedia Database Organization

In the process of defining the structure of multimedia documents, we found it important to distinguish clearly between what forms the filing hierarchy and what forms the documents themselves. Figure 4.1 shows those two aspects of the database organization. We show the filing hierarchy is inspired from the existing manual filing system in radiology departments. The documents (mainly radiographical images and clinical reports) are stored in folders by examination and by patient. This is a natural way to organize a document database. The leaves of this tree are the documents which can be single media (eg. images) or already structured to include multimedia objects.

A filing hierarchy is easily implemented in a relational database management system because its structure is fixed. Figure 4.2 shows the logical structure of a patient database

represented in a relational database management system. Between each level, there is a one to many relationship (a patient folder contains several examinations). At each level we show examples of some possible attributes.

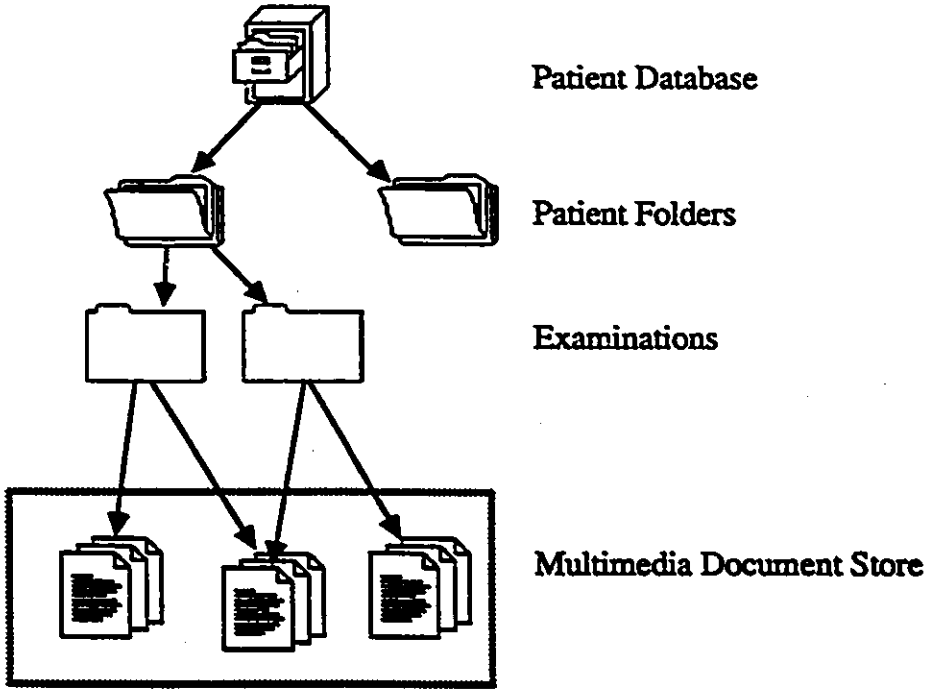


Figure 4.1. Multimedia Database Organization

At the lowest level, we find the documents which can be of many types but the most important for us are the x-ray images and the radiologist's report. The document profile is a part of a multimedia document which is introduced in the Office Document Architecture standard [ISO 88] and which holds information about the structure and the content of the multimedia document it describes. It may contain for instance an abstract, the author's name and the creation date as well as the media types find in the body of the document. Finally, the location of the document body is referred to by an internal reference: Document file location.

In the current testbed facility, the folder hierarchy is implemented using the filing system and not a Database Management System. This architecture is described in the following section.

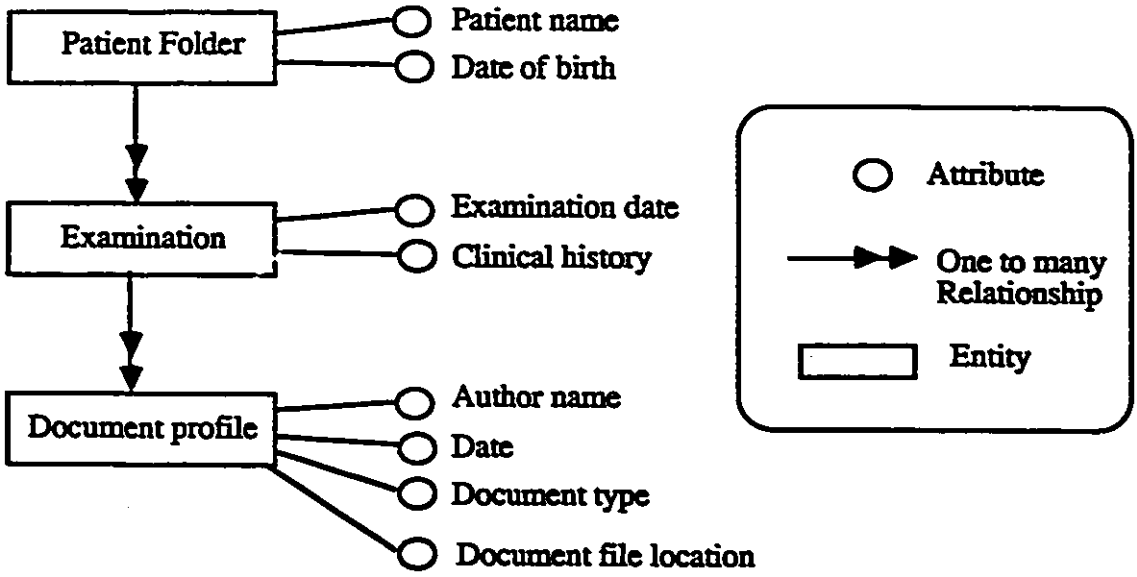


Figure 4.2. The Patient Database in a Relational Database

4.2 A Simple Multimedia Filing System

We describe in this section the structure of the document filing system used in the current testbed facility. To implement the patient folder multimedia database we have chosen a direct approach, which leads to a simple filing system, sufficient to allow a limited number of patient folders to be stored efficiently.

Our system is composed of several workstations attached to a file server. This file server may be seen as different servers in one (a text file server, a voice server and an image server). Figure 4.3 shows how those intelligent workstations access the file server. Each of them runs an instance of the data management agent. The file server provides shared access to the

centralized data. However, to reduce the transmission delay, current images can be downloaded in a local image storage at each workstation.

Based on this architecture, we propose a physical representation for the filing structure. An example of a similar filing structure can be found in [MAN 87].

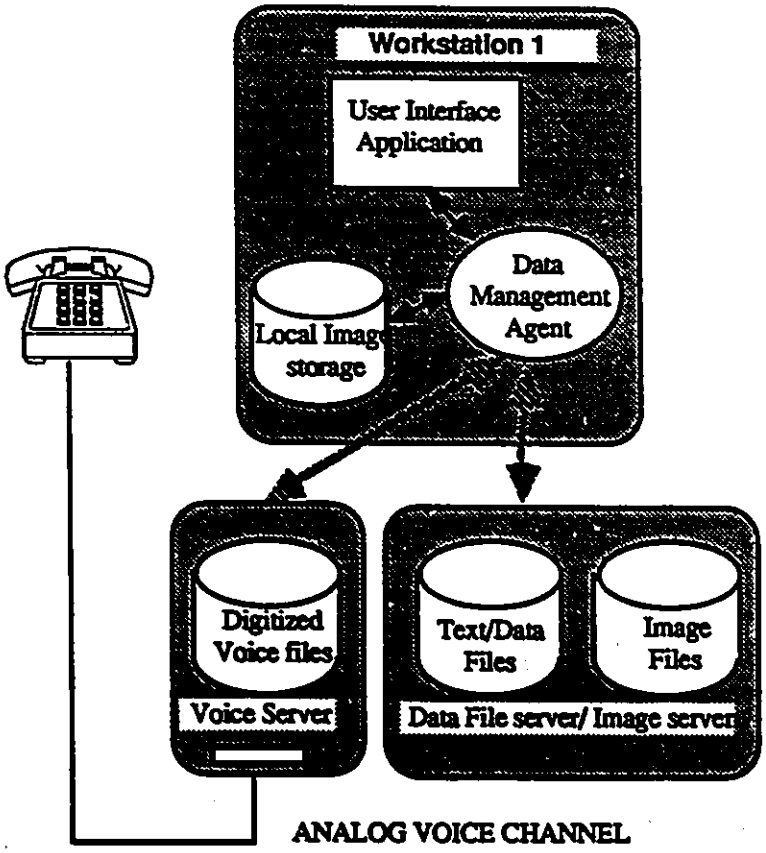


Figure 4.3. Filing System Hardware/Software Architecture

The filing structure is coded in a patient folder file and the content sections are stored in separate files (figure 4.4). The patient folder file holds the patient administrative information and a set of examinations. Each examination holds a sequence of medical data, text/voice reports and image file information. Important management and directory information is stored in an index file. This file allows requesting workstations to browse quickly through the patient

folder list. Each patient folder is stored in a different file and references are made to external image or voice files which are handled by specific servers.

Retrieval of data objects is done at the patient folder level. This means that, when a workstation accesses a patient folder, it copies the patient folder information file from the centralized file server into the local address space of its workstation and when it saves its modifications, it writes the complete file back to the centralized server.

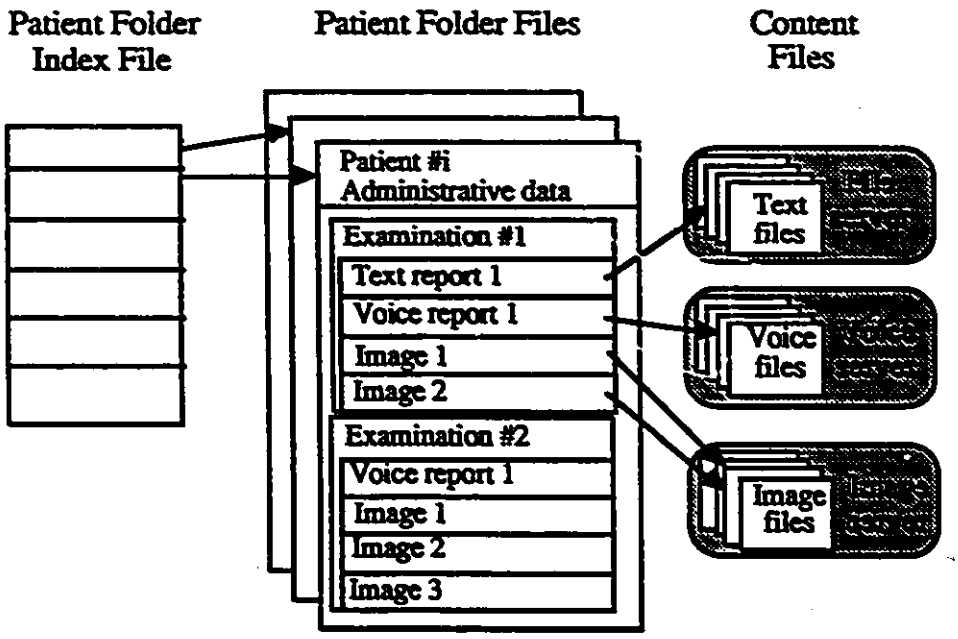


Figure 4.4. Physical Representation of Multimedia Patient Folders

Clearly, such a straightforward design does not provide the same services as an implementation based on a relational database management system. In particular it does not provide powerful concurrency control services at the record level.

4.3 Proposal for a Multimedia Document Structure

In this section we propose a structure for multimedia documents. Our goal is to create a multimedia report which would hold the report of the radiologist together with the relevant radiographical image information.

4.3.1 Requirements

The important multimedia document is the report produced by the radiologist when he studies the x-ray images of a patient corresponding to a specific examination. We will focus our study on the structure of such a report. However, the document structure should be general enough to allow the definition of other document types.

In the current system, a report can only be composed of a single media: text or voice. Our goal is to integrate the images together with the radiologist's report and possibly synchronize the display of the images with the voice section with which it is related. We think that the following services could be useful for a multimedia document architecture in the medical context:

- support the recording of the actions of the radiologist during the creation and the play-back of these actions at a later time. This means that, when a report is read, the system will play the sections of the report and display the relevant images at the same time to produce the same succession of image, voice and text sections which the radiologist has created.
- allow users to make annotations to sections of a document. Annotations are not displayed automatically but only on the request of a user. They could be graphical notes attached to an image which show an important area, or voice comments attached to a report text.

4.3.2 Office Document Architecture

We propose to use the Office Document Architecture standard [ISO 88] as a base for the definition of a multimedia medical report structure. The standard has been defined for the interchange of multimedia electronic documents in an office environment. Several aspects are covered by this architecture:

Document Profile: Each document contains a profile which defines the global characteristics of the document. For instance, the author's name, an abstract and the creation date.

Generic and Specific Structure: The standard uses an object oriented approach and the concept of document classes. Generic documents can be defined; all instances of one document type will share the same characteristics and the same logical or layout structure. A document type defines a skeleton for the logical structure which must be used in all the instances of this document type. For instance, the thesis document type would require the author to include a title, an abstract, a table of content, an introduction and so on.

Logical and Layout structure: The standard defines two type of structural information for the document. The layout structure holds the information for the rendition of the different content objects on a page, the position of paragraphs and footnotes. The logical structure defines the division of the document in logical elements defined by the author (chapters, sections and paragraphs).

The logical structure is the most important part of this standard because it is very general and can be used to define a large variety of document structures. It defines only three types of objects: the document logical root, composite logical objects and basic logical objects. These objects are organized in a tree (figure 4.5) which has the document logical root as root and the

basic logical objects as leaves. All other levels are composite logical objects. To each basic object corresponds a content element (text paragraph, word or graphic). This tree can be defined up to a certain level in a document type definition. Its creator gives a meaning to each object at his will.

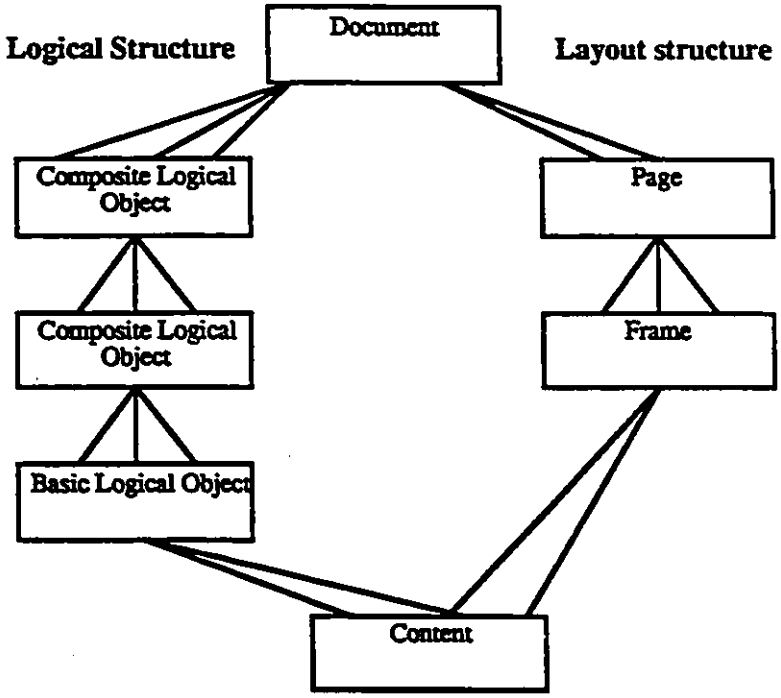


Figure 4.5. ODA Logical and Layout Structure Hierarchy

Until now the content types defined in the standard are limited to text and graphic information. But these will be extended to other content types like digitized voice. We believe this basic document structure can be useful in the definition of a multimedia radiological report. It provides at the same time an encoding of the document for the interchange through computer networks.

4.3.3 Proposed Document Architecture

Although our proposal is based on the Office Document Architecture standard, we could not directly use this standard for several reasons:

- The standard does not yet include digitized voice and high definition images content types.
- The different compatibility levels which are defined relate to office documents and their layout on a page of standard format. In our case, our main medium is the voice which has no such physical representation.

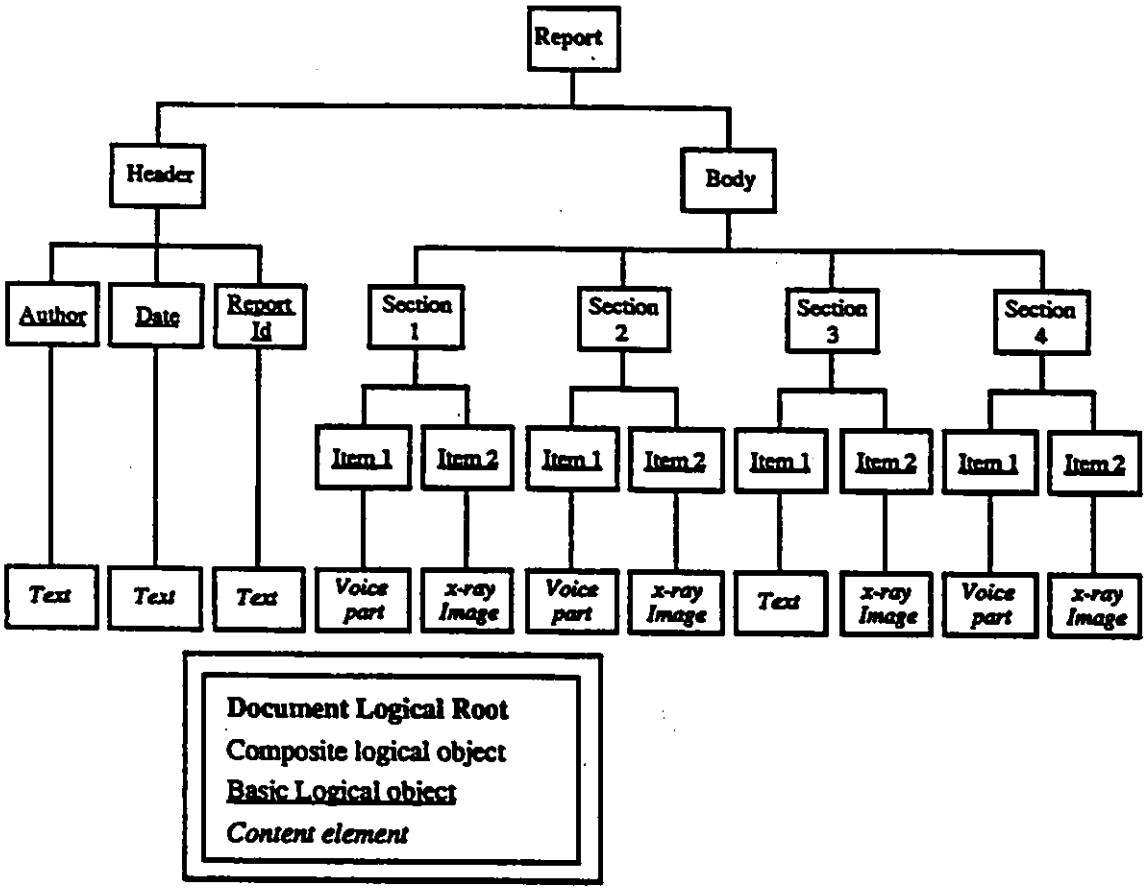


Figure 4.6. Logical Structure of a Multimedia Report

We propose to use only the logical structure defined by ODA as a basis for the creation of a multimedia radiological report. and to extend the content types to x-ray images and digitized voice. Figure 4.6 shows an example of a simplified document structure. It shows how required information can be encoded in basic element fields and how the system can generate a synchronized play-back from this structure.

The document is organized in sections which do not correspond to a standard section but more to a unit which is presented to the user. The user may request to go back a number of sections or to go forward a number of sections. The document can be presented without the action of the user (the system turning the "page" itself) and simulate a play-back of the report, or it might wait at each section until the user requests the next section. At each node of the hierarchy, presentation control attributes can be added to describe the presentation order of the subordinate objects. The default presentation order (as defined by the ODA standard) is the order in which the subordinate objects appear in the tree (from left to right in figure 4.6).

The advantages of our solution are that the document structure definition can be made independent of the content types used at the lowest level.

The synchronization between the basic elements may not be adequate since there is no temporal control information embedded in the definition of the logical objects. The definition of a structure for play-back with severe temporal control may be too complicated to be merged with the logical structure we used here. The document structure we used is a base which other structures can use. Also, the ODA standard defines only hierarchical relationship no annotation information is provided. However, once we defined the different objects of the structure, an external process can record annotations on the clearly defined objects of the document.

4.4 Conclusion

In this chapter we have described different solutions to the problem of multimedia document filing and we investigated the use of the ODA standard as a basis for the definition of multimedia radiological reports.

Chapter 5
Issues in Multimedia Conferencing

After presenting a general model for a conferencing system we will discuss some of the issues which arise in the development and some of the design choices we have to make. However, since this is a general discussion based on a general model, it will not address all the problems related to our specific implementation. Those will be presented in the next chapter.

5.1 Multimedia Information System Model

In order to refine the description and the discussion of the issues involved in the design of a conferencing system, we will use a system model. This model, presented in [GAR 86], takes its origins from the Message Handling System Model from CCITT [CUN 84], [CCI 84]. We use this architecture as an abstract model to present some issues. The functional components in our implementation will be different from those introduced hereafter.

Garcia Luna Aceves proposed a functional model for computer-based multimedia information systems [GAR 86]. This model is an extension of the one currently accepted for computer mail systems, which, as the author claims, should be a base of truly integrated multimedia conferencing systems in an open system environment. Figure 5.1 shows two interconnected remote sites with their functional components. The striped arrows indicate

communication protocols between remote components. Let us have a look at each component and describe its functions.

•Conference Management Agent (CMA)

The functions of a CMA are to:

• keep track of the current conferences

•ensure that the state of the information is the same on all sites.

•ensure of the correct time ordering.

•control the flow and routing of the information between participants of a conference.

•manage the creation of a conference as well as who joins and leaves the conference which may include access right restrictions.

•dispatch the floor control between participants.

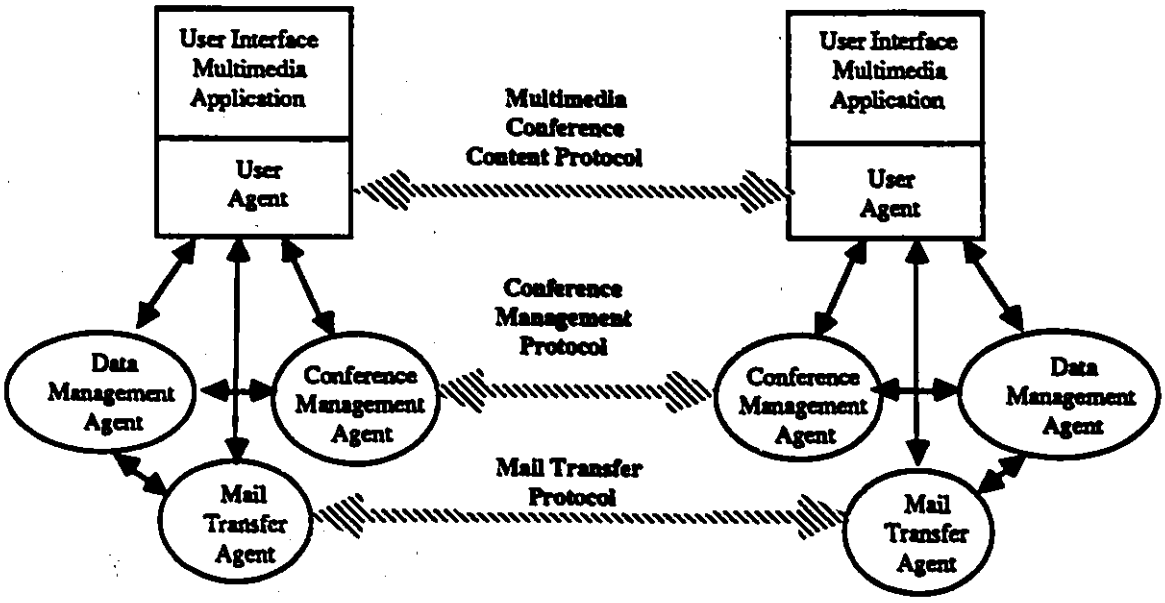


Figure 5.1. System Model

The CMA does not have to know the content of the messages it conveys between user agents. However, it might have to choose among different types of underlying network services depending on the media type the message contains and on what service the user agent requires.

•Mail Transfer Agent (MTA)

The MTA handles the transfer of non real-time messages. This is less important for our interests but it may be seen as a mean of providing asynchronous conferencing services.

The functions of a MTA include:

- •submission and delivery of messages between users.
- •monitoring and creating asynchronous conferences.

It may include access control for user agents and consultation of a directory system. This part of the system is currently standardized in the CCITT X-400 standard series [CCI 84] as well as related protocols.

•User Agent (UA)

The UA supports the following activities:

- •Structure the information exchanged between users in a standard form.
- •Output the information being exchanged during a conference.
- •Instruct the CMA or the MTA to deliver messages.
- •Negotiate with other UA's the multimedia presentation capabilities of a conference given the current workstation possibilities.
- •Access the directory system.

The UA concept has been defined in the X-400 standard series for mail handling systems. Its main role is to provide a unified interface to the Message Handling System and to convert the content information in the messages to whatever format is required by the user it serves.

User Agents serve two types of users in a system: computer processes and human users. Thus we cannot consider that the UA, when it serves a human user, manages the user interface, it is merely a component of the user application software among others like for instance the editor used to create multimedia messages in the case of a mail system.

The interactions between the UA and the user application and user interface during a conference are more important then in the case of a asynchronous mail system. The UA might have for instance direct access to the display to open a window or to move a shared cursor across the screen This depends, however, on the approach chosen for the conferencing protocol as we shall see later.

•Data Management Agent (DMA)

The model architecture shows the DMA is an important part of a multimedia system. There are several reasons for this.

The management of multimedia documents requires different data management schemes like the ones used for ordinary data. Reducing for instance the duplication of contents at a local site is an important issue since some media may have heavy storage requirements. On the other hand, this same space requirements may make replication of data necessary to reduce the transmission delays when participants of a multimedia conference are both in remote locations. Therefore the DMA must provide distributed and replicated data management services.

Also, as stated in our description in the previous chapter, the shared workspace in a conference can include documents stored in a database. The management of the conference workspace becomes a problem of distributed database which must be controlled by the DMA.

5.2 Basic Protocol Layering

Let us give an overview of the protocol layering necessary to implement a multimedia conference. We will use the layering proposed by Garcia Luna Aceves [GAR 86] as a base for our discussion.

The protocol layering proposed (figure 5.2) is interesting because it brings the distinction between a *conference content protocol* and a *conference management protocol* with respect to the layering of protocols in a conferencing system.

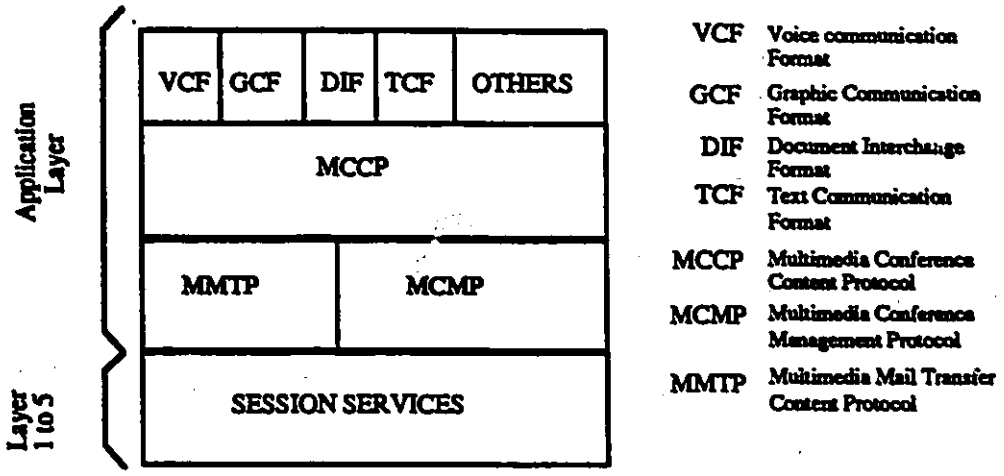


Figure 5.2. Basic Protocol Layering

•The Multimedia Conference Management Protocol (MCMP)

The protocol messages supported by this protocol are of two main types:

•*Conference management calls*, which contain messages for the creation of conferences, the joining and leaving of participants and the passing of the floor between active participants.

•*Conference content messages*, which are the messages distributed to all the users. The Conference Manager is responsible for ensuring that these messages arrive in the correct order to all the user agents.

•The Multimedia Conference Content protocol (MCCP)

The MCCP contains the messages which modify the shared workspace. There are two types of messages in a MCCP [GAR 86]: *Workspace Manipulation Calls* which contain the commands to modify the data objects of the shared workspace, and *User Agent Control messages* which do not affect the workspace directly but contain management information exchanged between User agents.

The shared workspace of a conference is composed of both documents or, more generally data objects, stored in a database and rendition of information on the workstation devices. Example of the latter are image or graphic displays on a screen or voice output on a speaker. Thus there are mainly two types of *Workspace Manipulation Calls*: those acting on the temporary rendition of the workspace and those which modify permanently the documents in a database.

Example of such protocol messages are: Graphic drawing commands which modify the display on the workstation screen. Messages containing entire documents to be stored locally, modifying the workspace in the database. Even commands to a multimedia editor which can modify both the visual aspect on the screen (i.e. by showing the modification to the document) and the workspace once the document is saved.

•Information Interchange Formats

The use of standards for the interchange of information is important in the context of an open system architecture. Most systems are built using private data structures which make it difficult to exchange data with different systems. We believe that the two most important standards in regard to real-time conferencing are: standards for the exchange of multimedia documents and graphic standards. The former are used to send complete documents to remote locations and the latter can be useful when the conference is implemented using the virtual terminal approach (see discussion in section 5.3.2).

Examples of multimedia document interchange are the Office Document Interchange Format (ODIF) from the European Computer Manufacturers Association (ECMA) [ISO 88] or the Document Interchange Architecture from NEC [SAK 85]. One interchange format defined by ECMA is used in the mixed mode facsimile systems.

All these standards are based on office documents. This implies that they are based on a page by page layout and that they support only text and low resolution graphics at this time. The content types will eventually be extended to other medias like voice and high resolution images. Also, document interchange format, although fundamental and well suited to mail handling systems are not sufficient. In the case of real-time conferencing, there might be a need for direct screen manipulation on the workstation. This depends of course of a design choice which we will discuss in the next section.

In the domain of medical system and PACS, one important standard to consider has been proposed by the American College of Radiology and the National Electronic Manufacturer Association (ACR/NEMA). This proposal embraces protocols at all seven ISO layers for the exchange of digital x-ray images [ACR 85]. It contains also a format for the interchange of images.

5.3 Issues in Real-Time Conferencing

After having seen an overview of the different building blocks of a multimedia conferencing system, we will study some issues and alternatives which arise in the design. We will relate it to our specific interest in developing a conferencing prototype in a local environment as opposed to an open system.

In this discussion we will assume that our system is based on multimedia documents composed of structured text, voice and graphics which includes high resolution images.

5.3.1 Document Location

Two aspects are important in the choice of the document location in the system: the transmission delay and the management of distributed data.

•Centralized data management

If there is only one data management agent updating the data during a conference, the control mechanism is simplified. However, this means that the part of the document shown on the screen must be displayed on request. Thus transmission delays occur during the conference. The solution is to distribute the documents before the conference starts and to synchronize the update of the copies.

•Distributed and replicated data management

The management of distributed documents is quite easy when there is no editing allowed on the data. Therefore this method is often used in document oriented teleconferencing systems which display non editable facsimile type documents. The system must only ensure that the correct document is displayed on each workstation.

In our system we have mixed the two approaches. Each workstation holds a copy of the high-resolution x-ray images which are never modified. The main document editing functions (content and structure editing) are done on a centralized copy of the files.

5.3.2 Software Architecture Alternatives

The issue of centralized or distributed data management is also closely related to an architectural choice which is described in this section. Figure 5.3 and figure 5.4 show two different alternatives for the software organization of a conference system. This choice has an influence on the load on the network and delays, the reusability of existing software and data consistency problems.

To support our discussion, we need a finer model of the software of a workstation. The *Workstation Agent* proposed in a user interface model in [LAN 87] is responsible for providing the basic, device dependent I/O abstractions. It includes primitives for the display of graphics, for the reading of keyboard and mouse events and for other media types. The *Multimedia Application* is typically the user interface running on the workstation, it interacts with the user through the primitives provided by the Workstation Agent and uses other system services through the network. A finer description could separate between the dialog manager and the actual application software. In the following discussion and on the figures, the conference software is intentionally hidden for the sake of clarity.

The **centralized architecture** is characterized by the fact that only one instance of the multimedia application exists. Each workstation acts as an intelligent terminal accessing the centralized application. Such an architecture is inspired from the terminal linking systems in which the shared visual workspace realized by replicating the text output on multiple terminal screens instead of one. In [SAR 85],this solution is denoted as the *virtual terminal approach* because the conferencing software acts as a virtual terminal controller. It duplicates the output

from the application (for example text or drawings) to the workstations and multiplexes the input events (typically mouse and keyboard events) from the workstation agents in such a way to realize the functionallity of the conference. Instead of interacting directly with a user's terminal, the program interacts with a virtual terminal controller.

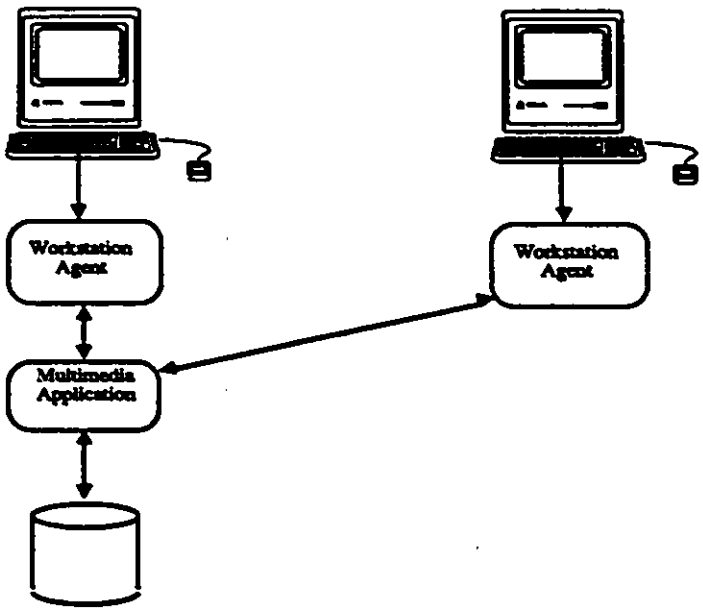


Figure 5.3. Centralized Architecture

This approach has several advantages:

•An application written for one user can be used in conference mode without modifications. This was an important issue even in our system implementation. The development of multimedia applications and user interface software is already a difficult task in a single user environment. Using this approach, if these two components, application and interface, use the primitives defined in the workstation agent, they can be integrated easily in the conference environment.

•There are no concurrency problems since the users share only an image of the data. Only the application has access to the database.

•The users see the same interface in conference mode as in autonomous mode.

This approach suffers however from several disadvantages[LAN 86]: The users may observe poor response time. For example, such an architecture will certainly not provide the same refreshing rate for pointer tracking as in single user mode. Each mouse displacement event must undergo two transmissions. The conferencing software will act as a bottle-neck if it receives too much I/O traffic. Also, since large amount of low level data may be generated, the load on the network will be important. Typically this is the case when the graphic protocol is defined at a very low level requiring screen bitmaps to be sent at each screen update.

However, current graphic protocols exist which use highly structured graphical data reducing the possible need of retransmission of data on the network [LAN 85]. The X-Window system [SCH 87] is an important example of such a protocol. It is a windowing system which is becoming a standard among workstation manufacturer and might eventually be used by several application software packages. X-Window acts as a virtual terminal protocol since it has been designed to be network transparent. An application can control windows on a remote workstations. It supports commands to draw basic shapes, to perform bitmap operations, to save segments of images and even to define objects like menus and windows. In this system, the content of a window can be saved so that when the window is obscured and displayed again, it can be updated locally and does not require the application to send the content through the network.

The **replicated architecture** (figure 5.4) has the advantage of lower response time. Since the documents are replicated at each site, the access to a specific file or multimedia document is very fast. This architecture is useful when the conferees are located on distant sites. Audio-document teleconferencing systems (see chapter 3) use typically such an architecture. Two main problems have to be solved in order to realize the shared workspace. One is how to

synchronize the two applications, the other is how to solve the problem of distributed and replicated data management.

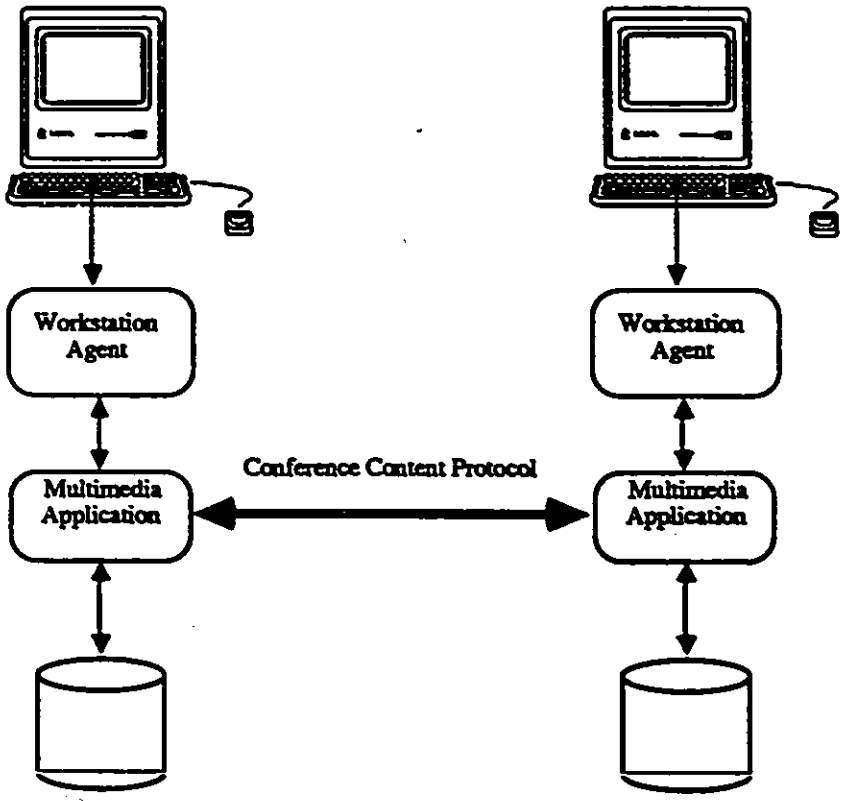


Figure 5.4. Replicated Application Architecture

Two possibilities exist for the synchronization of the applications. A straightforward solution consists in replicating the input events generated by the workstation in control to the other workstation. This approach has been tested at Stanford University [LAN 86] and has the advantage of simplicity. As in the centralized case, the application software does not have to be modified. However, a separate protocol must control the data management of the shared workspace.

Two situations can occur: if the application access files on their local storage (figure 5.4), the protocol must ensure that these files are replicated on both workstations, if they access

files on a centralized file server, the protocol must resolve access and update conflicts. We suppose here that our system is based on data files and not on a distributed database management system which would take care of concurrency problems.

The second synchronization protocol solution would be to use high level commands which could be interpreted by both applications. This again supposes that programmers include at each step of their algorithms the possibility of interaction with another application. In other words, they must rewrite their programs for specific conferencing environment. The application becomes a distributed application.

5.3.3 Sequencing and Floor Passing

One important service required to ensure the proper realization of the shared workspace, is the sequencing service. In the most general way, one must ensure that all action initiated on different workstations are performed in the same order on all the sites participating in the conference. In current conferencing systems, this problem has usually been addressed by different solutions.

Centralized sequencer: All commands are treated by one process and are therefore guarantied to be broadcast in the same sequence to each conferee workstation. The centralized architecture discussed in previous section could support such a solution. However, such a solution may result in ***interleaving*** problems. If two users request an action at the same moment, one of them may initiate an unwanted command because when his request will be executed by the system, the context of the shared workspace may already have changed [SAR 85]. For example, supposing the system uses mouse event replication as conference content protocol, two users can select a menu item in one state of the user interface. The first command once executed can display a sub-menu on which the second mouse event will be applied. The result will be inconsistent with the command requested by the user.

Floor passing mechanism: By allowing only one user to issue commands to the shared space at any time, interleaving problems are avoided. There are still many options on how to implement floor passing control mechanism during the conference.

In BBN's MMConf [FOR 85], one user has the control and he is said to own the floor of the conference. In this implementation the mechanism to allocate the control was awkward: the floor had to be requested and released explicitly. A dynamic allocation of the conference floor to each requesting user seems to be a more natural way. Requesting the control should be as simple as moving the mouse device attached to the station. In EMCE [AGU 86] for instance, the floor control is distributed by voice activation, who ever speaks gains the control of the floor. As in a face to face meeting, the one who speaks louder has the control and we must trust human protocols to avoid conflicts.

However, giving one user the control of the complete shared workspace is only one option among many. In RTCA[illegible], there is one cursor per user and they are all shown simultaneously on the screen. Participants in a conference can work concurrently on different parts of a shared calendar by making reservations on separate sections.

5.4 Conclusion

In this chapter we have introduced a general system model and protocol layering which can be applied to the realization of a conferencing service in the context of an open system. Different alternatives have been discussed for the data management, the software architecture and sequencing mechanism.

Chapter 6
A Multimedia Conferencing System for Medical Application

6.1 Requirements for a Multimedia Conference

We want to provide a shared workspace facility allowing a radiologist and an attending physician to discuss the clinical situation of a patient. It must provide services comparable to the available in a face-to-face meeting.

Currently, both physicians have to meet in person in a consulting room where they can display x-ray images on a viewbox. Both of them have access to the clinical data relative to the patient. They can exchange views and point out different areas on the images. Eventually, they can make modifications to documents in the patient folder. For instance, they can request further examinations to be made.

A computer-based conferencing system in this environment must provide:

- a voice channel allowing users to talk to each other.
- screens showing the relevant images simultaneously at both sites.
- a shared pointer displayed on all screens which represents the natural finger pointing of the physician or radiologist showing specific areas on an image.

•access to different patient folders and documents in those folders.

•eventually a way of creating and editing existing documents inside the patient folder.

A real-time conference consists of an information component and a discussion component. In our case, the discussion component is implemented by a voice channel and a shared pointer and the second component by the multimedia document itself. As stated before, the system should provide the same services to the user in conference mode as in autonomous mode. In our case, the system will provide access to most of the multimedia functions offered by the user interface as they have been described in chapter 2. Those operations consist mainly of two types: document consultation (eg. viewing of images, reading of text reports or listening to voice files) and document editing (eg. creation of new reports).

6.2 Relevant Features of the Conference in the MUSIC Project

Some user and system constraints for the conferencing system are specific to this medical application. The system should mainly provide a two-party conferencing capability. There seems to be no application for multiparty conference service at this early stage of study. The specific application we considered in the MUSIC project was to provide a conferencing system between two remote locations in the hospital, one in the emergency department and the other in the radiology department.

The conferencing capability is integrated with the rest of the system software. We want to provide the same view of the multimedia services in conference mode as in single-user mode.

Medical imaging applications have demanding delay requirements. The management problems caused by the large amount of data generated by radiographic equipment made it

necessary to use a distributed architecture for the image storage subsystem. These points have an effect on the way the conferencing workspace is realized.

6.3 User Interface in Conference Mode

The user interface in conference mode is separated into two parts: the shared workspace (ie. the information which is replicated on each workstation) and the private workspace (ie. the part of the workspace which is local to a workstation). We will describe briefly both of those elements.

The private workspace: When a workstation is in conference mode, a private window appears at the top of the control screen (figure 6.1). This window provides buttons to issue conference commands and message panes to display conference status information. The user controls this window with a private cursor (small arrow on figure 6.1).

The conference control button allows the user to quit and close the conference, the floor control button allows the user to request or release the floor. A conference status pane displays the names of the participants in the conference.

The shared workspace is composed of several elements:

- The shared visual workspace:
  - The graphic display of the control screen.
  - The high definition images displayed on the image monitor.
  - A shared cursor which is controlled by the floor holder (large cursor figure 6.1).
- The shared data files: The set of local image files which will be eventually be displayed on the image monitor during the conference.

•The shared *audio-workspace* which allows both users to speak to each other and to listen to the voice files.

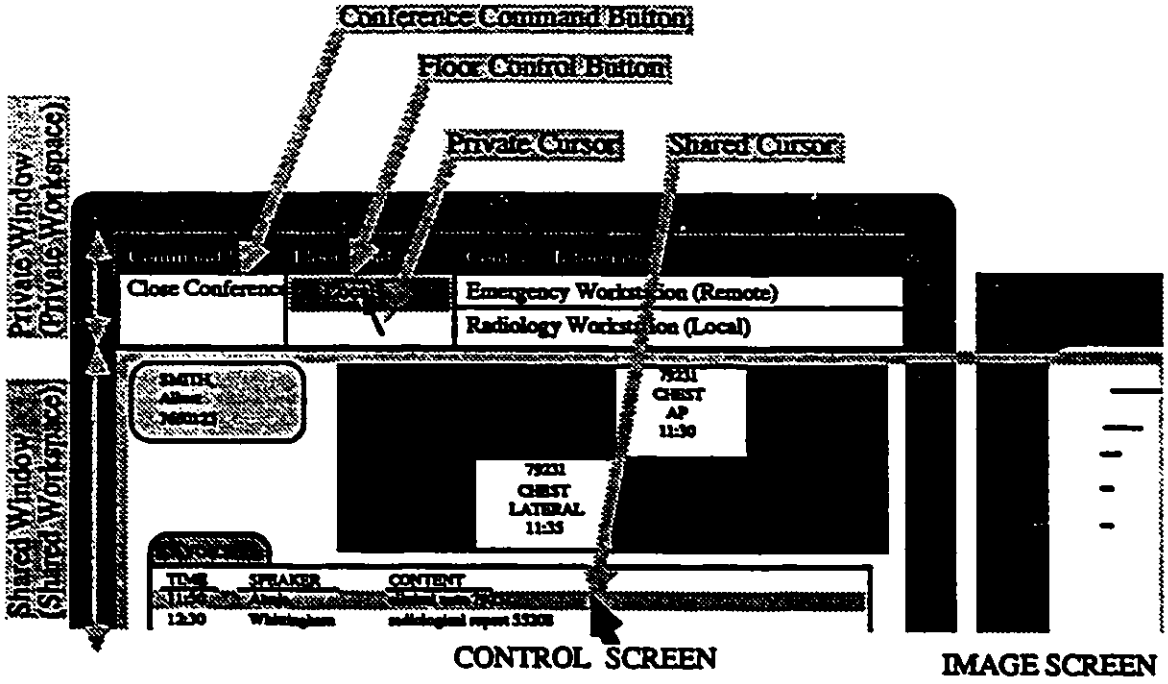


Figure 6.1. User Interface in Conference Mode

The realization of the shared audio workspace is still an open question. Since the audio information is transmitted under analog form to the workstations (refer to chapter 2), this media was not integrated in our protocol. It may become an issue when using ISDN network services. Voice services were not fully implemented in the early testbed facility but a solution to implement a shared audio workspace would be to initiate a three-party conference call between the voice server and the two participants' workstations. Such audio-conferencing capabilities are available on modern digital PBXs.

We consider the image files as elements of the shared workspace, because, due to the real-time requirements involved in the specific medical environment we are studying, the image files must be distributed into each workstation to lower the image transmission delay.

The participant who holds the floor can control the shared workspace using his pointing device (mouse) to move the shared pointer (pointing activity) or to select different menu and list items. On the image screen, he can use the cursor to point at specific areas on the image screen. He can also enter information using his keyboard.

This realization of the shared workspace, and particularly of the user interface, is one proposition. Other solutions are possible and have been implemented in other systems. For instance, the system could display two cursors of different shapes, one for each participant, as is done in the shared bitmap system [SAR 84]. In the COLAB experiment at XEROX PARC [STE 87], it was noted that the display of multiple cursors on a screen could be distracting.

Another question was to define the extent of the shared workspace. For instance, we could consider that only the images are part of the shared visual workspace and not all the operations on the user interface. However, our system did not provide sufficient patient information on the image screen to identify the image.

6.4 Operation of the Conference System

When a user decides to initiate a conference, he issues a menu command to the conference agent which displays thereafter the conference user interface. A window appears at the top of the screen holding the conference commands (Figure 6.1). The user clicks the Call button and is presented with a list of workstations with which he can initiate a conference. Once the user chooses the name of a workstation, the conference agent issues an open request to its conference manager. The conference manager then calls the remote conference manager

which in turn puts up a menu asking the remote user if he wants to accept or reject the conference request. If the conference is accepted, the two workstations user interfaced are initialized in the same state.

The originator of the call is initially the floor holder. The role of both statious are symmetric and both users can request the floor. In order to obtain the floor, the user has just to click a button which indicates the current floor state. When the floor button is marked **Remote**, the remote user has the floor. Clicking the button in this state means that the local user requests the floor. In our system, the floor is immediately given when requested and the button will then change to **Local** indicating that the workstation has the control of the conference. Clicking the local button will not yield any result, since the floor can only be requested and not given. We find that this method of allocating the floor is simple and efficient. Conflicts are resolved by the users relying on human protocols.

Either participant in the conference can terminate a conference. The conference is closed when one of the two users clicks the **Close** button.

6.5 Design Overview

For our design, we have followed the general model described in the previous chapter. The conferencing software is organized in two layers: the conference management layer provides services to the conference agent (the user agent layer) through a set of primitives which are described in the following section.

In each workstation there is a conference agent, a software module which interfaces with the other software modules in the workstation and with the user. It uses the primitives provided by the conference management layer to control the conference and to exchange

conference content messages. It is responsible for formatting the conference content messages (figure 6.2).

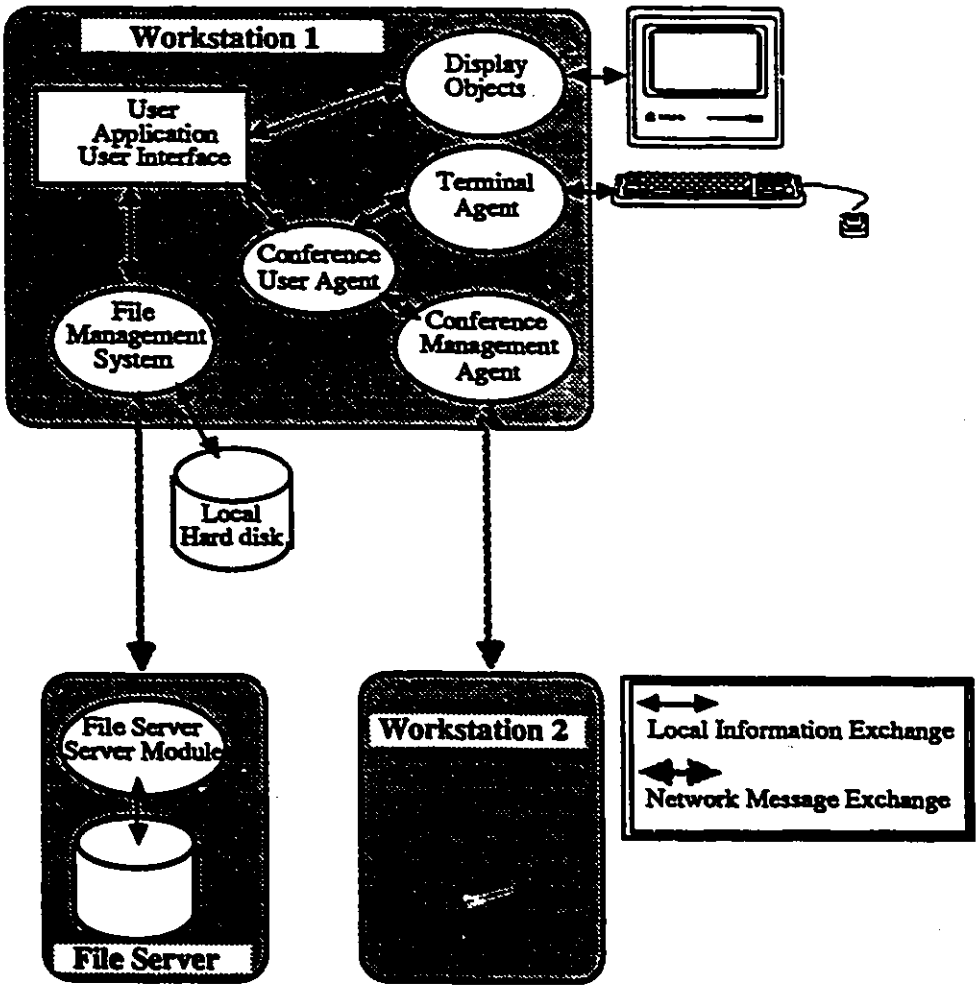


Figure 6.2. Conference Software Architecture

The conference is realized by a replicated application architecture. The user interface software is replicated in each workstation with the data files. They are synchronized at the beginning of the conference and then the input events generated at the workstation which has the floor are sent to the workstation which does not have the floor. This ensures that the same actions are performed on both workstations. The conference content messages exchanged

between the workstations hold mainly the cursor and keyboard events. This straightforward mechanism allows us to use the existing user interface software without modifications in conference mode. This is an important advantage. However, to work, one must make sure that all data files used in a conference are replicated on the different workstations.

6.6 Conference Management

The conference management layer groups conference control functions which are described by abstract service primitives. Those primitives are limited to our specific design where only two workstations can participate to a conference.

6.6.1 Conference Set-up

A conference between two workstations is initiated by using the open primitive.

C_OPEN_request(conference description, participant name)

C_OPEN_indication(conference description, initiator name)

C_OPEN_response(response)

C_OPEN_confirm(response, participant name)

Once the remote participant has responded to the conference request, the initiator decides to start the conference using the start primitive.

C_START_request()

C_START_indication()

C_START_response(success indication, [failure reason])

C_START_confirm(success indication, [failure reason])

The close primitives provide a confirmed service for the disconnection of the conference connection:

C_CLOSE_request(reason)

C_CLOSE_indication(reason)

C_CLOSE_response(success indication, [failure reason])

C_CLOSE_confirm(success indication, [failure reason])

Finally, a non confirmed closing service is provided to abort the conference in case of network error. This service is inititiated by the conference manager in case of network error or by the conference agent in case of a severe error:

C_ABORT_request(reason)

C_ABORT_indication(reason)

6.6.2 Floor Control

Each user must request the floor before he can send conference content messages to the other workstation. A set of primitives is provided to request and release the floor:

C_PLEASE_FLOOR_request()

C_PLEASE_FLOOR_indication()

C_GIVE_FLOOR_request()

C_GIVE_FLOOR_indication()

Different types of floor passing mechanisms can be implemented using these primitives. The responsibility of the floor management is left to the user agent layer. In our implementation for instance, the floor is immediately released by the floor holder when he receives the C_PLEASE_FLOOR_indication primitive.

6.6.3 Data Transfer

We have defined two types of data transfer primitives used for the two service requirement. One is an unreliable transfer service used for redundant information like the cursor tracking information in which case the loss of a data element is compensated by the next data element. The reliable data transfer for data elements which must be delivered.

C_RELIABLE_DATA_request (data)

C_UNRELIABLE_DATA_request(data)

C_DATA_indication(data)

6.7 Conference Content Protocol

The actual information component of the conference is conveyed by the data primitives of the conference management layer.

In our implementation, each workstation captures events from its local input devices. If these events are actions on the private workspace, they are treated locally. Otherwise, they are sent to the remote workstation, if the user has the floor. If the event was only a mouse displacement, it will be transmitted on an unreliable data transmission service. If it was a mouse button or a keyboard event, it will be sent through a reliable data transmission service.

6.8 Conclusion

This chapter gives an overview of the design in our conferencing application. In the next chapter we will describe more precisely its implementation aspects.

Chapter 7
Implementation

This chapter presents an implementation of a conference capability on the testbed facility. Before describing the conference implementation, we will describe the system environment

7.1 System Environment

Three aspects of the development environment are important for the description of the implementation: the operating system used on the workstations, the language in which the software was written and the services provided by the network layer.

7.1.1 Concepts of Object Oriented Programming in Smalltalk

We have used the Smalltalk/V™[1] environment and programming language to implement a prototype of the conferencing system. Smalltalk/V, an implementation of the Smalltalk–80™[2] language on the IBM PC/AT line of computers [DIG 86], was also used to develop the initial prototype of the user interface for image consultation in the MUSIC testbed facility.

---

[1] Smalltalk/V is a trademark of Digitalk Inc.

[2] Smalltalk-80 is a trademark of Xerox Corporation.

The basic concepts used in the Smalltalk language are objects, classes, messages and methods.

Objects

Objects have their own data or local memory, called instance variables. An object recognizes a set of procedures for manipulating this private data. These procedures are called methods and are invoked by sending messages to an object. Objects are protected data structures since the internal state of an object can only be modified by sending a message to it.

All Smalltalk variables are containers for objects. A variable contains a single object pointer which can point to an object of any class. There are three kinds of variables: instance variables which are the component part of an object, temporary variables created during the activation of a method and shared variables used by many other objects.

Classes

Classes are the program modules from Smalltalk. They describe the data structures (objects), algorithms (methods), and external interfaces (messages). Every object is an instance of some class and all the objects which are instances of the same class have the same instance variables and respond to the same messages.

Messages can be sent to classes to create new instances of the class. For example by sending the message new: 10 to class String, the following code:

```
aString := String new: 10.
```

creates an instance of class String with 10 characters and assigns it to the variable aString.

Messages and Methods

All processing in Smalltalk is done by sending messages to objects. Messages are the external interface by which we can perform operations on an object. Methods are the algorithms which are performed by an object in response to a message.

Any message can be divided into three parts: a *receiver*, a message *selector*, and zero or more *arguments*. The *receiver* is the object to which the *message* is sent. It must implement the method corresponding to the message selector. Let us show some examples of messages:

```
7 factorial
```

this is an unary message sent to an instance of class Integer which yields an instance of class Integer.

```
7 between: 2 and: 24
```

this is a message sent to an instance of class Integer (7), with selector **between:and:**, and arguments 2 and 24. The result of this message will be true or false.

Objects and Encapsulation

Two important aspects of the object oriented paradigm are encapsulation and hierarchy [DIE 87].

Different object types can respond to the same messages. In this way, the object is responsible for the implementation of the message, not the sender of the message. For instance, the message **display** can be defined for objects of type point or of type rectangle. If we were to display an object (point or rectangle), we would write:

```
someObject display.
```

This sends the message display to the object. The code is independent of the object type and can still be used if new object types must be displayed. In C the code for such an operation is different and cannot be reused:

```
switch (someObject->kind)
{
case POINT:
  displayPoint(someObject);break;
case RECTANGLE:
  displayRectangle(someObject);break;
default:
  error(); /*unexpected type*/
}
```

Class and Inheritance

Inheritance is a technique which allows new classes to be built on top of less specialized classes. Classes form a hierarchy, consisting of a root class called Object, and many subclasses. A subclass inherits all the variables and the methods of its superclass and adds new variables and new methods.

In our implementation for instance, we designed a class called Channel. Its instances represent a network session connection. As its instance variables we can find for example a buffer in which we copy the received data and the name of the computer on the other end of the connection. The class has a method which creates a new channel instance:

```
aChannel := Channel     connect:   aLocalObject
                            to:    anObjectName
                            on:    aRemoteComputer
```

and each instance has a method for sending a buffer to a remote computer:

```
aChannel send: aBuffer.
```

When we wanted to use the datagram service on the network, we designed a subclass of this class called **DatagramChannel**. This class used some of the instance variables of its superclass (a buffer for receiving) and redefined a new method for creating the channel:

```
aChannel := DatagramChannel    openFor: aLocalObject
                               groupName: aGroup.
```

This method is different since there is no connection with a remote computer when the channel is created. However, the same method inherited from class **Channel** is used to send a buffer:

```
aChannel send: aBuffer.
```

Smalltalk Programming Paradigm

In languages like Pascal or C the programming methodology is:

> Define the functions and procedures needed to solve the problem and implement the necessary algorithms to perform these procedures.

In Smalltalk, the methodology is:

> Identify the objects appearing in the problem, for each object type define the messages which act on these objects (the procedures are defined for each object type), and implement the methods which are the algorithms for these messages.

This programming style allows us to structure our software by defining objects which represent more closely the entities appearing in the real world problem.

7.1.2 Smalltalk System Features

The choice of Smalltalk as development environment and language for our prototype has been motivated by several points:

•The Smalltalk environment is ideal for fast prototyping and testing of software. The graphical interface to the system allows the user to browse quickly through its source code and add, delete or modify its methods. Methods are compiled one by one and therefore the code produced is compiled in an incremental fashion. Small corrections, when an error is detected, can be made immediately.

•The power of the inheritance scheme allows us to use existing code defined in classes by defining subclasses. Important changes can be made to modify the system's behavior just by designing subclasses of existing objects.

•The powerful graphical capabilities of Smalltalk makes it an ideal tool to develop user interfaces. This was important for developing the user interface for the image consultation application and for the conferencing system.

•Many operating system features are incorporated in the Smalltalk language [ING 81]:

Storage management is automatic. Objects are created by sending messages to their class and are disposed of when no further reference to them exists; The File system is available to objects like Files and Directories; The Display and keyboard handling is defined using objects like Forms, Cursors, TerminalStreams.

7.1.3 Network Architecture and Services

Before we describe our implementation, it is important to present the network services provided in the workstations and on which our software was built. As we explained earlier, our IBM PC/AT based workstations were interconnected by a SYTEK CSMA/CD (Carrier-sense multiple access with collision detection) broadband network operating at 2 Mbit/s. In each computer, a 6120 PC Network adapter card from SYTEK supports five layers of data

transfer protocol. The functions provided by the card to the host computer are session level services.

The interface to the network adapter card follows the Netbios (Network basic input/output system) standard defined by IBM. The use of such a standard makes the application programs independent from the actual communication protocols used to implement the functions. All the software implementing the Netbios services is executed on the network adapter card processor and memory. Let us describe the most important services:

Name Management:

The functions ADD NAME, ADD GROUP NAME, DELETE NAME control the management of a name table in the local adapter. Names are 16 character strings and are used to establish sessions (between unique names) or to send broadcast datagrams (for group names).

Session Support:

The functions CALL NAME, LISTEN, HANG UP control session connections. Sessions are established between two names on the network.

The data exchange is controlled by the functions: SEND, CHAIN SEND, RECEIVE. The normal command succession to establish a connection and to send data between two computers is described in table 7.1.

A maximum of 64 Kbytes can be sent during the reliable transfer provided by a session connection.

Commands issued by computer A (sender)	Commands issued by computer B (receiver)	Comments
ADD NAME ("A")	ADD NAME ("B")	Both computers register unique names on the network.
-	LISTEN ("A")	Computer B is ready to accept an incoming connection from name "A".
CALL NAME ("B")	-	The session is established and a session number is assigned to the connection.
-	RECEIVE	Computer B is ready to receive data for this session and allocates a buffer for the incoming data. This call is pending during the session time-out defined in the connection parameters.
SEND	-	Computer A sends data to computer B. If computer A isn't ready to receive, this call may time-out. Here the data transfer is completed.
HANG UP	-	Both computers can hang up the session connection.

Table 7.1. Sequence of Commands on a Session Connection

Datagram Support

SEND DATAGRAM, SEND BROADCAST DATAGRAM, RECEIVE DATAGRAM, RECEIVE BROADCAST DATAGRAM are the commands used to send small messages (2100 bytes maximum size). The messages are not acknowledged and there is no guaranty of delivery. Datagrams can be sent to a name, group name (several users can have registered the same name and receive the same messages) or to all the users.

Otther functions are not shown here. In the next section we describe how we interfaced the Netbios services with the Smalltalk environment.

7.2 Software Modules for Communication Services

7.2.1 General Description

The interface between Smalltalk and the network functions provided by the network adapter card can be organized in several layers (figure 7.1). The purpose was to provide some simple services to hide some complexity at the highest level. At the highest level we have network service users which basically deal with Channel or DatagramChannel objects. These objects model session connections and datagram ports on the network.

The NetworkManager is an object which keeps track of all existing Channel objects, and dispatches incoming events from the network to the destination channels. Incoming events can be data events, error events, disconnection requests. It also listens to incoming connection requests for channels.

At a lower level, NetworkControlBlock objects provide the basic command interface to the Netbios software. Objects of this class reproduce exactly the structure of the Network Control Blocks. Network Control Blocks are the structures which encode the Netbios commands and their parameters. Two steps are needed to execute a Netbios command: first one must encode the command code and the command parameters in the Network Control Block, and second issue a software interrupt call which gives the pointer to the structure to the Netbios software.

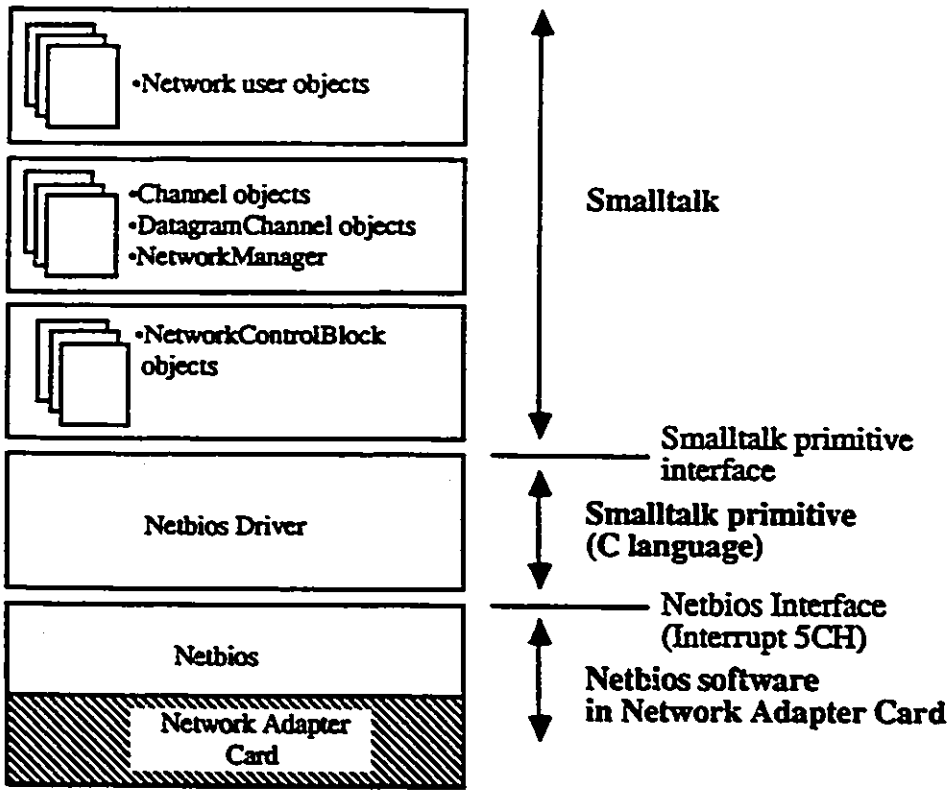


Figure 7.1. Communication Software Layers

Let us describe the different layers in more detail.

7.2.2 The Netbios Interface

As we stipulated before, the basic command interface to Netbios uses a structure called Network Control Block to encode the command, its parameters and its result. In order to make the Netbios commands available from Smalltalk, we defined a class called NetworkControlBlock. Instances of this class are arrays of bytes which hold the encoding and the methods of this class define the different Netbios commands. The commented listings describing this class can be found in Appendix I.

When a command request is made, the method encodes the parameters of the command and the command id in the data structure and calls a primitive method to execute the

command. Primitive methods are calls to a memory resident program in C or Assembler which does the processing of the command. In figure 7.1 we described it as the Netbios driver layer.

The network Primitive

The basic role of a primitive is to take the Network Control Blocks and data buffers from Smalltalk and to deliver them to the Netbios interface. It executes any commands without knowing their meaning.

There are two types of commands in Netbios: wait and no-wait commands. Wait commands are like procedure calls where control does not come back before the command has been executed or has erred. Using the no-wait option the caller regains control immediately and can then later check that the command has been completed.

Function id	Description
0	Reset the static buffer and Network Control Block list in the primitive.
1	Execute the command and update the receiver (aNcb) if wait command, otherwise copy the ncb and the buffer to internal variables.
2	Return true if the command was completed.
3	Return the new values of the ncb and the buffer once the command was completed.
4	Cancel the command if it was pending.
5	Poll all pending commands and return the Network Control Block of the one that has completed
6	Returns an error code which is set when the primitive failed.

Table 7.2. Primitive Functions

In the case of wait commands, the primitive program passes the pointer to the structures defined in Smalltalk to the Netbios interface. However, when no-wait commands are requested, the Network Control Block and the data buffer defined in Smalltalk must be copied to static structures and it is the pointers to these structures which are then passed to the Netbios interface. The reason for this choice is that Smalltalk objects are moved in memory during the automatic garbage collection process. For the Netbios interface, the pointer to the Network Control Block must remain valid at any time because results are written to it.

The primitive call has the following syntax and parameters:

```
aNcb netPrimitive: anInteger with: aBuffer.
```

anInteger is the code used to call 7 different functions which are described table 7.2.

Command Execution

Several steps are involved in the execution of a Netbios command. For instance to execute a Netbios wait command from Smalltalk the following steps are required:

1- Create an instance of class NetworkControlBlock:

```
aNcb := NetworkControlBlock new.
```

2- Request the command by the appropriate method:

```
aNcb addName: "EMERGENCY".
```

3- Check that there was no error:

```
aNcb  ifError: [ "handle  error"].
```

And to execute a Netbios no-wait command from Smalltalk:

1- Create an instance of class NetworkControlBlock:

```
aNcb := NetworkControlBlock new.
```

2- Request the command by the appropriate method:

```
aNcb receiveNoWait: aBuffer.
```

3- Wait until the command is completed:

```
aNcb    checkCompleted
        whileFalse:[].
```

4- Update the buffer and the receiver once the command is completed:

```
aNcb    getResult.
```

5- Check that there was no error:

```
aNcb    ifError: [ "handle  error"].
```

The use of the no-wait option is natural for commands which must be pending for a long time. In our implementation, there is always a listen pending and the workstation is ready to accept incoming session calls.

7.2.3 The Network Manager

There is one instance of class Network Manager. The functions of the Network Manager are:

•to listen for incoming session calls from any name on the network and to create a Channel object corresponding to the incoming request.

•to keep track of open channels and to pass network events to the Channel object.

•to initialize the environment on start-up and after a network error. This includes the registration of the network name of the local computer on the adapter card name table.

7.2.4 The Channels

The function of a Channel is to monitor a session interaction between two objects on different computers. A channel provides the means to exchange data on a previously established session connection.

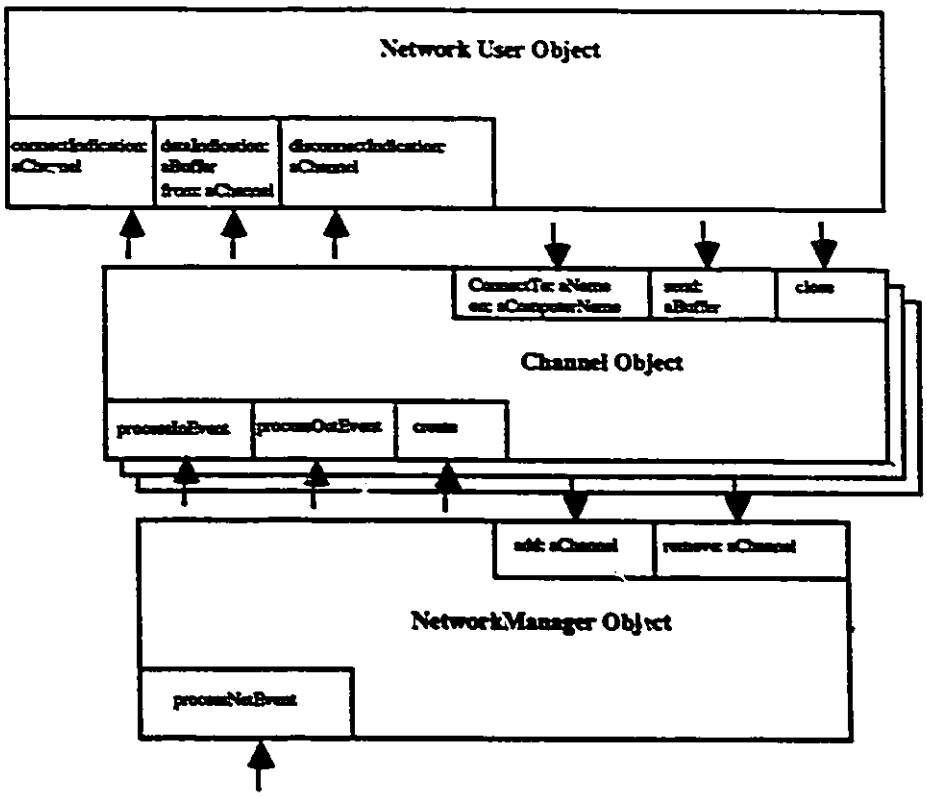


Figure 7.2. Communication Objects and their Interaction

Any object having a global name in the Smalltalk environment and which implements the methods to receive a connection request, a data request and a disconnection request, can be called from a remote computer. The parameters for a connection request are the name of the remote computer and the name of the object on the remote computer which will receive the session call (see figure 7.2).

A channel object receives event indications from the network manager object. It recognizes these events as data indications, connection or disconnection requests or errors and passes these events to its owner by a procedure call.

Outgoing messages are first queued on an outgoing message queue and sent by no-wait send commands. This generates a network event at each send command completion. The channel responds to these events by sending the next data element in the queue if there is one.

A channel is always ready to receive a message (a receive command is always pending). When a data buffer is received from the other end of the connection, it is immediately passed to the user object in the dataIndication call.

7.2.5 Performance

This communication architecture was satisfying by its ease of use and generality. The fact that it was written in Smalltalk made it very flexible. However, the data throughput that we could achieve between two computers was much lower than that of a C implementation. This was due to the slower execution rate of methods in Smalltalk compared to a program compiled in C and to the fact that we had to copy buffers and Network Control Blocks passed between Smalltalk and the C primitive.

The performance was nevertheless acceptable because we used it to send short messages. Our main concern was the rate at which we could send messages. For instance if we want to send pointer information and move a cursor at distance, we must have a minimal rate so that the cursor does not jump on the screen. The test we made indicated that a rate of 20 messages per second could be attained using small buffer sizes (20 bytes). This rate was enough for our needs.

We shall now describe the implementation of the conference services.

7.3 Software Modules for Conference

7.3.1 Implementation

The conference is implemented mainly by two software modules: the conference manager and the conference agent. They interact with the existing user interface software to provide the real-time conferencing capability. Figure 7.3 shows the architecture of the software modules. The prototype was built to be used for two workstations. In this implementation only two workstations could be participating in a conference at a time, multiparty connections were not provided.

The conference manager module provides a set of service primitives to the higher level to open and close a conference connection, to pass the floor between the two participants and to transfer data between the two stations. It does not know the semantics of the data which are exchanged. However it provides two traffic types: reliable and datagram. As shown figure 7.3, the conference manager uses a channel object representing a session connection for the reliable transfer of data and a datagram object for the transfer of data which does not need guaranteed delivery service.

The conference agent uses the service of the conference manager and interacts with the other software in the workstation. It manages the dialogue with the user for the set-up and the closing of the conference, and for the floor passing command. It also does the coding and decoding of the shared workspace commands between the two workstations.

During a conference, the control screens and image screen displays on both workstations are synchronized, forming what is called the shared visual workspace. The shared workspace is created during a conference by having the data files (the x-ray images) duplicated on both

workstations and then replicating the input event from the mouse and keyboard devices from the workstation holding the floor to the other workstation (refer to chapter 6).

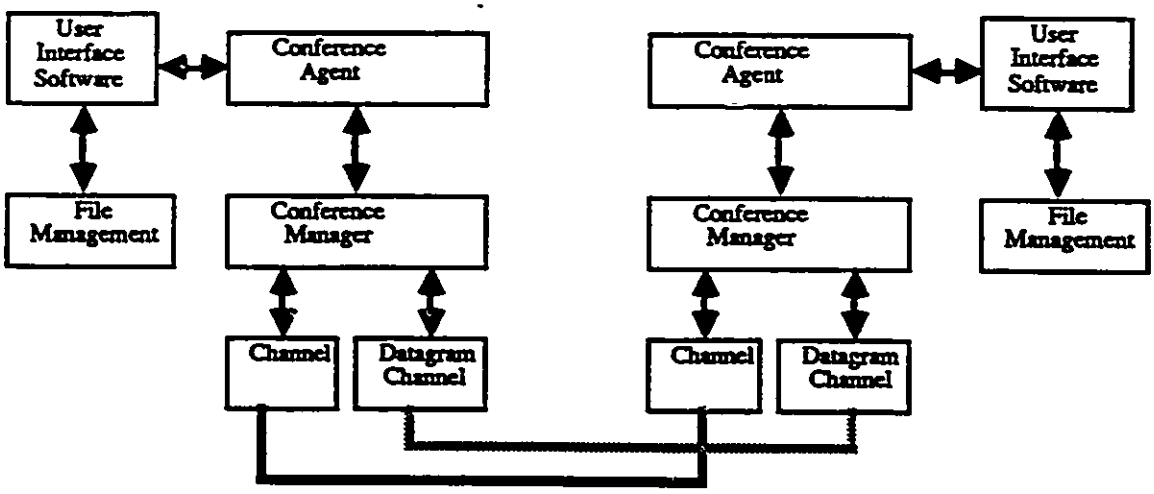


Figure 7.3. Software for Conferencing.

7.3.2 Service Interface

The service interface between the conference manager and the conference agent is implemented by a set of methods. Incoming primitives are implemented by methods in the conference agent object and called by the conference manager. Conversely, outgoing primitives are implemented by methods in the conference manager object and called by the conference agent.

The name of the methods corresponds to the name of the primitives as defined in chapter 6. For instance, the method named **pOpenReq:** corresponds to the primitive C_OPEN_request(parameters). The colon after the method name means that an argument is expected which encodes the parameters of the primitive as a string. These methods are described in the definition of the classes in appendix I. The primitives correspond to messages sent through the network by the conference manager. These messages are encoded on

character strings. For instance, the primitive **pOpenReq**: "<parameters>" is carried on the network by the string "mOpen" appended to the string "<parameters>".

The following service primitives have been implemented:

•Conference open, close, start, abort:

pOpenReq:
pOpenResp:
pOpenConf:
pOpenInd:

pStartConf:
pStartInd
pStartReq
pStartResp

pCloseReq:
pCloseResp:
pCloseConf:
pCloseInd:

pAbortInd:
pAbortReq:

•Floor control:

pGiveFloorReq
pGiveFloorInd
pleaseFloorReq
pleaseFloorInd

•Data transfer:

pRDataReq:
pUDataReq:
pDataInd:

7.4 Conclusion

Using the general model described in chapter 5, we have implemented a conferencing system in our medical workstations. The use of Smalltalk as a programming language allowed us to build a good prototype of such a system. However some characteristics of Smalltalk and of the operating system have been cumbersome:

•The version of Smalltalk/V we used had no support for multitasking. Multitasking capabilities are a natural way for handling network protocol layers for instance. Implementing the modules responsible for the network interface and the conferencing functions in tasks separated from the user interface tasks would have enhanced the modularity of our software.

•The 640 KBytes memory limitation of the underlying DOS operating system together with the high memory space needed by Smalltalk to run properly made it difficult to load all the software into the workstations.

•The operating system did not provide an easy interface to the network services and we had to build an interface architecture to make the network functions available in the Smalltalk environment. Many operating systems available today already provide a good interface to network services by predefined library functions which can be called from a C program. Now there are even some operating systems in which the interprocess communication mechanisms are network transparent. Messages can

be send from one task to another in an uniform way whether the tasks run on the same computer or on a different computer.

•Finally, we must pay for the power and versatility of the Smalltalk language by some degradations in performance compared to a software coded in C language.

These performance problems made it difficult to test effectively the real-time pointing capability. In fact, the refreshment rate of the shared pointer on the screen was irregular and difficult to use. This was due to the fact that we used dynamic buffer allocation for each packet on the network and the automatic garbage collection in Smalltalk, which takes place to collect buffers no longer beeing used , takes one or two seconds, causing the shared pointer mouse to freeze during this time interval.

However, we could test the floor assignment scheme and found it to be efficient and simple. Also, the prototype allowed us to test one possible user interface organization in conference mode: the use of separate areas on the screen, one controlled by the local user for the conference management, and the one for shared workspace.

One important issue was not addressed in this prototype, namely the interaction between the workstations in conference mode and the database. In our test, the patient database and the image files had to be duplicated on both workstations prior to the conference. At the time of the test, the user interface program did not load its database files or image files from a centralized file server. In a complete system, the commands exchanged between conference agents must include file management commands. This includes two important aspects:

•when image files are requested for display on a workstation, the other workstation must be checked to see if the same files are available or if they must be loaded from a centralized server.

•if we want the patient database to be modified during a conference, allowing for instance document to be added to a patient folder, we must take care that only on workstation has the right to update the database.

Chapter 8
Conclusion

In this thesis we have investigated the application of concepts used in the area of office communication systems to the field of medical communication systems. The goal of the MUSIC project is to build a prototype of a system which will handle medical data in a radiology department under electronic form, providing services for communication, archiving and production, replacing thereby the current manual x-ray examination process.

In the first part we have proposed an architecture and a data model for the management of patient folders and the images, text and voice reports contained in these folders. We also proposed a document architecture for the medical patient folder which permits the integration of the different information types contained in a patient folder.

Further we investigated different architecture alternatives and protocols for the implementation of a conferencing service in our testbed facility. We used a layer organization based on the model defined in the X-400 standard for message handling systems. The conference management layer groups the services for the conference establishment, the management of participants and the transfer of information inside a conference. The user agent layer is responsible for data transfer between workstations for the realization of the shared workspace using the services of the lower layer. It also interfaces with the user to

control the conference parameters. This basic model allows us to define service interfaces and protocols which are independent of the specific implementation architecture.

The implementation of the conference prototype used this two layer model. However different options were available for the software architecture and the realization of the shared workspace. We opted for a decentralized architecture based on the replication of input events and the duplication of image files. Such a design proved to make the task of integrating the conference software in the existing environment easier. However the problem of the data exchange between the software on each workstation and a centralized database or file server (holding the patient folder database) was not examined in this work and may prove complicated to solve.

We think that in the case of a conferencing system aimed to be used in a local environment, the use of a centralized architecture as discussed in chapter 5 may prove more interesting. In this architecture, one instance of the application program residing in a server node duplicates its output on client workstations using a graphics protocol. The image files must still be replicated on each workstation but all database access mechanisms are simplified since they occur only through one instance of an application.

Our real-time conference system actually implements only the on-line data and image components; voice communication is established by using the telephone network. Given that voice conferencing can be implemented on a packet network, it may be feasible to transmit both voice and data over a single communication network. Since the transmission requirements of voice and data are quite different, and not all computer networks can provide the desired level, networks capable of providing circuit switched and packet switched facilities, such as FDDI, could be used.

Further enhancement of the system can be proposed. Extending conference management services to accept several participants, dynamic joining and leaving of participants in a conference and testing conferencing through long-haul network are left for further study.

Bibliography

[ACR 85] ACR/NEMA Standard Publications, nº 300-198x, September 1985, Digital Imaging and Communications.

[AGU 86] L. Aguilar, J. J. Garcia Luna Aceves, "Architecture for a Multimedia Teleconferencing system", ACM SIGCOMM'86 Symposium, pp. 126-136, Aug. 1986.

[BIJ 87] Bijl, Koens, Bakker, de Valk, "Medical PACS and HIS: Integration needed", SPIE Medical Imaging, vol. 767, 1987.

[CCI 84] CCITT X-400 Recommendations on Message Handling, October 1984.

[CHR 86a] Christodoulakis, Theodoridou, Ho, Papa, Pathria, "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System", ACM Transactions on Office Information Systems, Oct. 1986.

[CHR 86b] Christodoulakis, Faloutos, "Design and Performance Considerations for an Optical Disk-Based, Multimedia Object Server", IEEE Computer, Dec. 1986.

[COE 87] F. Coelho, "Conception d'un service d'Archivage Multimedia dans un Environnement Bureautique Ouvert", These de doctorat de l'Université de Paris-VI, 27 mai 1987.

[CUN 84] D. Cunningham, "Electronic Mail Standards to Get Rubber-stamped and Go Worldwide", Data Communications, May 1984.

[DIE 87] J. Diederich and J. Milton, "Experimental Prototyping in Smalltalk", IEEE Software, May 1987, pp. 50-64.

[DIG 86] Digitalk Inc., "Smalltalk/V Tutorial and Programming Handbook", Nov. 1986.

[FOR 84] H. Forsdick, "Initial Experience with Multimedia Documents in Diamond", IFIP Conference on Computer-Based Message Services, May 1984.

[FOR 86] H. Forsdick, "Explorations into Real-time Multimedia Conferencing", IFIP Conference on Computer Message Systems, 1986.

[GAR 84] J. J. Garcia Luna Aceves, A. Poggio, "Multimedia Message Content Protocols for Computer Mail", IFIP Conference on Computer-Based Message Services, May 1984.

[GAR 86] J. J. Garcia Luna Aceves, "Towards Computer-Based Multimedia Information Systems", IFIP Conference on Computer Message Systems, 1986, pp. 61-75.

[GIB 87] Gibbs, Tsichritzis, Fitas, Konstantas, and Yeorgaroudakis, "Muse: A Multimedia Filing System", IEEE Software, Mar. 1987, pp. 4-15.

[GRE 87] I. Greif, S. Sarin, "Data Sharing In Group Work", ACM Transactions on Office Information Systems, Apr. 1987.

[HOR 84] W. Horak, Krönert, "An Object-Oriented Office Document Architecture Model for Processing and Interchange of Documents", GLOBECOM'83, San Diego 1983, pp. 1245-1249.

[HOR 84] W. Horak, "Concept of the Document Interchange Protocol for the Telematic Services-CCITT Draft Recommendation S.a", Computer Networks 8, 1984.

[HOR 85] W. Horak, "Office Document Architecture and Office Document Interchange Formats: Current Status of International Standardization", IEEE Computer, Oct. 1985.

[HOR 86] W. Horak, "Document Editing and Entry based on the Office Document Architecture Standard ECMA 101", GLOBECOM'86.

[ING 81] D. H. H. Ingalls, The Xerox Learning Research Group, "Design Principles Behind Smalltalk", BYTE, vol. 6, n° 8, pp. 186-298, Aug. 1981.

[ISO 88] ISO, "Information processing - Text and office systems -office document architecture (ODA) and interchange format (ODIF).", ISO/DIS 88.
DIS 88/1: Part 1 - General Introduction.
DIS 88/2: Part 2 - Office Document Architecture.
DIS 88/3: Part 3 - Document Profile.
DIS 88/4: Part 4 - Office Document Interchange Formats.
DIS 88/5: Part 5 - General Principles of Positionning and Imaging.
DIS 88/6: Part 6 - Character Content Architecture.

[IVI 87] Ivie, "Future Trends and Requirements for Databases in the PACS Environment", SPIE Medical Imaging, vol. 767, 1987.

[KOT 87] Kotera and Tanigawa, "Design of a document Oriented Teleconferencing System", GLOBECOM'87.

[LAN 85] K. A. Lantz, W. I. Nowicki, M. M. Marvin, "An Empirical Study of Distributed Application Performance", IEEE Transactions on Software Engineering, vol. SE-11, nº 10, October 86, pp. 1162-1173.

[LAN 86] K. A. Lantz, "An Experiment in Integrated Multimedia Conferencing", CSCW'86 Conference on Computer-Supported Cooperative Work, Dec. 1986, pp. 267-275.

[LEE 84] Lee, Woo, Lochovsky, "Officeaid: An Integrated Document Management System", ACM 84

[MAN 87] N. J. Manchovich, "An Image Database Structure for Pediatric Radiology", SPIE Medical Imaging, vol. 767, 1987.

[MUS 87] "Multimedia system for integrated communications (The MuSIC project)", Department of Electrical Engineering, University of Ottawa, Sept. 1987.

[NEM 83] K. Nemeth, "Principles of the Document Interchange Protocol for CCITT Telematic Services", IEEE Communications Magazine, Mar. 85.

[NIS 87] Nishihara et al., "High Speed Image Transfer Network for PACS", SPIE Medical Imaging, vol. 767, 1987.

[OOS 87] H. Oosterwijk et al., "PACS Implementation: A Tailored Approach", SPIE Medical Imaging, vol. 767, 1987.

[POG 85] A. Poggio, J. J. Garcia Luna Aceves, "CCWS: A Computer-Based Multimedia Information System", IEEE Computer, vol. 18, nº 10, Oct. 1985, pp. 92-103.

[ROG 86] D.C. Rogers, R. Wallace, B.G. Thompson, D.M. Parrish, "Information flow analysis as a tool for PACS development", SPIE Medecine XIV/PACS IV, vol. 626, 1986, pp. 690-697.

[SAK 85] Sakata, Ueda, "A Distributed Interoffice Mail System", IEEE Computer, October 1985, pp. 106-116.

[SAR 84] S. Sarin, I. Greif, "Software for Interactive On-Line Conferences", ACM-SIGOA Conference on Office Information Systems, Jun. 1984.

[SAR 85] S. Sarin, I. Greif, "Computer-Based Real-Time Conferencing Systems", IEEE Computer, vol. 18, nº 10, Oct. 1985, pp. 33-45.

[SCH 87] Scheifler, Gettys, "The X Windows System", ACM Transactions on Graphics, Apr. 1987.

[SEG 86] Segura, "L'imagerie Medicale en Reseau", Sciences & Techniques, Avril 1986.

[SHI 82] T. Shick, R. F. Brockish, "The Document Interchange Architecture: A Member of a Family of Architectures in the SNA Environment", IBM System Journal, vol. 21, nº 2, 1982.

[STE 87] Stefik, Bobrow, Foster, Lanning and Tatar, "WYSIWIS Revised: Early Experiences with Multiuser Interfaces", ACM Transactions on Office Information Systems, vol 5, nº 2, April 1987, pp. 147-167.

[THO 85] Thomas, Forsdick, Crowley, Schaaf, Tomlinson, Travers, "Diamond: A Multimedia Message System Built on a Distributed Architecture", IEEE Computer, Dec. 1985.

[TSU 86] Tsuruta, Mitsuhashi, Mera and Niwa, "Intelligent Communication Terminal for Integrating Voice, Data and Video Signals", ICC'86

[UED 87] T. Ueda, Sakata, Imai, "Multimedia Document Handling on a Distributed Interoffice Mail System", INFOCOM'87, pp. 1009.

[WAT 85] Watanabe., Marakami, Ishikawa and Kamae, "Audio and Visually Augmented Teleconferencing", Proceedings of the IEEE, vol. 73, nº 4, April 1985.

[XER 81] The Xerox Learning Research Group, "The Smalltalk-80 System", BYTE, vol. 6, n° 8, Aug. 1981, pp. 36-48.

Appendix I
Definition of Smalltalk Classes

Channel

A channel represents a connexion between a local entity (the owner of the channel) and its peer entity on a remote computer (given by the remote name). If the link is dropped due to an error, the channel is disconnected. An established channel is ready to receive incoming messages.

Inherits From: Object

Inherited By: CommandChannel DatagramChannel

Named Instance Variables:

bufferSize

errorString

inNcb
The networkControlBlock used for receiving.

outFlag

outNcb
The networkControlBlock used for sending.

outQueue

owner
The process which owns the channel.

ownerPointer

receiveBuffer
The buffer used for receiving.

remoteComputer
The name (string) of the remote computer of this channel.

state

Class Variables:

MaxBufferSize

Pool Dictionaries:

NetworkErrorDictionary

Class Methods:

connect: aLocalProcess
to: aProcessName
on: aRemoteName
This method establishes a connections with aProcess on a remote

computer (aRemoteName). It returns an instance of a channel once the connexion is established. aProcessName is the global name of an object in the Smalltalk dictionary which will be called in the remote Smalltalk system.

maximumBufferSize
Answers the internal Class variable to the maximum bufferSize supported by the network primitive.

maximumBufferSize: anInteger
Sets the internal Class variable to the maximum bufferSize supported by the network primitive.

Instance Methods:

basicSend: aBuffer
Sends aBuffer on the outChannel using the outNcb. It is a big buffer which is sent after all pending messages have been sent.

close
Removes the channel from the active channel list,
cancels the waiting Ncb,
hangs up the session and destroys the Channel.

closeNoErrorCheck
Removes the channel from the active channel list,
cancels the waiting Ncb is any, hangs up the session
and destroys the Channel.

getErrorString
Answer errorString with last reported error.

initialize

initializeOwner: aProcess
remoteName: aRemoteName
withNcb: aNcb
Initializes the local variables of the receiver

inNcb
Answer the inNcb.

isEvent
Answer true if the inNcb command has completed and false otherwise.

isOpen

outNcb
Answer the outNcb.

ownerProcess
Answers the owner Process of the channel.

postInNcb

processError
React to a network error.

```
processInEvent
     Process the incoming Event, issues a disconnectIndication to
     the ownerProcess if any error occures or a dataIndication.

processOutEvent
     Process a send event.

receive: aBuffer
     Sends aBuffer on the outChannel using the outNcb.

remoteComputer

send: aBuffer

sendDivide: aBuffer

sendPoll: aBuffer

sendWait: aBuffer
```

DatagramChannel

Datagram channels are used to send and receive data using the datagram service on the network.

Inherits From: Channel Object

Inherited By: (None)

Named Instance Variables:

bufferSize
(From class Channel)

errorString
(From class Channel)

groupName

inNcb
The networkControlBlock used for receiving.

nameNumber

outFlag
(From class Channel)

outNcb
The networkControlBlock used for sending.

outQueue
(From class Channel)

owner
The process which owns the channel.

ownerPointer
(From class Channel)

receiveBuffer
The buffer used for receiving.

remoteComputer
The name (string) of the remote computer of this channel.

state
(From class Channel)

Class Variables:

MaxBufferSize
(From class Channel)

Pool Dictionaries: (None)

Class Methods:

openFor: aLocalProcess
groupName: aString
Prepares aChannel for receiving broadcast datagrams.

Instance Methods:

closeNoErrorCheck
Removes the channel from the active channel listand closes it.

initialize

openFor: aProcess
groupName: aString
Prepares aChannel for receiving broadcast datagrams.

postInNcb
When we open a window...

processInEvent
Process the incoming Event, issues a dataIndication.

processOutEvent
Process a send event.

send: aBuffer
Sends aBuffer as Broadcast datagram.

sendPoll: aBuffer

NetworkControlBlock

The primitive calls to interact with the netbics interface
are implemented in this class.

Inherits From: ByteArray FixedSizeCollection IndexedCollection
Collection Object

Inherited By: (None)

Named Instance Variables: (None)

Class Variables:

Ncb

PrimitiveErrors

Pool Dictionaries:

NetworkErrorDictionary

Class Methods:

error: anInt

initializeClass
Initialize class variable Ncb as a NetworkControlBlock

ncb

new
Creates and initializes a new instance of the class.

pollNet
Polls network and process events.

resetPrimitive
Resets the static variables and index of the primitive.

Instance Methods:

addGroupName: aString
Adds aString to the local name table on the adapter as group name.
Answer the name number of the added name.

addName: aString
Adds aString to the local name table on the adapter

addressAt: anIndex
Answers the receiver's address at index position in an array
first: offset, second: segment.

buffer
Answers the receiver's buffer address field

```
callName
     Answers the receiver's  callName  field as a string

callName: aString
     Answers the receiver with aString as CallName  field

callName: aString myName: aString2
     Calls a remote name aString using as local name aString2.
     Receive and send time outs are local constants set during
     initialization. The local Ncb will hold the local session
     number if the command completes correctly.

callNoWaitName: aString myName: aString2
     Calls a remote name aString using as local name aString2.
     Receive and send time outs are local constants set during
     initialization. The local Ncb will hold the local session
     number if the command completes correctly.

cancel
     Cancels the receiver.

checkCompleted
     Answers true if the command has completed.

cmdCplt
     Returns the cmdCplt field. This is not meaningfull since the network
     interface updates a copy of the receiver, not the receiver itSelf.

command
     Answers the receiver's command field

command: aByte
      answers the receiver with aByte as  command field

deleteName: aString
     Deletes aString from the local name table on the adapter

entryId
     Returns the reference used by the primitive to find the copy of
     the receiver.

execute
     Executes the network command coded in the receiving
     NetworkControlBlock and returns the immediate   return code.
     The receiving NetworkControlBlock contains the updated information.
     It returns the immediate return code.

execute: anInteger with: aBuffer
     Executes the command coded in the receiving networkControlBlock using
     the Buffer (byteArray) aBuffer and the function code anInteger.
     This call returns the immediate  return code as smallInteger or
     a false or true pointer, or the pointer of a completed Ncb.

executeWith: aBuffer
     Executes the network command coded in the receiving
     NetworkControlBlock and returns the immediate   return code.
```

```
     The receiving NetworkControlBlock contains the updated information.
     It returns the immediate return code only if it was a wait command.

getResult
     Updates the receiver  with the information of a previous
     completed command.

getResult: aBuffer
     Updates the receiver and aBuffer with the information of a previous
     completed command.

getStringFrom: start to: stop
     Answers the receiver's    field as a string

hangUp
     This command closes the session estabblished with another
     name on the network.

ifError: aBlock
     Executes aBlock if command has errored.

initialize
     Initialize the instance of NetworkControlBlock.

isError
     Answers true if command has errored otherwise false.

isPending
     Answer true if the receiver if still a pending command.
     This is indicated by the last two bytes of the receiver
     ( entryId ). If the entryId is not 65535, the command is
     still pending.

lanaNum
     Answers the receiver's lanaNum field

length
     Answers the receiver's length field

listenName: aString myName: aString2
     listens for a remote name aString using as local name aString2.
     Send time outs are local constants set during
     initialization.
     A name '*' means listen for any remote name. The local Ncb will
     hold the local session number and the calling name if the command
     completes correctly.

listenNoWaitName: aString myName: aString2
     Listens for a remote name aString using as local name aString2.
     Send time outs are local constants set during initialization.
     A name '*' means listen for any remote name. The local Ncb will
     hold the local session number and the calling name if the command
     completes correctly.
     This command will return immediatly and must be checked for
     completion.

localSessionNumber
```

```
     Answers the receiver's lsn field

localSessionNumber: aByte
      answers the receiver with aByte as lsn field

name
     Answers the receiver's  callName  field as a string

name: aString
     Answers the receiver with aString as Name  field

netPrimitive: anInteger with: aBuffer
     Executes the network command coded in the receiving
     networkControlBlock using
     the Buffer (byteArray) aBuffer and the function code  anInteger.
     This call returns the immediate  return code as smallInteger.
     Function 0 resets the primitive buffer and ncb stack.
     Function 1  executes the receiver with aBuffer; if it was a wait command,
     both are updated on return; if not, a copy of the receiver and the buffer
     is executed and the receiver will only contain a reference
     number in it's last 2 bytes. The reference is used in function 2 and 3 to
     retrieve the information.
     Function 2  returns true if the ncb has completed else false.
     Function 3  updates the receiver and the buffer. It must only be called
     when a a previous nowait command was issued and completed.
     Function 4  cancels the receiver.
     Function 5  polls all pending ncb's and returns the ncb which has
     completed if event, false if none.
     Function 6  returns the errorCode on failure.

num
     Answers the receiver's num field

num: anInteger
     Updates the receiver's num field

post
     Answers the receiver's post address field

printOn: aStream
     Prints the NCB on a stream with beautiful format

putString: aString at: start
     Writes a String at index start, padded with 0 or truncated to be of
     length 16.

receive: aBuffer
     Receives aBuffer from a remote name. A session must have been
     established before issuing this command. ncb length is the
     real length of the received message (which is -hopefully- smaller
     then the length of aBuffer.

receiveAny: aBuffer
     Receives aBuffer from any remote name with whom a session has
     been established using the current local name.
     ncb localSessionNumber is updated to contain the session
     corresponding to the received message.
```

ncb length is the real length of the received message
(which is -hopefully- smaller then the length of aBuffer).

receiveNoWait: aBuffer
Receives aBuffer from a remote name. A session must have been established before issuing this command. ncb length is the real length of the received message (which is -hopefully- smaller then the length of aBuffer).
This command will return immediatly and must be checked for completion.

receiveNoWaitAny: aBuffer
Receives aBuffer from any remote name with whom a session has been established using the current local name.
ncb localSessionNumber is updated to contain the session corresponding to the received message.
ncb length is the real length of the received message
(which is -hopefully- smaller then the length of aBuffer).
This command will return immediatly and must be checked for completion.

receiveNoWaitBroadcastDatagram: aBuffer
Receives aBuffer from a remote name. ncb length is the real length of the received message (which is -hopefully- smaller then the length of aBuffer).
This command will return immediatly and must be checked for completion.

receiveNoWaitDatagram: aBuffer
Receives aBuffer from a remote name. ncb length is the real length of the received message (which is -hopefully- smaller then the length of aBuffer).
This command will return immediatly and must be checked for completion.

receiveTimeOut
Answers the receiver's RTO field

receiveTimeOut: aByte
Answers the receiver with aByte as RTO field

reserve
Answers the receiver's first byte of reserve field

resetSession: anInteger numberOfNcb: anotherOne
Resets local adapter

retCode
Answers the receiver's retCode field

send: aBuffer
Sends aBuffer to the remote name. A session must have been established before issuing this command.

sendBroadcastDatagram: aBuffer
Sends aBuffer to any station ready to receive. The receiver must already have been intialized (must contain the name number).

```
sendDatagram: aBuffer
     Sends aBuffer as datagram using the receiver name number field
     and the receiver callName.

sendNoWait: aBuffer
     Sends aBuffer to the remote name. A session must have been
     established before issuing this command.

sendTimeOut
     Answers the receiver's  STO  field

sendTimeOut: aByte
     Answers the receiver with aByte as STO  field

showError
     Handles errors occurring during execution of network commands.
     Returns if the returnCode is 0 otherwise it generates an error
     message.
```

NetworkManager

This class has only one intance: TheNetworkManager. It keeps track of the current opened channels and mediates the asynchronous network events to the relevant channel.

Inherits From: WorkstationAgent Object

Inherited By: (None)

Named Instance Variables: (None)

Class Variables:

Connected

HisName

MyName
The name (aString) of the local computer.

Ncb1
The network control block used for general purpose operations and to listen to calls from remote computers.

NcbDict
The dictionary which associates the ncb to the owner channels.

Recursion

Pool Dictionaries:

NetworkErrorDictionary

Class Methods:

cleanUpGarbage
This method should be called in case any error happens during test.

connected

connected: aBoolean
Changes the value of Connected.

giveName: aName hisName: any
Initializes or changes the values of the class variables MyName.

hisName

myName
Returns the network name of this workstation.

myName: me

Initializes or changes the values of the class variables MyName, the network name of this workstation.

new
Creates a new instance with all variables initialized and starts the network.

restart

Instance Methods:

addChannel: aChannel
Adds a channel to the local dictionary.

closeAllChannels

createNewChannel: aNcb
Creates a new channel and calls the object to which the connection request is made.

initialize

myName
Answers the name of this computer: MyName.

notPresent
Display a message saying that the network driver is not present.

processEvent: aNcb
Dispatches the network event to the relevant channel on which the event has occured.

processNetEvent
Processes an event: a connection request or an error.

removeChannel: aChannel
Remove the inNcb of the channel from the dictionary.

ExternalTerminalStream

This terminalStream subclass multiplexes the input from the remote terminal and the local terminal.
    If there is local input, it is read;
        if the cursor is in the private area, the result is returned.
        if not, the result is passed to the conference agent and a clock event is returned (no action).
    If not, the queue of external event is cheched and the first element (if any) is returned.

Inherits From: PierTerminalStream TerminalStream ReadWriteStream WriteStream Stream Object

Inherited By: (None)

Named Instance Variables:

collection
(From class Stream)

inBuffer

inQueue

inStream

lastEventType

mouseOffset
(From class TerminalStream)

mouseTime
(From class TerminalStream)

oldPosition

position
(From class Stream)

privateEvent

readLimit
(From class Stream)

sendStream

state
(From class TerminalStream)

toggle

writeLimit
(From class WriteStream)

x

y

Class Variables: (None)

Pool Dictionaries:

FunctionKeys
CharacterConstants

Class Methods:

new

Instance Methods:

addAll: aStream

initialize

isThereInput

read
Perform the current state of the input mechanism and send the buffer.

readFromQueue
The input is read from the external event Queue.

readLocal
Here we read a local input.

readPrimitive
Private - Answer the next character from the inQueue.

restoreOldPosition

setLoc
Private - Answer SetLoc. Displays the mouse at new offset

writeToStream: a
The input from the local terminal is coded on a character stream and pushed at the end of the external queue so that it has no priviledge over events sent by a remote computer.

ConfAgent

There is only one instance of this class: TheCA. It is the part of the User Agent software which handles the conference requests from the user and the conference manager.

Inherits From: Conf Object

Inherited By: (None)

Named Instance Variables:

confCommand

dispatcher

floorCommand

inStream

msgPane

state

stream

topPane

Class Variables:

ConfCreateMenu

ConfIndicationMenu

ConfWaitPrompter

Pool Dictionaries:

CharacterConstants

Class Methods:

new

Instance Methods:

activateWindow

becomeAutonomus

becomeFloorHolder

becomeInConference

becomeNotFloorHolder

becomeWaiting

changedAll

closeCommand

confCreateMenu
    Answer the menu for the conference initiation.

confIndicationMenu: aString
    Answer the menu for an incoming conference request.

confInfo

confWaitPrompter
    Answer the prompter for the conference waiting state.

dispatcher
    Answer the dispatcher.

do

doCall

doCloseConference
    Close the conference.

doCommand: aString

doLocal

doNothing

doNothing: aParameter

doRemote

floorCommand

hasCursor

initialize

mCurs: aStream
    New shared cursor position.

mEven: aStream
    Mouse informations.

open
    Create a window at the top of the display screen for the conference controls.

openConference

pAbortInd: aString

pCloseConf: aString

pCloseInd: aString

pDataInd: aStream

pGiveFloorInd

pleaseFloorInd

pOpenConf: aString

pOpenInd: aOriginatorName

processKey: aKey

protError: aString

pStartConf: aString

pStartInd

putEvent: aStream

putMove: aStream

putMsg: aString

redrawScreen
    Redraw the screen to reserve space for the conference window.

sendCursor: aString
    Sending mouse information as a string.

setPrivateRectangle

state

stateArray

systemError
    React to an error.

topPane

unsetPrivateRectangle

ConfManager

There is only one instance of this class: TheCM, the Conference Management Agent which interacts with the remote Conference Management Agent through messages ent on.the network.

Inherits From: Conf Object

Inherited By: (None)

Named Instance Variables:

channel

datagramChannel

inStream

originatorName

speakingChannel

state

Class Variables: (None)

Pool Dictionaries: (None)

Class Methods:

new

Instance Methods:

closeChannel: aChannel
Closes a channel after checking it is really a Channel.

connectIndication: aChannel
This is the message received when a remote computer is calling.

datagramIndication: aBuffer
aBuffer is received from the network on a datagram channel.

datagramOpen

dataIndication: aBuffer from: aChannel
aBuffer is received from the network on a channel.

disconnectAll

disconnectIndication: aChannel
aChannel has been disconnected.

initialize

```
mClos: aStream

mData: aStream

message: aString

mGFlo: aStream

mOAck: aStream

mOpen: aStream

mPFlo: aStream

pAbortReq: reason

pCloseReq: reason

pCloseResp: aString

pGiveFloorReq

pleaseFloorReq

pOpenReq: aName

pOpenResp: aString

postInNcb

pRDataReq: aString

protError: aString

pStartReq

pStartResp

pUDataReq: aString

remoteComputer

send: aBuffer

send: aBuffer  onChannel: aChannel

sendDatagram: aBuffer
```

Appendix II
Listings of Primitive Programs

Source code of the primitive program

```
#define VER "Primitive 94 v3.00"
/*************************************************************************/
/*****  Oper:Executes a NCB send by smalltalk, using the buffer   ***/
/*****       provided by Smalltalk. Returns the result code       ***/
/*****       (as SmallInteger) or a pointer to the ncb which has  ***/
/*****       completed.                                           ***/
/*****       -Returns a Smalltalk object pointer.                 ***/
/*****       -The polling procedure returns false when no event   ***/
/*****       is detected from the network.                        ***/
/*****  Note:Must be compiled in L / D model to use far pointers  ***/
/*****       and linked to pri94.asm v2.0                         ***/
/*************************************************************************/
#include "\lc\hdrs\stdio.h"
#include "\lc\hdrs\stdlib.h"
#include "\lc\hdrs\dos.h"
#include "\lc\hdrs\string.h"
#include "gen.h"
#define MAX 10
             /* The maximum number of static ncbs and buffers */
#define BUFFSIZE 512
             /* The buffer size, Smalltalk buffers must be smaller */
#define NIL -1
#define YES  -2
#define NO -1
#define SMALL_NIL 7   /* The Smalltalk object pointer for nil */
#define SMALL_ZERO 0  /* The Smalltalk object pointer for 0 */
#define SMALL_TRUE 33 /* The Smalltalk object pointer for TRUE*/
#define SMALL_FALSE 1 /* The Smalltalk object pointer for false*/

struct NCBE {
     char command;
     char retcode;
     char lsn;
     char num;
     char *buffer;
     unsigned int length;
     char callname[16];
     char name[16];
     char rto;
     char sto;
     char *post;
     char lana_num;
     char cmd_cplt;
     char reserve[14];
     int entryId;
          /* Not shown here are the 2 last bytes: the owner of the ncb
          used by Smalltalk */
     };

struct ENTRY_STR     /* All information relative to a pending command */
     {
     int freeEntry;   /* FreeEntry ==  FREE means this entry is free */
```

```
    int object;      /* The Smalltalk object corresponding to the static
ncb*/
    char buffer[BUFFSIZE]; /* The buffer for this Ncb */
    struct NCBE ncb; /* The static ncb corresponding to the Smalltalk
object */
    };

struct ENTRY_STR entryT[MAX];
struct NCBE cancelNcbe; /* used for cancelling previous issued commands */
int errorCode;          /* integer which is updated when an error occurs
                           and can be returns by calling function 6 */
int lastPolledEntry = -1;
extern  unsigned long _psize,_dsize;
extern  unsigned int _tsize;
int i,j;                /* global loop variables */
unsigned int ASM_B_SIZE;

/*************************************************************************/
/*  execute(pNcb)                                                        */
/*  Action          :  executes netbios command                          */
/*  Global variables:  pExternalNcb: pointer to Ncb                      */
/*  Global function :  executeNcb() defined in the assembler program*/
/*  Returns          :   immediate return code                           */
/*************************************************************************/
char *pExternalNcb;
extern char executeNcb();

int execute(pNcb)
struct NCBE *pNcb;
    {
    pNcb->post = 0;
    pExternalNcb = (char *)pNcb;
    return((int) executeNcb());
    }
/*************************************************************************/


/*************************************************************************/
/*  main()                                                               */
/*  Action          :   initialize and install the primitive             */
/*************************************************************************/
main()
    {
    int i;

    reset();
    printf("\n pgm+data :%ld",_psize+_dsize);
    printf("\n pgm: %ld,data: %ld,total: %d",_psize,_dsize,_tsize*16);
    printf("\n pgm+data :%d",primSize());
    printf("\n %s installed ( %d buffers of %d bytes)",VER,MAX,BUFFSIZE);
    installBis();
    }/* end of main() */
/*************************************************************************/
```

```
/*************************************************************************/
/*  min(a,b)                                                             */
/*  Action          : returns the smallest integer of a and b            */
/*************************************************************************/
unsigned min(a,b)
unsigned int a,b;
     {
     if (a > b) return(b);
     return(a);
     }
/*************************************************************************/

/*************************************************************************/
/*  cancel(pncbe)                                                        */
/*  Action          : cancel the networkControlBlock to which            */
/*                    pncbe is pointing.                                 */
/*  Returns         : the immediate return code.                         */
/*************************************************************************/
cancel(pncbe)
struct NCBE *pncbe;
     {
     cancelNcbe.command = 0x35;
     cancelNcbe.buffer = (char *)pncbe;
               /* give address of ncb to cancel */
     pExternalNcb = (char *) &cancelNcbe;
     return((int)executeNcb());
               /* execute cancel command */
     }
/*************************************************************************/

/*************************************************************************/
/*  hangUp(pncbe)                                                        */
/*  Action          : hangs up the session corresponding to the          */
/*                    networkControlBlock *pncbe                         */
/*  Returns         : the immediate return code.                         */
/*************************************************************************/
hangUp(pncbe)
struct NCBE *pncbe;
     {
     pncbe->command = 0x12;
     pExternalNcb = (char *) pncbe;
     return((int)executeNcb());
               /* execute hangUp command */
     }
/*************************************************************************/

/*************************************************************************/
/*  reset()                                                              */
/*  Action          : resets the entry table. All entries are freed      */
/*************************************************************************/
reset()
     {
     int i;
     for (i=0;i< MAX; i++)
          {
          entryT[i].freeEntry = YES;
          entryT[i].object = 0;
```

```
        }
    }
/*************************************************************************/

/*************************************************************************/
/*  getEntry()                                                           */
/*  Action  : checks to see of there is a free entry in the static       */
/*            entry table. If true it assigns this entry to the          */
/*            current networkControlBlock provided by Smalltalk.         */
/*  Returns : the entry id (an integer) or NO if failed.                 */
/*************************************************************************/
int getEntry()
    {
    int i;

    for(i=0;i<MAX;i++)
        {
        if (entryT[i].freeEntry == YES )
            {
            entryT[i].freeEntry= NO;
            entryT[i].object = ASM_RECEIVER;
            return(i);
            }
        }
    return(NO); /* no more left */
    }
/*************************************************************************/

/*************************************************************************/
/*  putEntry(aEntry)                                                     */
/*  Action  : liberates the entry correspnding to the id aEntry.         */
/*  Returns : YES if ok or NO if failed.                                 */
/*************************************************************************/
int putEntry(aEntry)
int aEntry;
    {
    if ((aEntry < MAX) && (aEntry >= 0))
        {
        entryT[aEntry].freeEntry= YES;
        entryT[aEntry].object = 0;
        return(YES);
        }
    return(NO);/*invalid entry*/
    }
/*************************************************************************/

/*************************************************************************/
/*  isEntry(aEntry)                                                      */
/*  Action  : checks to see if agiven entry is available of in use.      */
/*  Returns : TRUE or FALSE or NO if failed.                             */
/*************************************************************************/
int isFree(aEntry)
int aEntry;
    {
    if ((aEntry < MAX) && (aEntry >= 0))
        return(entryT[aEntry].freeEntry);
    return(NO);/*invalid entry */
```

```
    }
/*************************************************************************/

/*************************************************************************/
/*  pollNet()                                                            */
/*  Action  : checks all pending commands in the static entry table.*/
/*  Returns : the pointer to the completed networkCommandBlock or   */
/*            SMALL_FALSE if there was none.                             */
/*************************************************************************/
pollNet()
     {
     int j;

     for(j=0;j<MAX;j++)
          {
          lastPolledEntry++;
          lastPolledEntry =lastPolledEntry % MAX;
          if((isFree(lastPolledEntry) == NO) &&  (
(entryT[lastPolledEntry].ncb.cmd_cplt) != 0xFF))
                    return(entryT[lastPolledEntry].object);
                         /* returns the pointer to the completed ncb */
          }
     return(SMALL_FALSE); /* SMALL_FALSE means no event */
     }
/*************************************************************************/

/*************************************************************************/
/*  Global variables used in the following function and  the             */
/*  dispatcher function.                                                 */
/*************************************************************************/
int             currentEntry, returnCode;
unsigned int    inBufferSize,inNcbSize;
struct NCBE     *pInNcb,*pStaticNcb;
char            *pStaticBuffer;
/*************************************************************************/

/*************************************************************************/
/*  doReset()                                                            */
/*  Action    : cancel, hangup and reset all pending entries             */
/*  Returns   : the maximum buffer size (in Smalltalk format)            */
/*************************************************************************/
int doReset()
     {
     int i;
     for(i=0;i<MAX;i++)
               {
               if (isFree(i) == NO)
                    {
                    cancel( &entryT[i].ncb);
                    hangUp( &entryT[i].ncb);
                    putEntry(i);
                    }
               }
     return(BUFFSIZE * 2);
     }
/*************************************************************************/
```

```
/*****************************************************************/
/*  doPostCommand(pInBuffer,pReceiver)                           */
/*  Action: perform the command coded in the smalltalk structure */
/*          *pReceiver with the buffer *pInBuffer.               */
/*          If a wait command, execute the given command and buffer */
/*          and return the result. If a no-wait command, copy the */
/*          ncb and the buffer to static structure, execute the  */
/*          copy and return.                                     */
/*  Returns:SMALL_NIL in case of failure (buffer too big or no   */
/*          entry available), or the return code.                */
/*****************************************************************/
int doPostCommand(pInBuffer,pReceiver)
char *pReceiver;
char *pInBuffer;
    {
    if ((pInNcb->command & 0x80) == 0x80)
         {         /* this is a nowait command */
         if ( inBufferSize > BUFFSIZE)
              {
              errorCode = 1;
              return(SMALL_NIL);
              }    /* fail if requested buffer is bigger then static */
         if ((currentEntry =getEntry()) == NO)
              {
              errorCode = 2;
              return(SMALL_NIL);
              }    /* get a new Entry, fail if no more available */
         pStaticNcb = &entryT[ currentEntry].ncb;
              /* assign the staticNcb pointer */
         *pStaticNcb = *pInNcb;
              /* copy ncb to static ncb */
         pStaticNcb->length = inBufferSize;
         pStaticNcb->buffer = entryT[ currentEntry].buffer;
         if (inBufferSize > 0)
                   memcpy( pStaticNcb->buffer, pInBuffer, inBufferSize);
              /* copy inBuffer */
         pInNcb->entryId=currentEntry;
              /* execute copy with copy buffer */
         if ((returnCode =  execute(pStaticNcb)) != 0)
              {/* command has errored: copy back ncb and pushcurrentEntry
                   does not copy buffer */
              *pInNcb = *pStaticNcb;
              pInNcb->entryId = NIL;
              if (putEntry(currentEntry) == NO)
                   {
                   errorCode = 4;
                   return(SMALL_NIL);
                   }
              }
         }
    else
         { /* this is a wait command */
         pInNcb->buffer= pInBuffer;
         pInNcb->length= inBufferSize;
              /* execute original with original buffer */
         returnCode = execute(pInNcb) ;
         }
```

```
    return(returnCode * 2);
    }
/*************************************************************************/

/*************************************************************************/
/*  doCheckCOmpleted(entryId)                                            */
/*  Action: check to see if the command in the entry table has           */
/*          completed.                                                   */
/*  Returns:SMALL_NIL in case of failure or SMALL_TRUE or SMALL_         */
/*          FALSE.                                                       */
/*************************************************************************/
int doCheckCompleted(entryId)
int entryId;
    {
    if (entryId == NIL)
        {
        errorCode = 3;
        return(SMALL_NIL); /* This is not a valid ncb */
        }
    if  (entryT[ entryId ].ncb.cmd_cplt == 0xFF)
        return(SMALL_FALSE);
    else
        return(SMALL_TRUE);
    }
/*************************************************************************/

/*************************************************************************/
/*  doGetResult(entryId,pInBuffer)                                       */
/*  Action: copy static ncb and buffer to the one's given by             */
/*          Smalltalk.                                                   */
/*  Returns:SMALL_NIL in case of failure or SMALL_ZERO                   */
/*************************************************************************/
int doGetResult(entryId,pInBuffer)
int entryId;
char *pInBuffer;
    {
    if ( inBufferSize > BUFFSIZE)
        {
        errorCode = 1;
        return(SMALL_NIL);
            /* fail if requested buffer is bigger then static */
        }
    if (putEntry(entryId) == NO)
        {
        errorCode = 5;
        return(SMALL_NIL);
        /* This is not a valid ncb */
        }
    pStaticNcb = &entryT[entryId].ncb;
    if ( pStaticNcb->command == 0x95)
        {
        pInNcb->retcode = pStaticNcb->retcode;
        pInNcb->length = pStaticNcb->length;
        /* copy only needed info */
        }
    else
        *pInNcb = *pStaticNcb;
```

```
          /* copy whole ncb */
     pInNcb->entryId = NIL;
     if (inBufferSize > 0)
          memcpy( pInBuffer, entryT[ entryId].buffer,
min(inBufferSize,pStaticNcb->length));
     return(SMALL_ZERO);
     }
/*************************************************************************/

/*************************************************************************/
/*  doCancelGet(entryId)                                                 */
/*  Action: cancel pending command and get result without buffer         */
/*  Returns:SMALL_NIL in case of failure or return code                  */
/*************************************************************************/
int doCancelGet(entryId)
int entryId;
     {
     if (putEntry(entryId) == NO)
          {
          errorCode = 5;
          return(SMALL_NIL);
          /* This is not a valid ncb */
          }
     returnCode = cancel( &entryT[entryId].ncb);
               /* execute cancel command */
     *pInNcb = entryT[entryId].ncb;
               /* copy result ncb */
     pInNcb->entryId = NIL;
     return(2 * returnCode);
     }
/*************************************************************************/

/*************************************************************************/
/*  dispatcher(functionIndex,pInBuffer,pReceiver)                        */
/*  Action : executes the function given by functionIndex with           */
/*           the parameters pInBuffer and pReceiver.                     */
/*  Input parameters: pInBuffer: the pointer to a Smalltalk buffer       */
/*                    pReceiver: the pointer to a Smalltalk SuperNcb     */
/*                    functionIndex: an Integer function code            */
/*************************************************************************/
int dispatcher(functionIndex,pInBuffer,pReceiver)
int  functionIndex;
char *pReceiver;
char *pInBuffer;
     {
     inBufferSize= ASM_B_SIZE - 2;
     pReceiver+=2;
     pInBuffer+=2;
     pInNcb=(struct NCBE *) pReceiver;

     switch(functionIndex)
          {
          case 0:
               return(doReset());
          case 1:
               return(doPostCommand(pInBuffer,pReceiver));
          case 2:
```

```
                return(doCheckCompleted(pInNcb->entryId));
            case 3:
                currentEntry = pInNcb->entryId;
                return(doGetResult(currentEntry,pInBuffer));
            case 4:
                currentEntry = pInNcb->entryId;
                return(doCancelGet(currentEntry));
            case 5:
                return( pollNet());
            case 6:
                return(2 * errorCode);
            default:
                errorCode = 6;
                return(SMALL_NIL);
            }
        }
/*****************************************************************************/
```

Assembler part of the primitive program

```
;*************************************************************************;
;***  Primitive 94, v2.0  assembler part:                                ***;
;***  Translates calls from smalltalk to C and reverse.                  ***;
;***  Input: 1)the receiver (a SuperNcb object)                          ***;
;***         2)the first argument (a SmallInteger) function code         ***;
;***         3)the second argument (a String object) a buffer            ***;
;***  Output:see the c program.                                          ***;
;***  Action:verifies that the objects are not swapped out               ***;
;***    and translates them to far pointer. Then calls C function to     ***;
;***    do the processing.                                               ***;
;***  Note:  link with pri4.c v3.0                                       ***;
;*************************************************************************;

          INCLUDE \smaltalk\fixdptrs.usr
          INCLUDE \lc\d\DOS.MAC
IF1
          INCLUDE \smaltalk\access.usr
ENDIF

          PUBLIC PRIM94,SMA_CS

IF        LPROG
          EXTRN dispatcher:FAR
ENDIF
IF        LPROG EQ 0
          EXTRN dispatcher:NEAR
ENDIF

          PEXTRN pExternalNcb;pointer to Ncb
          EXTRN  ASM_RECEIVER:WORD;contains the Smalltalk object
                              ;= the receiving Ncb
          EXTRN  ASM_B_SIZE:WORD

          DSEG
          SMA_CS DW ?
          ENDDS

          PSEG

prim94    PROC FAR
;This is code that will be executed everytime primitive
;is invoked from "Smalltalk/V"
          enterPrimitive     ;enter primitive macro
          MOV  BX,SEG DGROUP ;used for lattice interface
          MOV  DS,BX
          MOV  BX,[BP+12]
          MOV  SMA_CS,BX

          getClass DI,DX,ES  ;DI contains the receiver (a NCB)
          CMP  DX,ClassSwappedOut
          JNE  nofailure
          exitWithFailure 2  ;failure macro
```

```
nofailure:MOV ASM_RECEIVER,DI    ;ASM_RECEIVER to be used by the c pgm.

          getObjectAddress DI,BX,ES;do not use ES in s and p model
          PUSH    ES
          PUSH BX
          MOV  DI,[BP+16]     ;this is the last argument: abuffer
          getClass DI,DX,ES   ;DX is the Classpointer register.
          CMP  DX,ClassSwappedOut
          JE   failure2
          MOV  BX,DI
          getObjectAddress BX,DI,ES;do not use ES in s and p model
          PUSH ES         ;give address to primitive
          PUSH DI         ;first 2 bytes is an integer
                               ;wich equals total object size in byte
          MOV  AX,ES:[DI]
          isSIzeEven BX,CX,ES
          JZ   even
          DEC  AX
even:     MOV  ASM_B_SIZE,AX
          MOV  AX,[BP+18]
          SHR  AX,1       ;this is the first arg: smallint
          JC   failure4   ;the value was not a small int
          PUSH AX

          CALL dispatcher
          POP  BX         ;flush args
          POP  BX
          POP  BX         ;flush args
          POP  BX
          POP  BX
          CMP  AX,nilPtr ;nilPtr = 7 means an error occured
          JE   failure         ;primitive fails

          MOV  BX,AX           ;result must be aSmalltalk
                               ;object.

success:  exitWithSuccess 2  ;and enter into successful exit
failure4:POP  AX
          POP  AX
failure2:POP  AX
          POP  AX
failure:  exitWithFailure 2  ;failure macro
prim94    ENDP

BEGIN     executeNcb
          PUSH      BP
          MOV  BP,SP
          LES  BX,pExternalNcb     ;load ES:BX with ncb pointer
          INT  5CH                 ;issue software interrupt, retcode in AL
          POP  BP
          RET
executeNcb ENDP

          ENDPS
          END
```