A Presentation Agent for Distributed Multimedia Information Systems

by
J.A. Rody

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 of Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario

1996

ISBN 0-612-15757-1

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Table of Contents

List of Figures

List of Tables

Abstract

Distributed multimedia information systems aim to improve the way people exchange information by supporting a wider variety of media types. These media types include the traditional static media like text, graphics and images as well as the continuous media types like audio and video. Integrating these different types of media within computer based communication systems is no easy task. Indeed, the successful realization of multimedia information systems will depend on the ability of these systems to overcome the problems associated with information integration. There are two types of integration involved in multimedia information systems: spatial integration and temporal integration.

This thesis addresses the problem of multimedia information integration; in particular temporal integration. Temporal integration deals with the inter-media and intra-media synchronization of multimedia information. In distributed multimedia information systems, synchronization is compounded by the presence of a computer communication network. Because most computer communication networks are asynchronous communication networks, these networks are not well suited for ensuring constant data delivery rates for continous media.

A remote multimedia presentation agent is designed to handle the synchronization of multimedia information from distributed sources. This remote multimedia presentation agent uses a multimedia document as the information interchange vehicle of the distributed multimedia information system. The remote multimedia presentation agent uses the progressive document retrieval strategy and dynamic quality of service management to help overcome the problems involved in the synchronized retrieval of multimedia information.

Acknowledgements

I would like to acknowledge the guidance which I have received from my thesis supervisor Dr. Ahmed Karmouch. Dr. Karmouch, through the MEDIABASE project, has offered me a unique learning experience for which I am very grateful.

I would like to thank Professor Glenn Warnock for his technical support throughout the implementation stage of my thesis. Professor Warnock was a continuous source of information which has allowed me to refine my implementation.

I am especially grateful to my wife, Pamela, for her support and encouragement, without which none of this would have been possible. Thank you for being there, always.

Thank you to Mom and Dad.

Acronyms

AAL	ATM Adaptation Layer
ANSI	American National Institute
API	Application Program Interface
ASCII	American National Standard Code for Information Interchange
ATM	Asynchronous Tranfer Mode
BER	Bit Error Rate
B-ISDN	Broadband Integrated Services Digital Network
CCITT	Consultative Committee for International Telegraphy and Telephony
CGM	Computer Graphics Metafile
CSCW	Computer Supported Cooperative Work
DMIS	Distributed Multimedia Information System
DQoS	Dynamic Quality of Service
DTD	Document Type Definition
FDDI	Fiber Distributed Data Interface
GIF	Graphics Information File
GUI	Graphical User Interface
IP	Internet Protocol
IPC	Inter-Process Communication
ISDN	Integrated Services Digital Network
ISE	Independent Synchronized Entities
ISO	International Standards Organization
JPEG	Joint Photographic Expert Group
LAN	Local Area Network
MIS	Multimedia Information System
MPA	Multimedia Presentation Agent
MHEG	Multimedia and Hypermedia information coding Experts Group
MPEG	Moving Pictures Expert Group
NFS	Network File System
NTSC	National Television Systems Committee
ODA	Office Document Architecture
OO	Object-Oriented
OSI	Open Systems Interconnection
PDR	Progressive Document Retrieval
PER	Packet Error Rate
POSIX	Portable Operating System for UNIX
RTR	Ready-to-Run
QoS	Quality of Service
SGML	Standard Generalized Markup Language
TCP	Transmission Control Protocol
TIFF	Tagged Information File Format
UDP	User Datagram Protocol
WAN	Wide Area Network
XNS	Xerox Network Systems

Chapter 1
Introduction

1.1 Motivation

The trend in today's information industry is to adapt communication services to the growing demand for multimedia communications. Multimedia communications aim to improve the way people communicate by supporting data streams of various media types within a uniform multimedia information system. The media types which must be supported by the multimedia information system include the discrete data types, like text, image and graphics, as well as the continuous data types, like audio and video. Because each data type

has very different manipulation and presentation requirements, the multimedia information system must provide support for media integration.

Media integration combines the various media so that they can be manipulated and presented coherently by the multimedia information system. Multimedia information systems rely on two types of media integration. The first, spatial integration, groups the media according to their spatial relationships, whereas the second type of integration, temporal integration, deals with the timing relationships needed to communicate the media.

Communication systems are typically characterized by a large number of different communicating entities within a widely distributed environment. The distributed multimedia information system is the generalized form of multimedia information system which takes into account the distributed nature of communication systems. In distributed multimedia information systems, we distinguish three levels of media integration: the source level, the transport level and the presentation level. All three of these levels work in conjunction to provide a reliable multimedia information system which will deliver the media according to their spatial and temporal relationships.

Ottawa University's Multimedia Information Systems and Research Laboratory created the MEDIABASE project to study various aspects of multimedia information systems. The goal of MEDIABASE is to design an advanced high performance multimedia information and communications system with a particular focus on document architectures, database models, high-level communication and synchronization, and real-time physical storage of multimedia

data [KAR 93, MEG 94]. MEDIABASE is currently developing a distributed multimedia information system which uses a multimedia document as its multimedia information exchange vehicle. This multimedia document is based on the MEDIADOC architecture [EME 93].

One of the major challenges in designing distributed multimedia information systems lies in the attempt to perform media integration. This is due to the fact that distributed multimedia information systems typically deal with media types which may communicate very large amounts of data over a computer network and which may require real-time inter-media and intra-media synchronization. Multimedia documents provide a framework which can logically handle the integration of multimedia data. However, the physical integration must be handled by multimedia presentation agents which control the synchronized retrieval of multimedia data from source to sink over the communication network.

1.2 Thesis Objectives

The main objective of this thesis was to design and implement a multimedia presentation agent to handle the end-to-end synchronized playback of multimedia documents based on the MEDIADOC architecture. Because of the nature of MEDIADOC documents, the presentation agent had to satisfy certain constraints. Firstly, it had to be able to integrate documents with potentially complex hierarchies. Secondly, the presentation agent had to be designed to support interactive document playback and interactive media types like dialogues

and control buttons. Finally, the presentation agent had to be able to play individual media streams by using independent synchronized playback processes as opposed to interlacing the media within a single playback process. This final design requirement was imposed to maximize the re-usability of media data as well as simplify the implementation and possible extensions to presentation processes for composite media.

The presentation agent was also designed to operate within a distributed environment. Therefore, the presentation agent had to be robust in order to handle the retrieval of multimedia data over computer networks. Dynamic fall-back and fall-forward functionality had to be included in order to handle the continuously changing transport services available within the communication network.

1.3 Thesis Outline

Chapter 2 of this thesis will review Distributed Multimedia Information Systems (DMIS). In this chapter, client/server architectures and multimedia documents are presented as the means to overcome the problem of heterogeneity which is typical of all DMISs. As well, multimedia learning, multimedia news and groupware applications are explored as DMIS based applications.

Chapter 3 will develop the issues involved in multimedia synchronization at the source, transport and presentation levels. Two strategies designed ot alleviate the problems

associated with end-to-end multimedia synchronization are also presented. Chapter 4 presents the architectures of the multimedia presentation agent, transport manager and source server.

Chapter 5 presents the implementation of the multimedia presentation agent, transport manager, source server, and the MEDIADOC document. This implementation allows for the continuous transfer of multimedia data over a communication network and is capable of dynamically re-configuring itself to the changing communication environment. Chapter 6 sets out the conclusions and offers suggestions for further research.

1.4 Main Contributions

The main contribution of this research is the design and implementation of a multimedia presentation agent capable of synchronizing a multimedia document based on the MEDIADOC architecture. This presentation agent differs from other agents because it uses the scenario of the MEDIADOC document to identify and schedule retrieval processes for the document's independent synchronized entities. The presentation agent makes use of dynamic quality of service parameters to support variability in network transfers. Also, the presentation agent uses an event synchronization model and inter-process communication primitives to achieve lip-sync quality inter-media synchronization within a multi-threaded environment. The presentation agent has a client/server architecture which permits it to be independent of the network and graphical display hardware and software of the local workstation.

Other contributions of the work performed for the completion of this thesis include the creation of a flexible, object-oriented data model to implement MEDIADOC documents. A graphical user interface, called MediaPlayer was created as the client application for the multimedia presentation agent. A transport protocol for the synchronized transfer of multimedia objects over a computer communication network was also introduced. Finally, MediaEditor, an early prototype for the presentation agent was used to combine document creation and playback within a single graphical user interface client.

Chapter 2
Multimedia Information Systems

2.1 Introduction

Multimedia Information Systems (MIS) result from the merging of the computing, communications and broadcasting industries. This merging is made possible today due to the technical advances in high-speed broadband networks, computer desktop workstations and information storage and compression techniques. Multimedia information systems aim to improve the way people communicate by supporting communication for various types of media and thereby creating a more "natural" way to exchange information [KAR 93].

The media types which must be supported by the multimedia information system include the discrete media types such as text, image and graphics as well the continuous media types such as audio, video and animation. Table 2.1 lists these media types and compares their storage and presentation requirements. As can be seen from this table, the storage and presentation of the different media vary greatly with respect to one another. In order to establish reliable multimedia communications, the multimedia information system will have to satisfy the full gamut of services required by these diverse media.

Table 2.1: Multimedia Storage and Presentation Requirements

Media	Data Type	Storage	Presentation
Text	- ASCII - Formatted ASCII (i.e. Postcript)	- approx. 2 kB per page	- static visual display - 0 B/s
Image	- Bitmapped Graphics - colour: 1-24 bit - size: iconic-full screen	- 64 kB - 75 MB per image	- static visual display - 0 B/s
Audio	- coded (or non-coded) digitized stream of voice (8 kHz) or CD (44.1 kHz) quality sound	- voice: 6-44 kB per second of data - CD: 176 kB per second of data	- continuous aural playback - voice playback rate: 6-44 kB/s - CD playback rate: 176 kB/s
Animation	- synchronized stream of images	- 2.5 MB per second of 16 bit colour, 320x240 pixels per frame, 16 frames per second	- continuous visual playback -playback rate: 2.5 MB/s
Video	- TV analog or digital stream of images	- 27.7 MB per second of 24 bit colour, 640x480 pixels per frame, 30 frames per second	- continous visual playback - playback rate: 27.7 MB/s

* Note: data from [FUR 94].

In general, most communications occur between individuals or groups which may be widely separated in space and time [HOD 90]. Therefore to become the defacto communication standard of the future, multimedia information systems must be able to handle the most general case of information exchange. That is, multimedia information systems must be capable of communicating information which may be stored or created live between communicating entities within a distributed environment. This leads to the more generalized distributed multimedia information system (DMIS).

This chapter is an introduction to distributed multimedia information systems. The next section will discuss distributed systems in general and the distribution and integration problems that these systems entail. Section 2.3 will then present some applications which are built upon distributed multimedia information systems; particularly, multimedia learning, multimedia news and multimedia groupware.

2.2 Distributed Multimedia Information Systems

2.2.1 Overview

The distributed multimedia information system communicates multimedia data between entities which may be both geographically and temporally dispersed. Figure 2.1 demonstrates a distributed multimedia information system, composed of various user-sites and applications connected to different computer networks. There are three fundamental elements to DMISs in

general. These are the source, presentation and transport elements required to manipulate the multimedia information within the system.

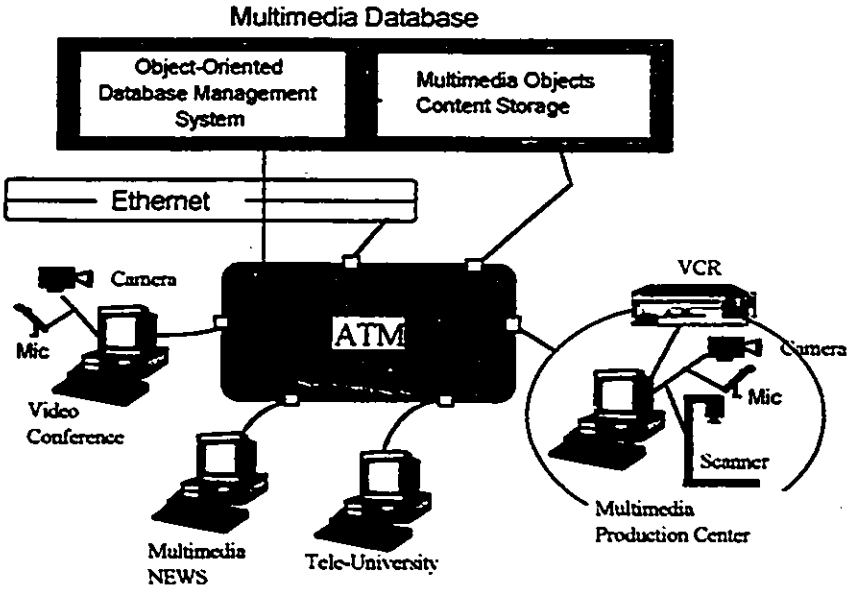


Figure 2.1 A Functional Architecture of a Distributed Multimedia Information System

The source element deals with the origins of the multimedia information. This includes the production and the storage of multimedia information. Media production can be handled by media editors like text and image editors or by using computer digital technology to scan images or digitize audio and video streams. In any case, media data can be produced for either immediate consumption and discarding, thereby creating live sources, or for future retrievals and therefore requiring some form of storage for the media data.

Storage deals with the persistence of the media information as well as information security, versioning, aging and volume management. Multimedia databases and file servers will be responsible for managing the media information which will be stored electronically in the form of files or documents. In general, electronic files and documents are physically stored on the

computer's hard disk. However, table 2.1 demonstrated that multimedia data can require enormous amounts of data. For instance a 30 minute audio/video clip will require approximately 50 GB of the comuter's hard disk. Since most computer systems have limited primary storage capacity and because of the sheer volume of data typically found in multimedia information systems, storage servers based on secondary memory systems like tape drives, optical disks or "juke boxes" will also become necessary for most applications. Sophisticated compression techniques like JPEG and MPEG, which are capable of reaching 15:1 and 200:1 compression ratios respectively, will also be used to reduce the storage burden of image and video data.

The presentation element deals with communicating the multimedia information at the user's local workstation. It is within the presentation element that the human/machine interaction occurs and therefore, the most important aspect of the local presentation is the user interface. For multimedia applications, graphical user-interfaces (GUIs) must be designed so that they provide powerful semantics for the coherent interaction between the user and the various types of media. This interaction must be independent of the origin of the media (local disk, remote live source, compressed images, tape drives,...) and must offer a natural if not obvious means to interface with the user. Hence, it is essential for the presentation element to offer graphical user interfaces which are serendipitous and where the media appear to be ubiquitous within the distributed multimedia information system.

The transport element deals with the communication of multimedia data throughout the DMIS. For multimedia data, robust, real-time transport protocols are of prime importance. These protocols must be flexible enough to support widely diverse transport requirements: from

relatively small, bursty transfers of text, image and graphics data; to large continuous transfers needed for audio and video media. These transport protocols must also be able to dynamically adjust to the changes of service quality caused by the non-deterministic nature of communication systems and by the diversity of the communication components which form a distributed multimedia information system.

Distributed multimedia information systems are characterized by one main factor: heterogeneity [LIT 90a]. There are two sources to heterogeneity in DMISs. The first is caused by the very nature of multimedia data, i.e the diversity of the media types; and the second source is due to the variety of computer and communication equipment linked together in typical distributed communication systems. Discussions on distributed communication systems and media integration are presented next to provide a solution to the problem of heterogeneity in DMISs.

2.2.2 Distributed Communication Systems

The most distinctive aspect of distributed communication systems is heterogeneity. Heterogeneity is due to the fact that large distributed systems are typically composed of many different types of networks, protocols and communications equipment. The successful realization of distributed communication systems is due primarily to the implementation of layered architectures and client/server modeling [WAL 91]. The client/server paradigm describes a communication model between the clients or service consumers, and the service providers or

servers. By allowing servers to become clients to other servers, complex services can be built by combining simpler services.

Layered architectures are the result of hierarchically grouping services according to their complexities and functionalities. Entities in one layer request services from entities in the layer which is immediately beneath them and provide services to the entities in the next higher layer of the hierarchy. The OSI reference model is a computer communication architectural model which is used to standardize communication protocols so that various computer systems may connect with one another. There are seven layers to OSI model; they are the physical, data link, network, transport, session, presentation and application layers. The lower layers control single link, node to node bit and packet transmission, while the next layers control the end to end packet transmission which provide the basic communication services to the upper application layers. Figure 2.2 shows how the ATM protocol maps to the lower layers of the OSI reference model.

The client/server and the OSI reference models provide a powerful architecture which simplifies the design, implementation and maintenance of large communication systems. The standard layer interfaces allow designers to plug components into their respective layers and to use the services offered by these components. Therefore the client/server model maximizes re-usability and inter- connectivity and facilitates the implementation of common services for the whole system independently of any of the constituent communicating entities which make up the communication system.

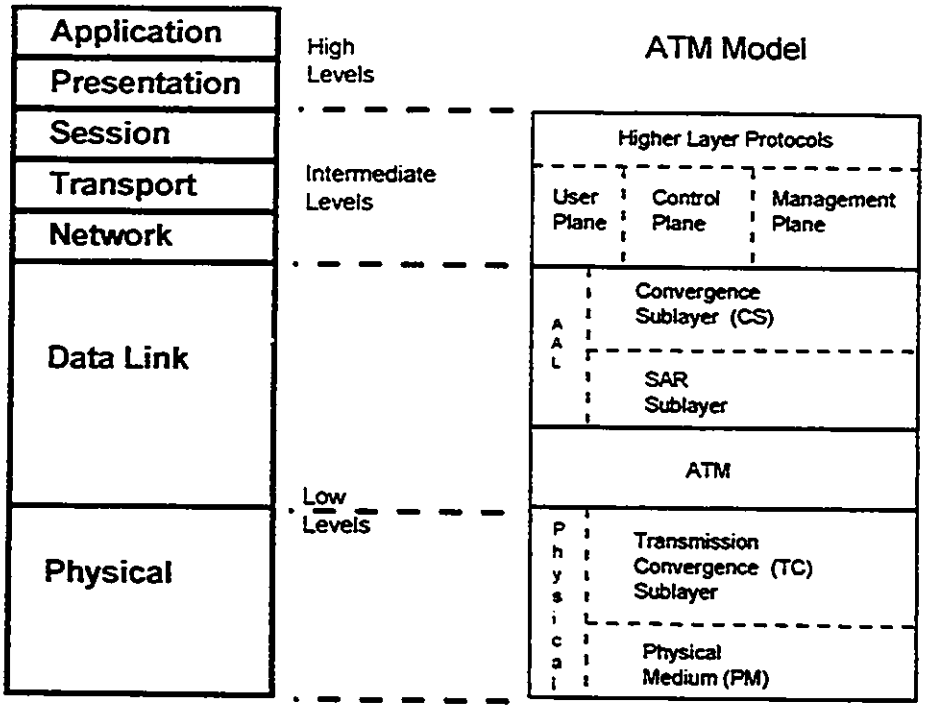


Figure 2.2: Open Systems Interconnection Reference Model and the ATM Model [MCD 95]

2.2.3 Media Integration

The aim of multimedia applications is to be able to efficiently communicate various types of media. Because each media type has very different manipulation and presentation requirements, the multimedia information system must provide support for media integration. Indeed, one of the major challenges of designing multimedia information systems lies in the ability to perform this media integration.

Media integration combines the various media elements so that they can be manipulated and presented by the multimedia information system. There are two types of media integration

required by multimedia information systems: spatial and temporal. Spatial integration groups media elements according to their spatial relationships. Such relationships include position, dimension, orientation and any information pertinent to the visual representation of the media. Temporal integration, or synchronization, deals with the timing relationships needed to render the media at the local workstation. These relationships include start times, durations, playback rates and any other information required to support real-time concurrent media presentation.

There are two aspects to the problem of multimedia synchronization: definition and delivery [STE 92]. Definition deals with the specification of the inter- and intra-media relationships of the multimedia application. Definition encompasses the set of language constructs, hyper-links, scripts and document architectures needed to integrate the multimedia information at the logical level.

Delivery deals with the actual presentation of the media information according to their spatial and temporal relationships. It is the delivery system which schedules the media retrievals, controls their synchronized playbacks and determines the output layout for the media information. The delivery system is the set of algorithms, protocols and servers which enforce the policies of the synchronization definition from source to destination.

In order to handle media integration definition, many systems use the concept of the multimedia document [EME 93, BUF 94, REI 94]. In general, multimedia documents provide architectures capable of representing the spatial and temporal relationships between the document's constituent media elements. The advantage of the multimedia document is that it

facilitates application design and integration by using the single document as the vehicle for the information interchange as opposed to juggling an amalgam of media elements along with their complex relationships.

2.3 DMIS Applications

There exist many application domains for which distributed multimedia information systems are required. These application domains include: training, education, simulation, publishing, public information, groupware, sales, advertising, entertainment, as well as applications which may yet evolve as distributed multimedia information systems may become widely available. In this section, three DMIS applications will be reviewed so that their common components may be identified as base requirements for distributed multimedia information systems in general. The applications are multimedia learning, multimedia news and computer supported collaborative work (CSCW).

2.3.1 Multimedia Learning

The usefulness of computers in education has already been widely recognized, even before the arrival of multimedia technology. However, in the past, the benefit brought by computer systems was mainly due to their ability to efficiently perform time-consuming, repetitive administrative and clerical tasks. Although testing and monitoring were easily

performed by computer systems, they were seldomly used for teaching purposes because they offered no distinct advantage over conventional methods [HOD 90]. The advent of multimedia technology, however, has renewed interest in computer based learning [HEP 94, LAR 94]. The main reasons for this renewed interest is because distributed multimedia information systems can enhance computer learning systems by improving information sharing and information reachability.

Multimedia learning systems are typically composed of a multimedia production center, database and user workstations, all of which are connected by a computer network (figure 2.3). The production center is used to digitize media and create the documents which will be used for the development of teaching materials and student assignments. These documents and digitized media are stored in the multimedia database. Students and teachers are connected to the computer network from workstations which may be in classrooms, offices, labs and even their own homes.

Multimedia learning applications can increase information sharing because of the multimedia database which is inherent to these systems. This database allows many users to access the same information through the computer network thereby increasing the distribution of information to local and remote users. The database also has a greater potential for information re-use, since files and documents which may have been created for one course may be copied wholly or in part for the creation of other course materials. For instance, a course document which was designed to teach group interactions for the psychology department may also be used to teach team management for the administration department.

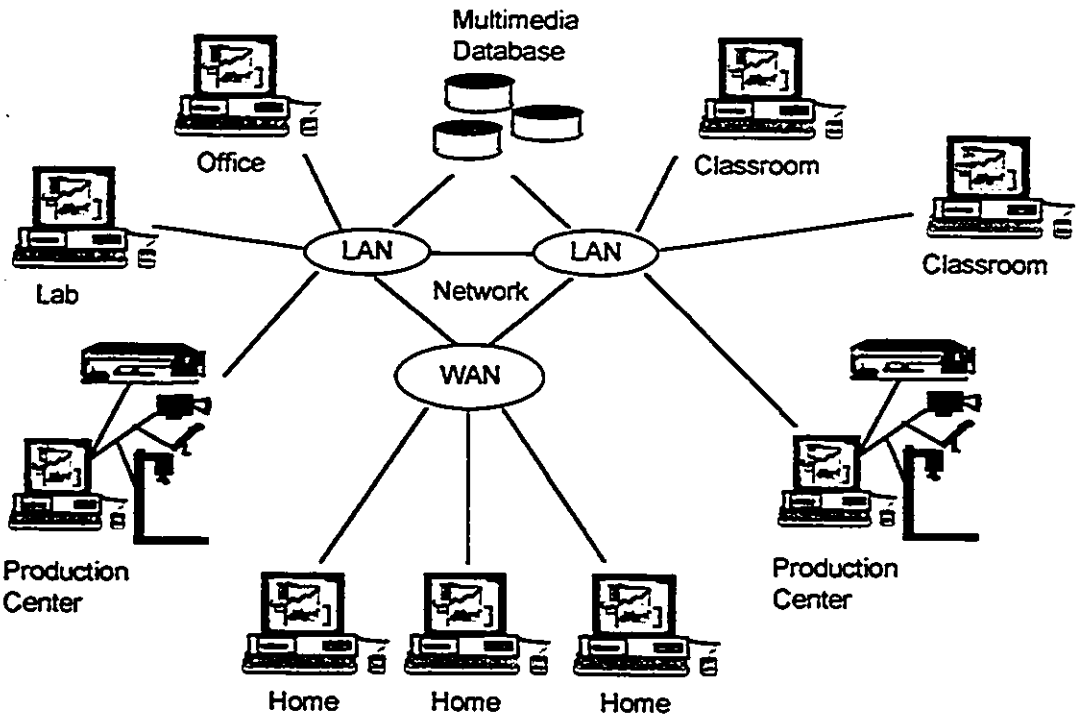


Figure 2.3: General Architecture of a Multimedia Learning System

Multimedia learning applications can also increase course material production. Because the production center of the distributed multimedia information system permits educators to digitize images, sound-tracks and video-tracks, these systems can thereby remove the dependency on expensive material resources. For example, a lesson on the modern art works of the National Gallery could be created by digitizing and presenting only that portion of a video filmed at the gallery. Educators gain two-fold from this use of digital computer technology. Firstly, educators can now create shorter and more specific sequences of information from larger generalized media sources, and secondly, educators can use materials which result from their own personal experiences. For instance, the video could have been taken during the teacher's last vacation or field trip at the gallery.

Multimedia learning applications can improve information reachability because they provide support for a wider choice of media than conventional teaching methods. Course materials communicated through text, image, audio and video formats are absorbed with varying degrees of efficiency by different viewers. Therefore, by designing lessons which communicate information in as many formats as possible, the reachability of the information is optimized for all potential viewers.

The most interesting innovation brought to education by multimedia technology is learning through computer simulation. Simulations permit users to experience events and learn from these experiences. Although no simulation can provide a full or complete experience, partial experience is better than none [HOD 90]. Simulations also allow for immediate explanation and interpretation of experienced events, which is not always the case for real-life situations. Through experience and explanation, multimedia simulations help create a unique learning environment.

2.3.2 Multimedia News

Multimedia news is an initiative which aims to provide news on-demand to satisfy the information requirements of a highly diversified client-base [MIL 93]. The multimedia news system exhibits many advantages over the news services currently available. In effect, multimedia news systems built on multimedia databases can cater to specific client interests. Clients create a profile to register their current interests with their news providers which will then only communicate the information matching this profile. The advantage of the profile is that it permits

the client to receive specific news stories without eliminating the possibility of later retrieving other information from the multimedia news database. The profile also reduces the burden involved in distributing information conceived to satisfy a general client base.

Another advantage of multimedia news systems is their ability to communicate time-critical information. A change in the interest rate announced by the central bank is an example of a time-critical news event. Investors may immediately react to the change and modify their portfolios accordingly. If investors receive the announcement too late, it could represent a significant loss of value of their portfolio.

Multimedia news systems are also more efficient than current news systems when it comes to information management. Information management includes the production, distribution and storage of news stories. Through computer digital technology, news providers will be able to consolidate the production of video, audio and printed news stories within a single news production centre (figure 2.4). Distribution can be geared to satisfying the needs of individual clients as opposed to satisfying printing and broadcasting deadlines. Distributed multimedia databases using broadcast, multi-cast and point-to-point communication protocols will provide the support for client profiles requesting news alerts, story updates, prioritized information delivery or even regular information broadcasts.

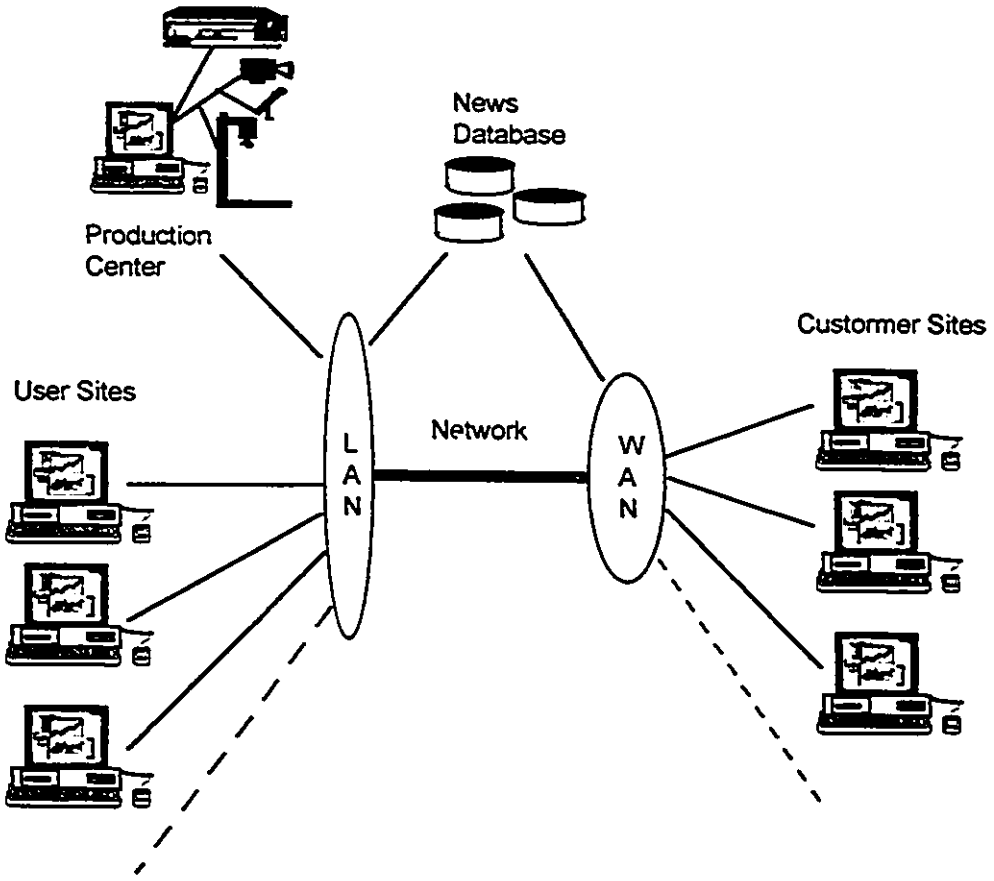


Figure 2.4: Architecture for a Multimedia News System

2.3.3 Computer Supported Cooperative Work

Computer supported cooperative work (CSCW) or "groupware" aims to improve the way groups of people collaborate through better use of computer and communication technologies. Consulting, video conferencing, joint editing, and project planning and management, are a few examples of CSCW applications. In all of these applications, the group works together to achieve a common goal. In joint editing for instance, the group's common goal is to produce a document.

While one member creates an image for the document, another prepares the textual information to accompany the picture.

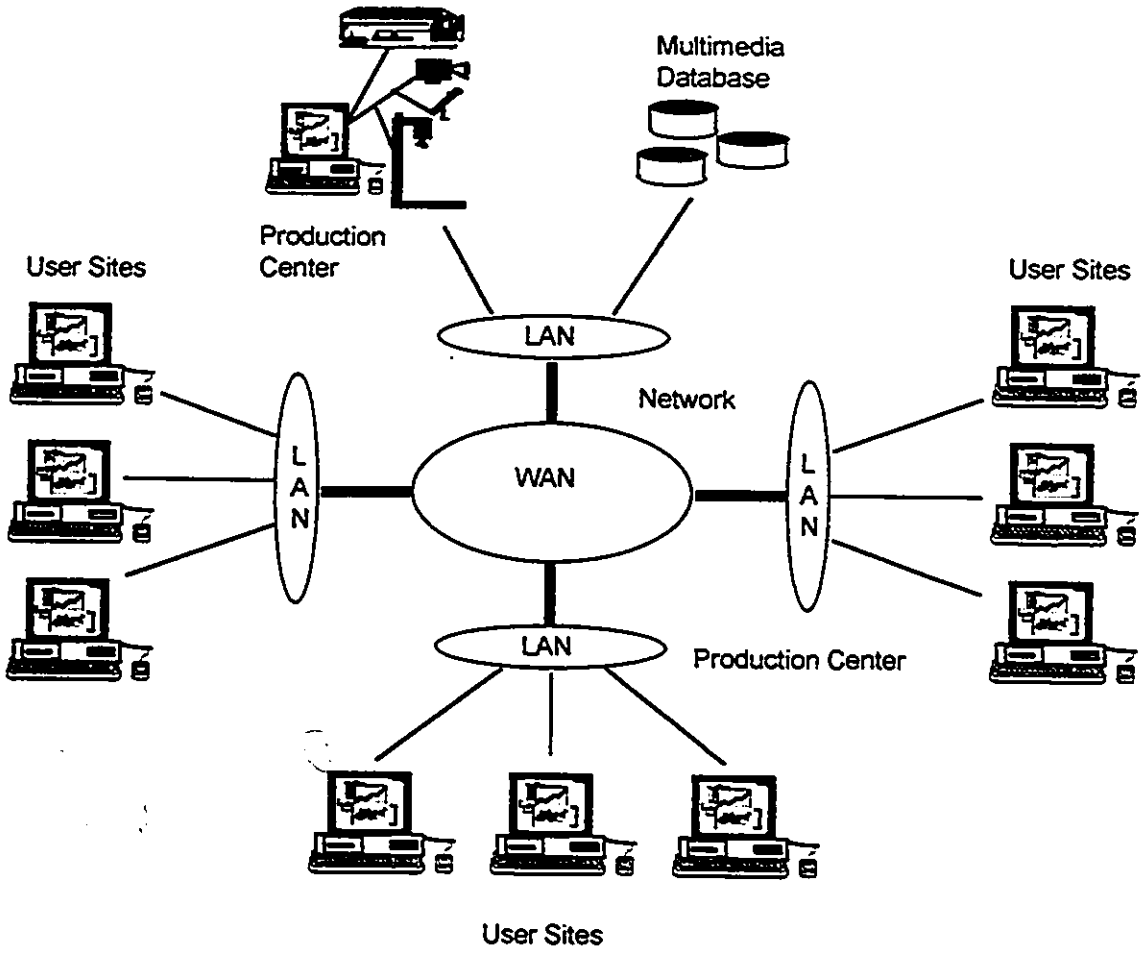


Figure 2.5: A Multimedia Collaborative System

There are several aspects to consider in order to successfully realize CSCW applications. These aspects include multi-user interfaces and collaboration strategies, protocols for synchronous and asynchronous communications and the sharing of data, services and resources within a distributed environment. Because many of these aspects also apply to distributed multimedia

information systems. DMISs provide a solid foundation from which cooperative multimedia applications can evolve.

In general, cooperative multimedia uses the concept of a multimedia document to organize, represent and structure information exchanged between remote cooperating agents [KAR 93]. For example, the multimedia document is the end goal of joint editing applications; project schedules and progress reports are created by project planning and management applications; and even video conferencing can use the concept of the multimedia document by recording the audio, visual and data information into the minutes of the conference.

CSCW applications will potentially increase productivity and efficiency of group collaborations. DMISs will provide the backbone for CSCW applications and help create an ideal communication environment for representing ideas in the greatest number of ways. CSCW applications will help reduce the expense and time required for members to organize and travel to their meetings. CSCW applications will also help reduce the cost and time associated with the production and revision of information by making use of digital computer and database technologies.

2.4 Conclusion

The ability to exchange various types of media within widely diversified information networks will create a new communication environment with a greater potential for information

reachability. This new communication environment is the result of the distributed multimedia information system (DMIS). Not only do DMISs enhance the way information is represented, but they also improve the way this information is communicated. DMISs can increase information sharing, cross geographical and temporal boundaries and they can also help create a global interactive communication environment for all types of media information.

In this chapter, we have reviewed distributed multimedia information systems in general, as well as three potential applications for these systems. From this review, several components which are common to all DMISs have been identified. These components include production centers, storage centers, user sites and computer communication networks. Technologies which are currently used in the production and storage of computer information, as well as the ability to communicate this information to remote users through computer networks will have to evolve to overcome the complexities brought on by the large, real-time nature of multimedia information.

Two problems were identified as major obstacles to the successful realization of distributed multimedia information systems. The first problem is due to the distributed nature of distributed communication systems and services. Client/server architectures were presented as a means to overcome this problem. By breaking down services into functional layers, system implementation is made more manageable by the re-use of peer services via standard protocols.

The second problem identified in this chapter is media integration. Media integration combines the various types of media so that they can be stored, manipulated, communicated and presented coherently by the distributed multimedia information system. Of the two types of

media integration needed, only temporal integration, or synchronization, is needed for the production, storage, network and user interface components of the distributed multimedia information system.

The multimedia synchronization problem is broken down into two parts: synchronization definition and synchronized delivery. The case for multimedia document architectures was presented as a means to implement the definition aspect of the multimedia synchronization problem. Such architectures allow for the design of multimedia applications to be independent of the system targeted for the implementation. Although multimedia documents provide models to logically integrate multimedia data, the actual physical integration of the multimedia data will be controlled by the set of services which make up the synchronized multimedia delivery system. The next chapter will develop the issues involved in multimedia synchronization in order to develop synchronized delivery systems for distributed multimedia information systems.

Chapter 3
Synchronization in a Distributed Environment

3.1 Introduction

Synchronization is responsible for the end-to-end temporal presentation of the multimedia information. In the previous chapter, we have identified two aspects to the problem of multimedia synchronization. They are synchronization definition and synchronized delivery [STE 92]. Synchronization definition deals with the specification of the synchronization relationships between the various media within a multimedia application. Synchronization definition

encompasses the set of language constructs, hyper-links, scripts and document architectures needed to integrate the multimedia information at the abstract or logical level.

Synchronized delivery deals with the actual presentation of the media information according to their temporal relationships. It is the delivery system which schedules the media retrievals and controls their synchronized playbacks. In order to sustain a coherent communication, the synchronized delivery system must ensure that the media are delivered to the reader at a discernible rate. Because computer systems can deliver information at speeds which can be either too fast or too slow for readers to absorb, the synchronized delivery system is required to present the media information according to their "bounded" perceptible delivery rates. The delivery system is the set of algorithms, protocols and servers which enforce the policies of the synchronization definition from source to destination.

In distributed multimedia information systems there are three levels of synchronization: the source level, the transport level and the presentation level (figure 3.1). The source level synchronization is responsible for retrieving the media information from the computer system's persistent store. The presentation level is responsible for the synchronized playback of the media to the user while the transport level synchronization contends with the issues involved for the synchronized retrieval of media information over the communication network.

All of these levels work in conjunction to provide a reliable multimedia information system which will deliver the media according to their synchronization definitions. Multimedia synchronization systems must be able to handle both the inter-media and intra-media

synchronization definitions. Inter-media synchronization deals with the temporal relationships between autonomous media. If for example, simultaneous audio and video rendering is required, it is the inter-media synchronization which must ensure the lip-synchronization between the two media.

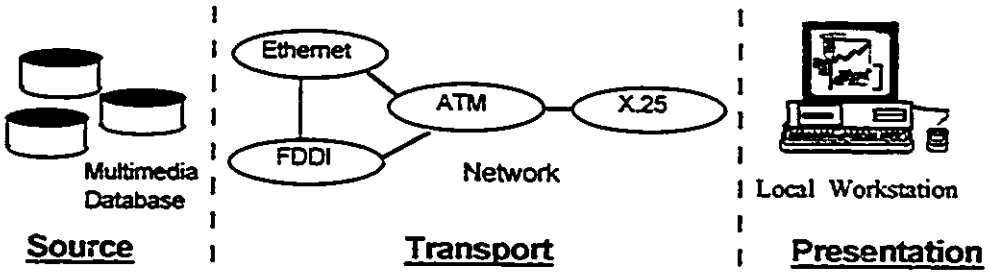


Figure 3.1: Three Levels of Synchronization for Distributed Multimedia Information Systems

Intra-media synchronization which is sometimes referred to as the self-synchronization, deals with the scheduling of the media information. The intra-media synchronization also deals with the temporal ordering of information segments for continuous media. For example, consider the rendering of a NTSC video stream or a voice quality audio stream. The intra-media synchronization is responsible for rendering these continuous media streams according to their defined data delivery rates: that is 30 frames per second for the video and 8000 samples per second for the audio stream. Therefore the intra-media synchronization must present one frame every 1/30th of a second and one sample every 1/8000th of a second.

Both the sources and destinations of the media are responsible for maintaining synchronization. In distributed systems, there are four generalized data location models [BUL 91]. These models which are shown in figure 3.2, are the local single source, local multiple sources, distributed single source and distributed multiple sources. Because of the different

possible topologies between sources and destinations which may result in varying data paths for distributed media, most distributed multimedia information systems rely mainly on the destinations to maintain synchronization [EHL 94].

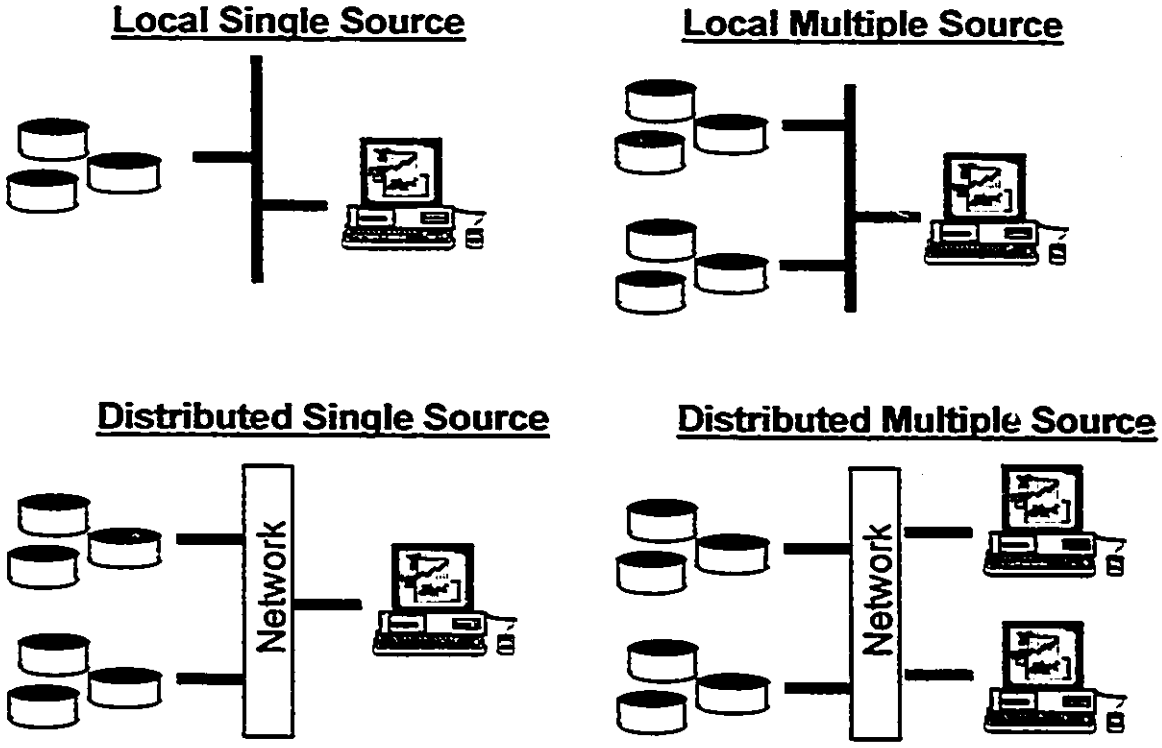


Figure 3.2: Data Location Models for Distributed Systems [EHL 94]

The satisfactory delivery of multimedia information will depend on the ability of the distributed multimedia information system to manage the flow of the various media types. In distributed multimedia information systems there are many sources which contribute to the complexity of the synchronized delivery of multimedia information. The next sections of this chapter will develop specific solutions used to resolve the problem of multimedia synchronization. In section 3.2, the use of multimedia documents for synchronization definition is discussed. Section 3.3 follows with the basic techniques used to implement inter-media and intra-media synchronization. Sections 3.4 and 3.5 present two strategies which are used to reduce the

complexity of synchronized multimedia information retrieval. These strategies are the progressive document retrieval strategy and dynamic quality of service management.

3.2 The Multimedia Document

Electronic documents are the most widely used form of information representation within computer systems. Indeed, documents account for at least 80 percent of corporate electronic information [REI 94]. This widespread use of electronic documents is due to the fact that electronic documents provide standard information interchange formats. These standard formats increase portability of documents between different computer systems and applications and therefore increase information sharing and re-use.

The ISO committee has proposed the Office Document Architecture (ODA) as a standard for the interchange of electronic information [ISO 88]. This document model has been created specifically for the office environment and is used to represent electronic documents composed of text, graphics and images in terms of their logical, layout and content structures. The logical structure represents the hierarchical relationships used to organize the media information into logical units. These logical units, which may be used to represent paragraphs, sections or chapters are generally needed so that the information conveyed may be understood with greater ease. The content structure is the actual data definition of the electronic document. Specifically, the content structure represents the actual information and type of information which the author wishes to

communicate to the reader. The layout structure represents the spatial relationships or the two-dimensional view of the electronic document and its media elements.

The Standard Generalized Markup Language (SGML) is another document model used to standardize electronic information interchange. It was specifically designed for the publishing field and is used to represent documents like books, journals and encyclopedias. The SGML syntax uses a set of generic tags for coding specific document type definitions (DTD), along with the allowable contents and attributes of the document's elements. Although SGML is used to represent electronic documents in terms of their logical structures and supports the notion of content structures, SGML does not support layout structures needed for the consistent presentation of the document's media elements.

The ODA and SGML document models are ill suited for multimedia information systems because they cannot integrate time-based media, nor can they represent temporal relationships between the document's media elements. Therefore, such document architectures must be extended to include the intra-media and inter-media temporal relationships needed for multimedia information systems. Intra-media specifications will require attributes which characterize the media's start time, end time, duration and playback rates. On the other hand, inter-media specifications will rely on a set of links which identify the type of synchronization and the media or events involved in this synchronization relationship. Hence, the combined intra-media and inter-media specifications define a temporal structure. Adding the temporal structure to the logical, layout and content structures of conventional document models, creates a new multimedia

document architecture necessary for the specification of interchange formats for multimedia information systems.

HyTime or Hypermedia/Time-Based Structuring Language is an ISO document standard which is based on the the SGML document architecture. HyTime documents achieve media integration by associating media elements with coordinate systems which may represent space, time or any other quantifiable dimension [BUF 94, NEW 91]. The main disadvantages of the HyTime multimedia interchange format is that it does not support interactivity and like SGML, it does not support presentation modeling [BUF 94].

Several other document architectures have been designed to standardize multimedia information interchange. Notably, the MHEG (Multimedia and Hypermedia information coding Expert Group) document architecture. This is an ISO multimedia document standard which defines an object-oriented model for multimedia and hypermedia information interchange. The standard directly supports interactivity as well as the real-time delivery of multimedia information over computer networks [BUF 94].

Multimedia document architectures provide several advantages over monolithic systems which attempt to control all multimedia information within single processes. Firstly, multimedia documents standardize information interchange which allows information to be shared by a greater number of applications and systems. Secondly, these multimedia documents can carry with them information about their origins, identities and executable code needed to manipulate or render them [REI 94]. Multimedia documents are dynamic, modular entities which simplify

integration and allow for greater flexibility in the organization and presentation of the multimedia information.

3.3 Synchronization Techniques

The end goal of the distributed multimedia information system is to be able to deliver messages coherently. Therefore, the synchronized delivery of the multimedia information is the most crucial aspect of the distributed multimedia information system. Indeed, if the media synchronization of the DMIS performs un-satisfactorily, the entire system appears to fail.

Recall from section 3.1 that there are two components to multimedia synchronization: the intra-media and inter-media synchronization. Inter-media synchronization is responsible for the joint delivery of autonomous multimedia data. This type of synchronization is used to sustain both loosely-synchronized media like audio annotated images or slide shows and tightly-synchronized media like lip-synch audio/video presentation.

Techniques used for achieving inter-media synchronization can be generalized into three different categories. The first category of inter-media synchronization simply interlaces the delivery of the different media within a single process. MPEG audio/video players use this type of inter-media synchronization. The MPEG presentation application reads an MPEG formatted file which is composed of groups of compressed video and audio tracks. The application reads a group structure, extracts the video, extracts the audio, plays the audio, then plays the video.

Although this type of inter-media synchronization is simple and effective, an application based on the interlaced synchronization method can only be used for the target media types or file formats.

The second category of inter-media synchronization is the time line technique. Such techniques involve creating delivery processes for each media and synchronizing these processes to a common time line or real-time clock. Inter-media synchronization is achieved indirectly because logically synchronized media are not synchronized to each other, but are synchronized to the clock. The advantage of this technique is that any number of media can be synchronized simply and effectively.

The third category for inter-media synchronization is the rendez-vous technique. This technique creates delivery processes for each media and relies on inter-process communication (IPC) primitives to achieve inter-media synchronization. The rendez-vous synchronization technique is most complex of the synchronization techniques. It requires computer environments which support inter-process communications and necessitates more overhead than the other two synchronization techniques. However, the rendez-vous synchronization technique ensures true inter-media synchronization and, unlike the time line synchronization technique, it is impossible for one delivery process to "run away" from another. Consider for example an audio annotated slide show with fixed inter-frame arrival times. If the time line synchronization technique is used and if for some reason a series of slides fail to arrive by their specified deadlines, the audio process may still play and therefore render meaningless information. Alternatively, the audio information may be discarded or even delayed since intra-frame delay does not adversely affect the joint presentation. The problem with the time line synchronization technique is that the audio

and image media are synchronized only logically, and therefore it is more difficult for the application to guarantee synchronization under such conditions. On the other hand, the rendezvous technique logically and physically synchronizes the media and can therefore explicitly guarantee synchronization under all stresses which may be applied to the computer communication environment.

The intra-media synchronization is responsible for scheduling the media's delivery. There are two common methods for scheduling multimedia information delivery. The first method relies on the ordering of the media objects within a multimedia document or application. By serializing the media information within a document or ordering sequential deliveries within an application a very simple yet naive form of media scheduling can be achieved.

Media scheduling can also be achieved by the use of relative time stamps and/or the generation of events which activate the delivery of the multimedia information. Event generation can be caused by time events, which will trigger the delivery once the specified (absolute or relative) time has been reached. Scheduling events can also be generated by other sources like the end (or start) of another media's delivery or even from external sources like the user clicking on a start button. This type of scheduling differs from the previous because it is event driven while the previous method is application driven. The disadvantage of application driven scheduling is that there is no concept of a scenario or temporal flow to the application. Such applications always follow some pre-conceived presentation format and therefore cannot alter their presentation as may be required in interactive communication environments.

Intra-media synchronization is also responsible for delivering the successive information segments of continuous media like audio and video. This type of synchronization must ultimately be able to deliver the multimedia information according to the playback rate defined for the media object. For example, the intra-media synchronization for NTSC video with a playback rate of 30 frames/second will require the delivery of one video frame every 1/30th of a second. This type of delivery is controlled by synchronizing video frames with a real-time clock.

The complexity of intra-media synchronization for continuous media is not due to the synchronization of successive information segments with the real-time clock. Rather, it is due to the underlying source hardware and software which attempt to deliver the media information according to their defined presentation rates. For example, consider a video presentation application which retrieves a video media from the computer's persistent store. If the video requires a playback rate of 27.7 MB/s and if the application can only retrieve 20 MB/s either because of a slow physical medium or because of a poorly designed retrieval process, the application will not be able to sustain the defined presentation rate requested for the video media [BAS 94].

In the next section, the progressive document retrieval strategy will be presented. This strategy is one of the two strategies previously mentioned, which are used to reduce the complexity of the synchronized retrieval within distributed multimedia information systems.

3.4 The Progressive Document Retrieval Strategy

Synchronization deals with the end-to-end presentation of the multimedia document to the local workstation. Generally, multimedia documents are composed of large, distributed data types. These data types cannot be stored at the local workstation because of its limited storage capacity. Therefore, the synchronization problem must contend with the real-time retrieval, processing, communication and presentation of the media from highly distributed environments.

In order to solve the synchronization problems of multimedia documents, the progressive document retrieval (PDR) strategy has been proposed by [KAR 93]. This strategy consists in retrieving, transferring and displaying independent synchronized entities (ISEs). An ISE is a sub-tree element which can be synchronized independently of any other sub-tree element within the multimedia document hierarchy. Figure 3.3 shows an example of a multimedia document, e.g. a multimedia newspaper, which is composed of three independent sections. The "Sports" section contains the "World Cup" and "Wimbledon" articles which are independent of each other. The article titled "World Cup", however, is composed of a text, an image, an audio and a video object. These objects are inter-dependent for their presentation and therefore the "World Cup" article is an example of an ISE in this document.

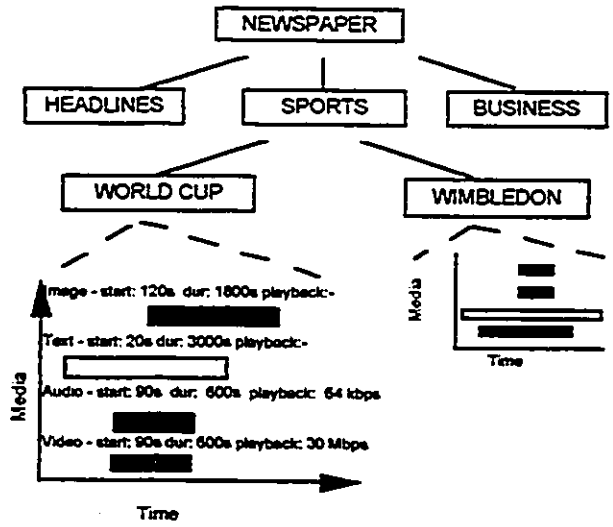


Figure 3.3: Multimedia Document Hierarchy demonstrating Independent Synchronized Entities

The progressive document retrieval strategy begins by parsing the document's temporal structure to identify the ISEs for retrieval. This is possible because the document's temporal structure may be transferred to the local workstation independently of the document's content. The progressive document retrieval strategy would then order the ISEs for retrieval according to the information available in the temporal structure or scenario of the multimedia document.

The Progressive Document Retrieval method allows us to display one ISE while its successor is being retrieved over the network. This method does not over-burden the storage capacity of the local workstation, but it does require continuous end-to-end synchronization for the entire duration of the document playback. This end-to-end synchronization must provide a consistent flow of information from the source, through the transport and into the presentation synchronizer of the DMIS (fig 3.4).

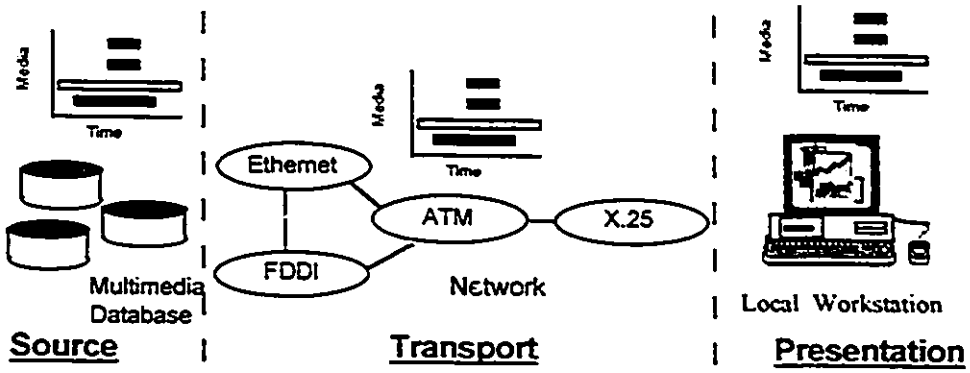


Figure 3.4: Progressive Document Retrieval for the Multimedia Document

3.5 Dynamic Quality of Service Management

A fundamental characteristic of any communication system is the quality of service (QoS). The QoS is defined as the collective effect of service performance which determines the degree of satisfaction of a user of the service [CCI 88]. This definition implies that an effective communication system depends not only on the actual service provided by the system but also on the perceived satisfaction obtained from this service. Therefore, the QoS may be approached from two different points of view. The first is from the point of view of the service quality offered by the communication system, and the second is from the point of view of the service demanded by the user of the system [TAW 93].

The quality of service demanded from the system is usually expressed in terms of certain service parameters. Such parameters include the throughput, maximum delay, jitter, bit error rates (BER) and packet error rates (PER). Jitter is defined as the instantaneous difference between two synchronized streams. On the other hand, the quality of service parameters which are used to

express or measure the performance of the communication network, also include the speed ratio, utilization ratio and the skew. The speed ratio is the ratio between the actual presentation rate and the nominal presentation rate for continuous data streams. The utilization is equal to the actual presentation rate divided by the available delivery rate. The skew is the sum of the jitter over "n" synchronization points [LIT 90].

In traditional communication systems, quality of service parameters are used to define and enforce media communication requirements and are essential for establishing transport protocols. They offer a means to negotiate the offer of a service and the demand for a service between a service user and a service provider. The use of QoS parameters is also needed for distributed multimedia information systems. However, these systems differ from traditional communication systems because of the largely heterogeneous nature of both the computer communication networks and the media being transmitted. Therefore dynamic quality of service (DQoS) parameters, along with some form of dynamic quality of service management is required for distributed multimedia information systems.

In today's extremely diverse communication environment, local area networks (LANs) and wide area networks (WANs) differ greatly among each other in terms of their speed, medium access times and reliability. LANs connected by Ethernet networks have a throughput of 10 Mbps and unbounded delay times, while LANs built on FDDI networks have bounded delay times and a throughput of 100 Mbps. Similarly, WANs can range from the 64 kbps throughput available on copper wire telephone lines to the 150 Mbps offered by B-ISDN networks. The reliability of computer networks can also vary significantly. The more reliable fibre optic networks can offer

bit error rates which are less than 10^{-9} while the less reliable copper wire computer networks offer bit error rates which are sometimes greater than 10^{-6} [HEH 90].

The type of media information being communicated is another constraint imposed on the communication system. Table 3.1 demonstrates the transfer requirements of the various media types in terms of their throughput, delay, jitter, bit error rates (BER) and packet error rates (PER) [HEH 90]. From this table we see that the transport requirements vary from relatively small, lossless, bursty data which is typical of text and image media to un-compressed video which requires the continuous transfer of very large amounts of data with relatively high bit error and packet error rates. Attempting to synchronize such widely differing media types which may also be distributed over many different communication networks is by no means a trivial task. Because the media and network both contribute to asynchrony within the communication system, asynchronous transport protocols must be designed to adapt to the media being communicated and to the network on which the communication is occurring.

Table 3.1: Transfer Requirements for Multimedia Information

Media	Average Throughput (Mbps)	Maximum Delay (sec.)	Maximum Jitter (msec.)	Tolerable BER	Tolerable PER
Text	2-100	1	N/A	0	0
Image	2-10	1	N/A	10^{-4}	10^{-9}
Voice	0.064	0.25	10	$<10^{-1}$	$< 10^{-1}$
Video	100	0.25	10	10^{-2}	10^{-3}
Compressed Video	2-10	0.25	1	10^{-6}	10^{-9}

* Note: data from [HEH 90].

The dynamic quality of service has been proposed as a means to overcome the diversity of service requirements which are characteristic of distributed multimedia information systems [LIT 90, HEH 90, BES 94, TAW 93]. The DQoS is a quality of service parameter which is continously managed and exchanged by the service providers and the service users. The DQoS provides a means to instantaneously quantify the performance of the communication system and subsequently represent the quality of service which is currently available for new users.

Service users express their service requirements with the DQoS parameter. They can also use this parameter to express bounds on the level of service which they are willing to tolerate. For example, a user may request the delivery of voice data with a PER of 1 in 1000 and a tolerable PER of 1 in 10. In the event that the desired service quality is not available or should become un-sustainable after the communication has begun, the DQoS is used to re-negotiate the delivery of the audio data. In this case, the tolerable DQoS indicates how much the service user is willing to sacrifice the quality of data delivery in favor of the delivery itself. If for instance the service user had expressed a tolerable PER of 1 in 1000 and if this quality service was no longer available to the user, the data delivery service would then terminate. In this situation, the user is not willing to sacrifice the quality of the data delivery and would rather terminate the delivery of service altogether.

The advantage of the DQoS parameter is that it provides the means to lower the quality of service when there is service degredation and to re-establish the higher quality service when it becomes available. This type of dynamic fallback and fall-forward negotiation is essential for asynchronous communication systems since they cannot guarantee constant service quality.

There are many external factors which can contribute to the degradation of quality of service in asynchronous communication systems. These include traffic, noise, data loss and even equipment breakdown. Because these factors are by nature non-deterministic, the DQoS parameter helps sustain the service throughout the continuously changing communication environment.

Another advantage of the dynamic quality of service concept is that it is not limited to the transport layer of the distributed multimedia information system. Because the DQoS is used to implement a protocol, the concept of the DQoS can be extended to any service level of the DMIS. Indeed, the next chapters will develop the idea of using the DQoS to manage the diversity of service requirements for the source, transport and presentation levels of synchronization.

Chapter 4
Architecture

4.1 Introduction

The architecture of the distributed multimedia information system designed for the MEDIABASE project is presented in this chapter. The presentation of this architecture will be focused principally on the components of the system which deal with multimedia synchronization, for both synchronization specification and synchronized delivery.

In order to handle the synchronized delivery of multimedia information, the distributed multimedia information system delegates synchronization services to the presentation, transport

and source levels of the DMIS shown in figure 4.1. The source level synchronization is responsible for the synchronized retrieval of multimedia information from the DMIS's file system and multimedia database. The tranport level controls the synchronized retrieval of the multimedia information across the system's communication network while the presentation level service delivers the synchronized multimedia information to its graphical user interface clients. Together, these three levels create a client/server architecture which controls the synchronized retrieval of the multimedia information from source to sink.

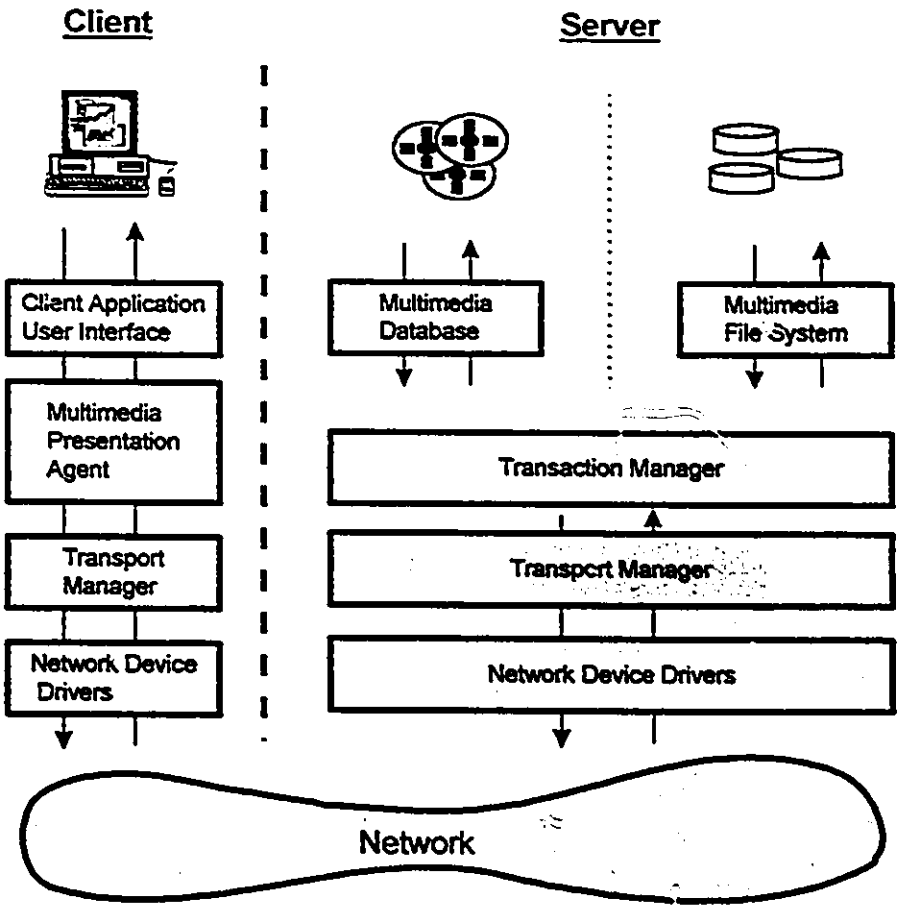


Figure 4.1: System Architecture

In this distributed multimedia information system, a multimedia document is used to handle synchronization specification. Recall from section 3.2, that the main advantages of using multimedia documents over monolithic systems for synchronization specification, is due to their abilty to exchange information between different applications and computer systems. The multimedia document acts as the vehicle for information interchage between the presentation, transport and source level synchronization servers. This multimedia document is based on MEDIADOC architecture [EME 93] which will be presented in the following section.

4.2 The MEDIADOC Document Architecture

MEDIADOC is a document model which provides a framework to describe and organize multimedia information objects and to represent their spatial and temporal relationships in a single entity called a multimedia document [EME 93]. MEDIADOC is an interactive multimedia document architecture which supports the integration of continuous media like audio and video, and provides a synchronization scheme to describe the synchronization specification between the document's media objects.

The MEDIADOC architecture is based on the ISO's Office Document Architecture (ODA) model. This model is used to represent electronic documents composed of text, graphics and images in terms of their logical, layout and content structures. The logical structure (figure 4.2) represents the hierarchical relationships between the media objects within the document. These hierarchical relationships exist for various levels of granularity; from the upper-layer independent

and scene related object blocks, to the mid-layer concurrent and sequential object blocks, down to the lower-layer basic media objects - text, image, audio, etc.[EME 93]. The content is the actual data definition, both data type and information, that the author wishes to communicate to the reader. The upper, mid, and lower-layer object types as well as the basic media object types define the content of the multimedia document. The layout structure (figure 4.2) represents the presentation relationships or the two-dimensional view of the document and its media.

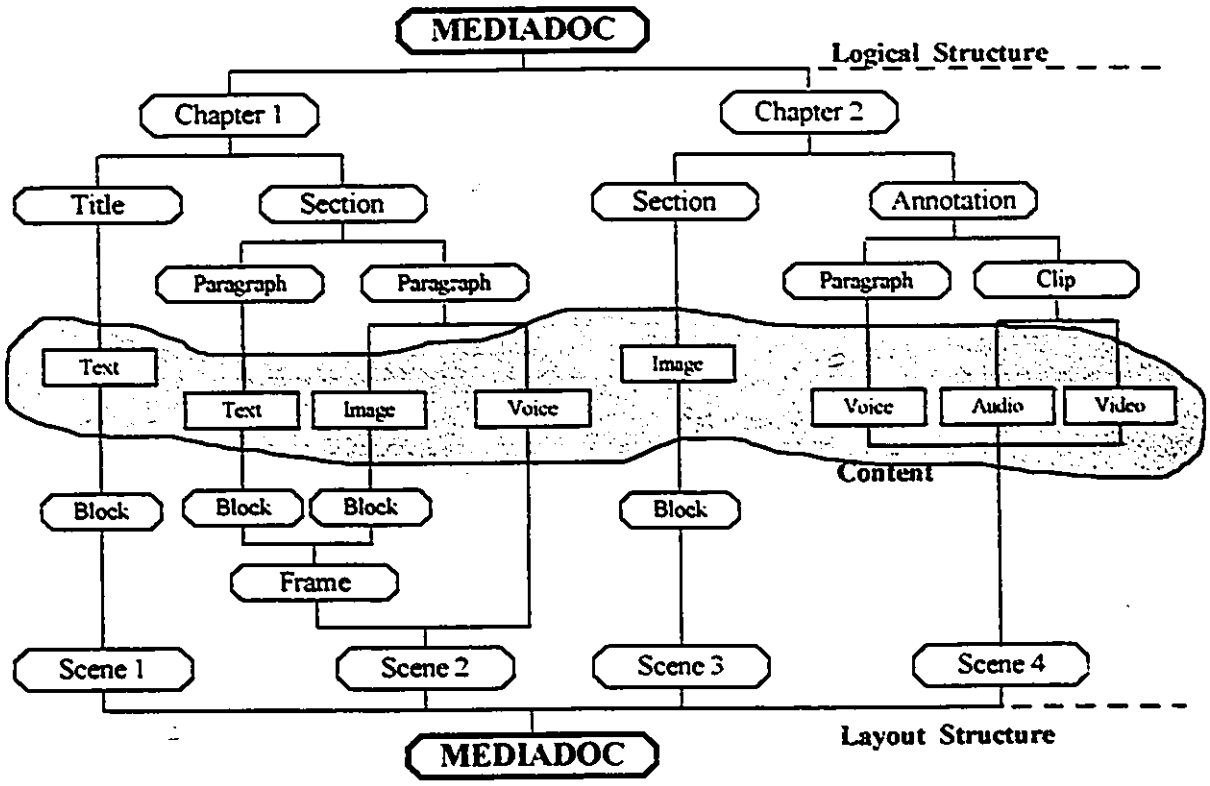


Figure 4.2: The MEDIADOC logical and layout structures

MEDIADOC uses a rendering schedule, referred to as a scenario (figure 4.3), to represent the synchronization information required for multimedia data integration. These scenarios are composed of live scenes which contain scenario objects. Scenario objects are used to define the set of temporal characteristics and relations required for the rendering of the multimedia document.

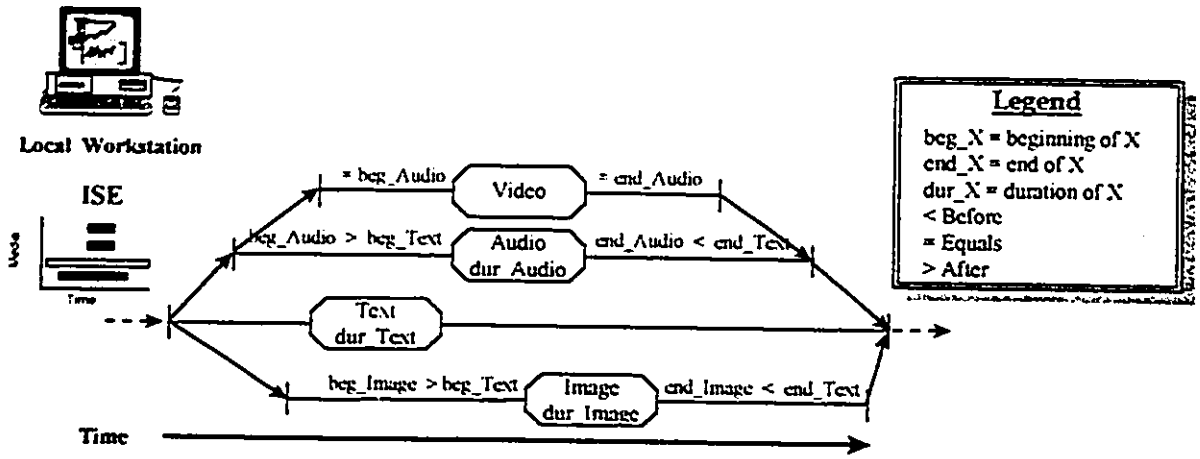


Figure 4.3 The MEDIADOC Scenario

The temporal characteristics of the scenario object are the start time, end time and duration of the scenario object. The temporal relations describe the synchronization relationship between the scenario object's start and end times with some kind of synchronization event. There are three types of temporal relations: before, equals and after. For example, a scenario object's end time may occur 5 seconds before the occurrence of some event. This event may be one of the four event types recognized by MEDIADOC. These event types are time of day, start or end of live scenes, start or end of scenario objects and other events such as keyboard or pointer input [EME 93].

MEDIADOC classifies scenario objects as being either embedded or non-embedded. Embedded scenario objects are the scenario objects that have temporal relationships with other scenario objects; on the other hand, non-embedded scenario objects do not depend on other scenario objects for their rendering. Therefore, the scenario of the MEDIADOC document can also be used to identify the independent synchronized entities (ISEs) of the multimedia document.

Recall from section 3.4, that the independent synchronized entities are the sub-tree elements of the multimedia document whose synchronization information is independent of any other sub-tree element of the multimedia document. The ISEs are used by the Progressive Document Retrieval strategy to optimize the synchronized playback of the document's media information. During playback, the scenario objects of the MEDIADOC document are rendered either independently, sequentially or concurrently therefore creating a set of sequential and parallel execution threads for each scenario object. The MEDIADOC scenario adds a temporal dimension to the document, thereby creating a document architecture capable of representing and manipulating its own presentation flow.

4.3 The Presentation Level

In distributed multimedia information systems, the presentation level is responsible for communicating the multimedia information at the local workstation. It is at the presentation level that the human/machine interaction occurs. This level also interfaces with the transport service provider. Because the presentation level is responsible for interfacing with the user and with the transport level, the presentation level must be able to manage event responses, information presentation and the communication connection.

There are two types of events which the presentation level must respond to: user events and system events [BAT 94]. The system events are the set of asynchronous occurrences generated by the computer configuration or communication environment. Examples of system events include termination requests signalled by a memory overflow or warnings generated when

a network connection is not responding to a transmission request. User events are the set of events generated by the user while interacting with the computer communication system. These events include the input type and click events supported by the user-interface as well as the output responses to these commands. In any case, the presentation level must be able to respond, meaning that it must implement a set of policies which will govern the course of action if a user or system event should occur. Therefore robust multimedia applications require that the presentation level map service policies to handle the various types of system and user events.

The multimedia presentation agent (MPA) is a functional component of the presentation level which manages information presentation at the local workstation (figure 4.4). The MPA offers a set of primitives to the client (graphical user-interface) application which controls the actual display of the multimedia information. The MPA also makes use of the service primitives of the network communication manager to handle the transmission of the media objects from the remote multimedia information source. This design allows the MPA server to be independent of both the graphical display routines of the local workstation and the communication environment of the computer system.

The primitives offered by the MPA to the client graphical user interface (GUI) are similar to the control buttons found on common audio/visual recording equipment: i.e. **stop()**, **pause()**, **rewind()**, **forward()** and **scan()**. The **stop()** and **pause()** primitives respectively abort and suspend the presentation processes for the multimedia document. The **forward()** and **rewind()** primitives relocate the relative temporal position of the multimedia document rendering process.

The **scan()** primitive permits the client application to render the document (forward or backward) at an accelerated rate.

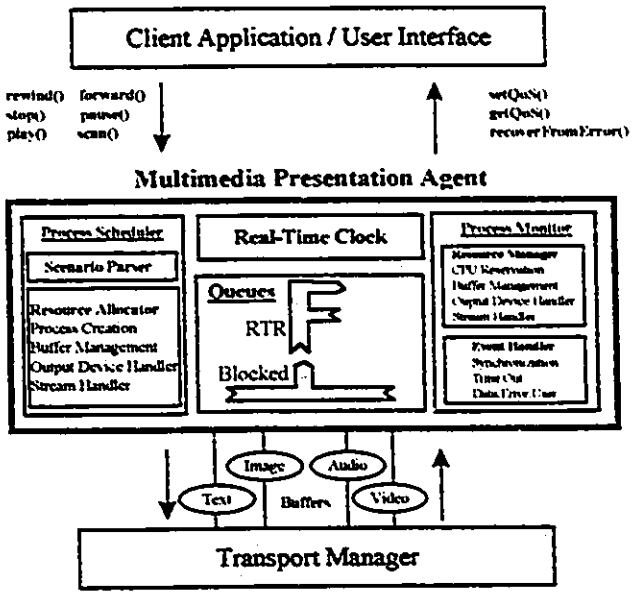


Figure 4.4: Multimedia Presentation Agent Architecture

The **recoverFromError()**, **setQoS()** and **getQoS()** primitives are the control and feedback primitives necessary to the client application. The **recoverFromError()** primitive is used to implement the service policies to handle the various types of system and user events. The **getQoS()** primitive is used to query the quality of service required for the presentation of media objects while the **setQoS()** allows the application to set the quality of service to a level which is available (or sustainable) by the application. Together these primitives are used to dynamically re-configure the presentation of the media information and therefore allow the system to adapt to the changes in the presentation environment. For example, consider a scenario where a video becomes covered by another media object. The application can tell the MPA that it need not physically retrieve the video information and can therefore free up system resources. The video

does continue its playback, however the frame retrieval is only at a logical level, i.e. it merely keeps track at the frame position within the video.

The client application begins document rendering via the **play()** primitive. Figure 4.5 demonstrates the flow control for document playback at the MPA. When the **play()** primitive is invoked, the MPA requests the scenario of the MEDIADOC document without the content of the document. Once the scenario is received by the MPA, the process scheduler parses the scenario and identifies the ISEs within the document. The ISEs are ordered according to their playback schedule and wait on their start times before they begin playback.

The process scheduler initiates the rendering of an ISE. When the start time of an ISE has been signalled by the real-time clock or by another ISE, the process scheduler creates a rendering process for each media within the ISE. These rendering processes control, monitor and synchronize the playback of each media within the ISE. The MPA controls the playback process of each media by suspending the process, thereby putting the rendering process on the blocked queue, or by resuming the playback process, i.e. putting the process on the ready-to-run (RTR) queue.

The media within the presentation process request their content parts from the remote information sources. The content of each media is generally segmented in either frames, samples, pages, lines or blocks. The presentation agent does not directly control the rate at which the segments are delivered to each media, instead the presentation agent requests a data delivery rate from the transport manager via a dynamic quality of service (DQoS) parameter.

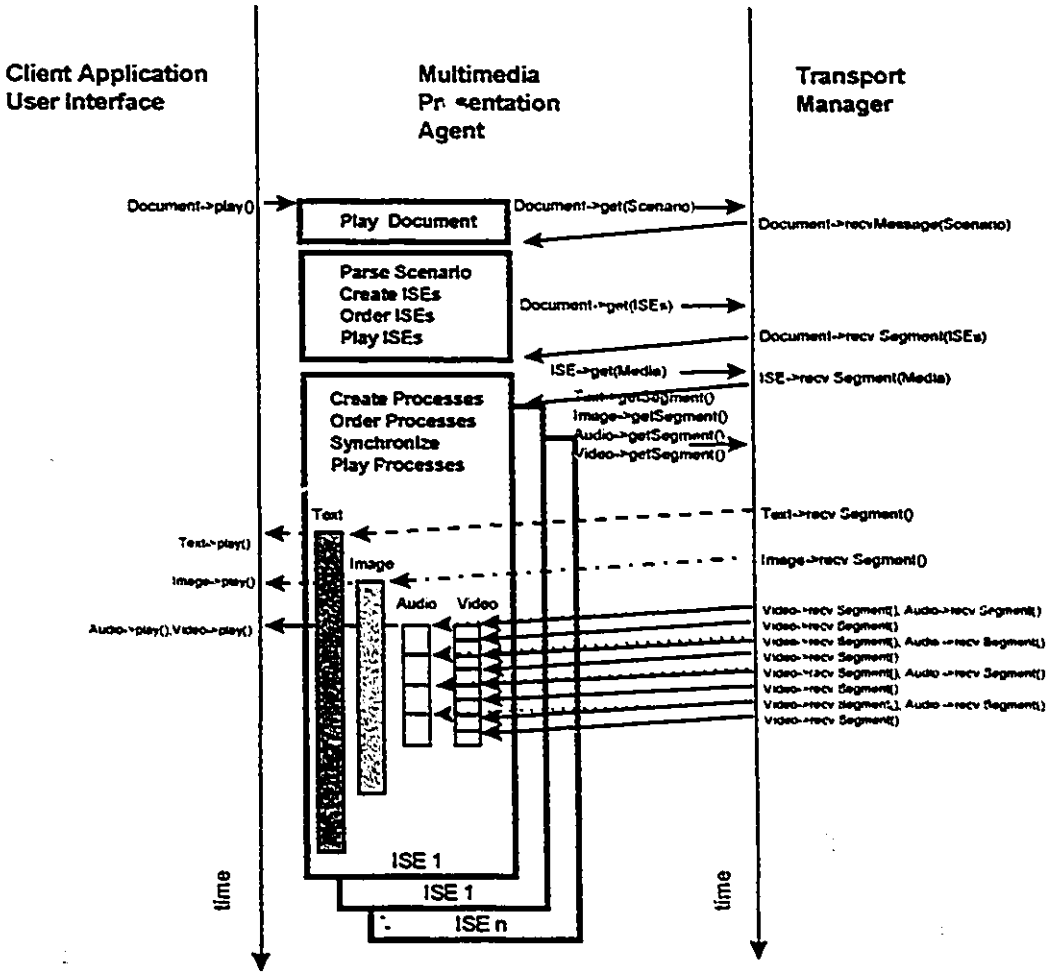


Figure 4.5: Flow Control for document playback

4.4 The Transport Level

The transport level handles the network communication aspect of the DMIS and therefore must deal with the problems associated with network communications. These problems include network traffic, bandwidth allocation, reliability and delay. Because each media type possesses different requirements with respect to bandwidth, delay, jitter, bit error rate and packet error rate, the transport level must be able to create communication connections which cater to these

differing transmission requirements. Also, the transport level must continuously monitor the transfer of information over the connection in order to provide a reliable delivery system for multimedia information.

The transport manager (figure 4.6) is the part of the DMIS which manages the flow of information through the computer communication network. This server defines a set of primitives which allow network clients to create two-way communication streams with remote information sources. The transport manager is built on top of the network layer which may use either IP, XNS, ATM, or other standard and non standard communication protocols. This design makes the transport manager portable across a wide variety of communication systems.

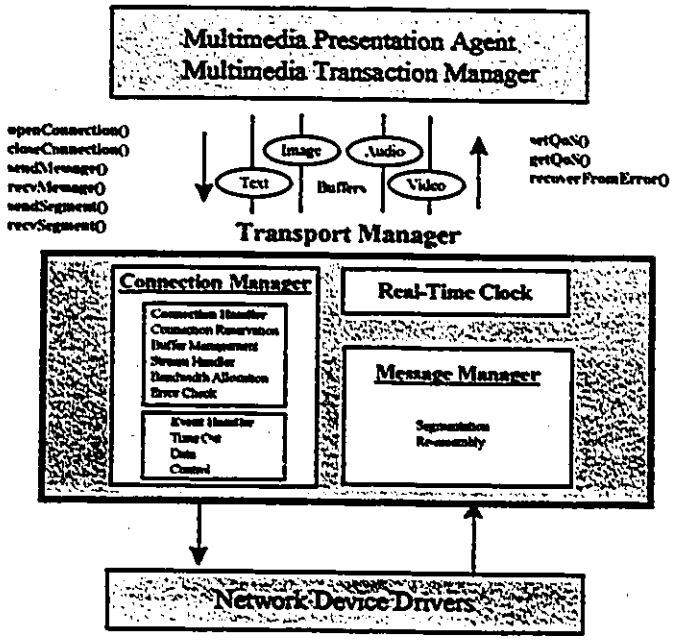


Figure 4.6: Transport Manager Architecture

The primitives offered by the transport manager belong to the domain of either the message manager or the connection manager. The connection manager encompasses the set of

services which handle the creation and maintenance of communication connections. The **openConnection()** primitive creates the connection while the **closeConnection()** primitive destroys it and releases the resources bound to the connection. The **getQoS()** and **setQos()** primitives permit the network client to query the available service quality offered by the transport manager and set or request the service quality for the connection.

In order to transfer information to a remote entity, a network client will invoke the **openConnection()** primitive of the transport manager. The client will then use a DQoS parameter and the **setQoS()** primitive to define the type of transmission service expected from the transport level. If, for instance, a client should request a 128 kbps bandwidth to transmit audio information, the connection manager will allocate this bandwidth if it is available. If the bandwidth is not available, the transport manager will notify the client of the currently available bandwidth, and allow the client to re-negotiate its bandwidth demand. In this case, the client may be contented with 64 kbps and therefore transfer the information at a reduced audio quality level.

The interface of the message manager include the **sendMessage()**, **recvMessage()**, **sendSegment()**, **recvSegment()** and **recoverFromError()** primitives. The **sendMessage()**, **recvMessage()**, **sendSegment()** and **recvSegment()** primitives are the basic communication primitives which allow information to be interchanged between communicating entities. The **recoverFromError()** primitive is the primitive which enables clients to map their service policies to network communication events, thus allowing applications to respond gracefully to service degradation.

Policies implemented to manage service degradation could vary from drastic measures which would naively abort the communication or even the application, to more sophisticated policies which may use the **setQoS()** and **getQoS()** to allow the client to dynamically re-negotiate with the transport manager, the service quality of the connection. For example, if a client was given a throughput of 20 Mbps to transmit video data at 30 frames per second when the connection was created, and if the available throughput should fall below this value sometime during the transmission, the **recoverFromError()** primitive will notify the client of this change. If the client's service policy allows the client to re-negotiate its desired quality of service, then the client could accept the 10 Mbps throughput and transmit every second frame only.

4.5 The Source Level

The source level of the distributed multimedia information system is the part of the system which retrieves the multimedia information from the system's persistent storage. The source level provides the services which interface with the elements of the system which store and retrieve the information to be delivered to the multimedia client applications. Therefore the source level is responsible for both media management and transaction management.

Media management is concerned with organizing and maintaining the multimedia information in order to optimize information storage and retrieval. Security, volume management, persistence and aging are a few services which fall within the scope of media management. These services belong to the domain of source management systems which include

file servers, database management systems, and live production centres. Multimedia applications which interface with these type of source managers engage in what is typically known as a master/slave relationships: that is the client application controls the session by always being the initiator of transactions and queries.

Transaction management is responsible for servicing the transactions initiated by the peer client application. A transaction is the set of actions which require access to the source level information of the DMIS. The transaction manager, must ensure that the source level of the DMIS remains consistent for all clients throughout the execution of all transactions being currently serviced.

In order to optimize the multimedia source servers, transaction management must account for the temporal nature of multimedia information. The components which are common to the source management systems are grouped within the transaction manager of figure 4.7. The transaction manager uses a message decoder to convert messages to transactions and a message encoder to translate the results into messages. The functionality of the transaction scheduler is similar to the MPA's process scheduler. The scheduler is used to anticipate information retrievals and to prepare the source servers so that retrievals may be optimized. For example if an ISE is being retrieved from the persistent store, the scheduler will use the ISE's scenario to order the retrieval processes, allocate buffers for each process and use whatever idle time the server may have to move data from the persistent store to the process buffers. Also, in the event that the client is interacting with a dialogue box which may cause the scenario to branch into different possible presentations, the scheduler can buffer the first ISE of all branches. When the user's

choice arrives back to the transaction manager, the unwanted ISEs may be discarded while the correct one is immediately available for retrieval. This allows the source managers to utilize the server's idle time more efficiently.

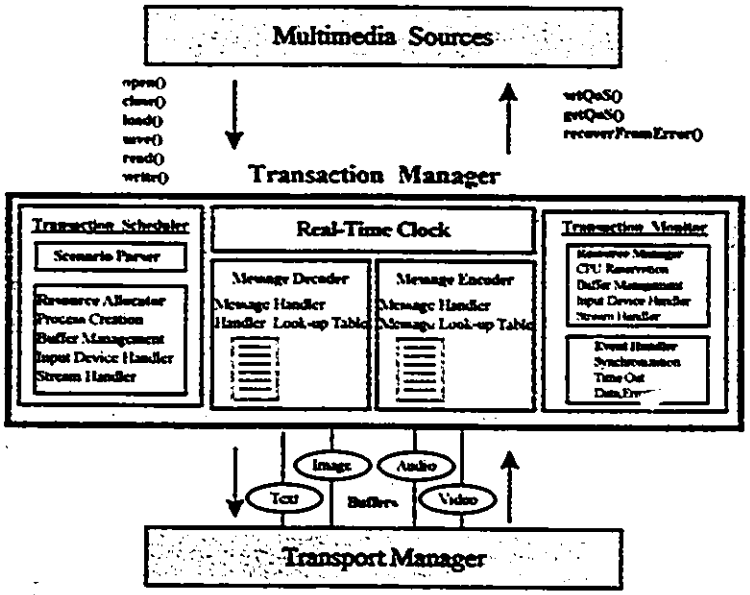


Figure 4.7: Transaction Manager Architecture

The transaction monitor is responsible for synchronizing the retrieval of the multimedia information. Ideally, the transaction monitor will deliver the media information to the client according to the media's playback rate. However, there are many constraints which may act on the computer system and thereby impede the retrieval of the media. Shared secondary memory resources which continuously serve many active clients is an example of these types of system constraints. In order to overcome these constraints the dynamic quality of service is used to negotiate the service offer and demand between the retrieval service and the client application.

Consider a client application which at a given moment is retrieving 30 frames per second of video data from the shared database. If the database suddenly becomes heavily loaded servicing many new client requests, the database manager may then only be capable of delivering 15 frames per second of video data to the initial client. The server can decide to either continue the video service at a reduced quality or terminate the service depending on the limits set by the client's DQoS parameter. The client's DQoS can also initiate a re-negotiation between the client and the server so that the client can reset limits for the transmission.

The transaction monitor tries to guarantee the data retrieval rate for each media by dynamically controlling either the buffers allocated for the retrieval process or by suspending and resuming the retrieval processes. The role of the transaction monitor is to produce the ISE's media elements according to the client's requested quality of service. In the event that the transaction monitor cannot meet the demanded quality of service, the transaction monitor is then responsible for managing the graceful degradation of services for the remote source.

4.6 Summary

This chapter described the architecture of the distributed multimedia information system which deals with the synchronized retrieval of multimedia information. Because DMISs require synchronization at three different levels, the components of the synchronization system were divided into three levels: the presentation level, the transport level and the source level. The multimedia presentation agent, the transport manager and the transaaction manager are the

synchronization components which are responsible for handling the synchronized retrieval of the media at the presentation, transmission and source levels respectively.

A client/server architecture was used to implement the three synchronization components. All of the components were designed to be independent of the graphics, network and storage environments of the computer system. Therefore the client/server architecture not only simplifies the design of the DMIS but it also adapts to different computing environments. By adapting to the different communication and computer environments, this client/server architecture maximizes re-usability and inter-connectivity between the different components.

The synchronization components exchange the media and ISEs of a MEDIADOC document. This multimedia document architecture extends on the ODA document model by incorporating a scenario which is used to define the synchronization relationships of the multimedia information. This model supports simple and complex document structures as well as interactive and multiple scenario representations. By using the scenario of the document, the synchronization components can use the progressive document retrieval strategy to optimize the retrieval of the multimedia document.

In order to adapt to the widely varying computer and communication environments, a dynamic quality of service (DQoS) parameter is used to control the synchronized retrieval of multimedia information. The DQoS is also used to adapt playback routines for varying types of media. The DQoS provides a mechanism which imposes constraints on the retrieval, transmission and presentation of the media data. These constraints can be statically and dynamically

configured so that the tolerable limits for the quality of service can be continuously negotiated between the different levels of the DMIS.

Chapter 5
Implementation

5.1 Introduction

This chapter presents the implementation of the Multimedia Presentation Agent, the Transport Manager, and the Transaction Manager which will be the source server for the DMIS. Because the DMIS uses a MEDIADOC document as the means for interchanging information between these components, the first step to implementing the system is creating the multimedia document model.

The first prototype for the DMIS was implemented using SUN Microsystems Inc. SPARC workstations. The file server was implemented on a SPARC/10 while the client applications were implemented for SPARC IPCs and IPX workstations. Each workstation has a 10 Mbps Ethernet connection, while certain workstations also have 100 Mbps FDDI and OC3 155 Mbps ATM connections (figure 5.1)

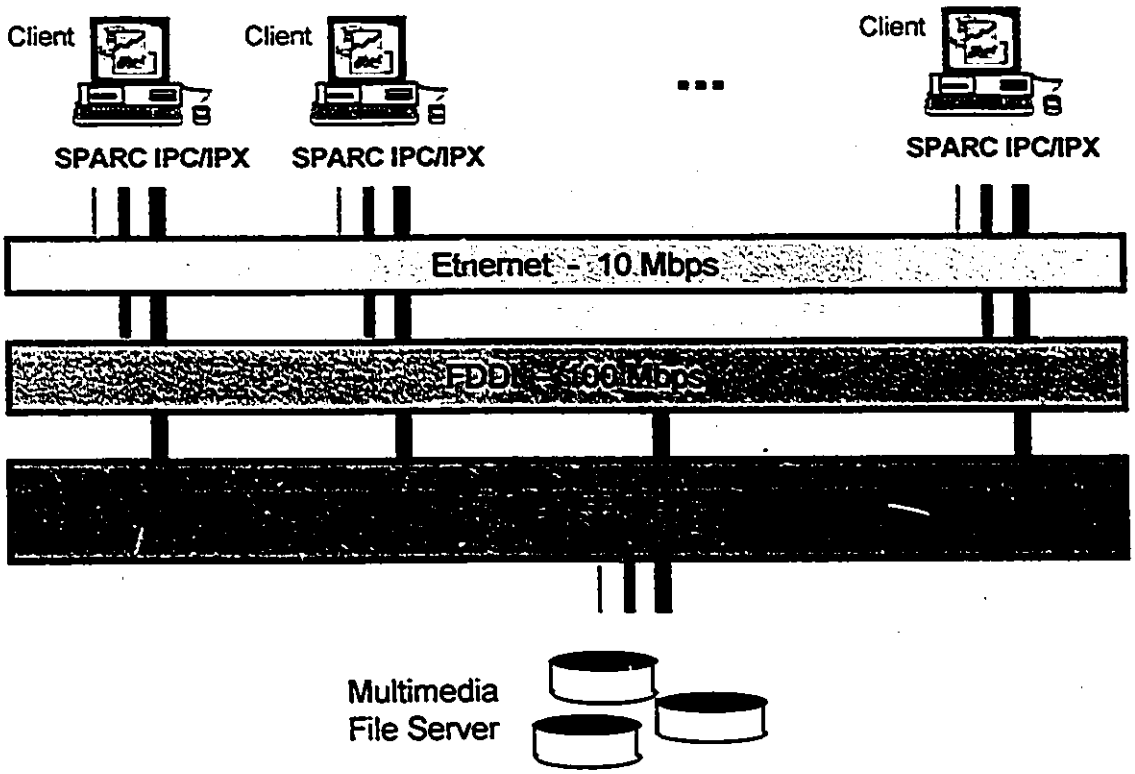


Figure 5.1: MEDIABASE Computer System

The operating system for the workstations is Unix running under the SUN Microsystems Inc. Solaris operating system ver 2.3. The programming language used to implement the multimedia document model, the MPA and the transaction manager is GNU's C++ ver. 2.7.1. The transport manager was implemented using the C programming language. The MPA's client interface was built using the C language with the Motif Toolkit [HEL 91]. Motif is a graphical

user interface toolkit which is designed to simplify the creation of large graphical user interface applications. Motif is based on the X11 graphical system protocol. The workstations mentioned earlier are currently running X11 release 5.

5.2 The MEDIADOC Document

An object-oriented data model was used to implement the multimedia document. The main reason for using an object-oriented design in order to implement the multimedia document is due to the compatibility between the object-oriented and multimedia document paradigms.

Object-oriented development is fundamentally different from traditional software development because the object-oriented paradigm allows real-world objects to be modeled by corresponding programming objects. Traditional programming environments are characterized by procedural programming paradigms which tend to model real-world tasks by procedures. In this approach data is secondary, while in object-oriented techniques objects have parts, and behaviours [PUG 94]. Therefore the object-oriented paradigm is a data flow paradigm while traditional techniques are control flow.

The MEDIADOC document has media objects which constitute its parts, and the relationships between these media objects define the behaviour of the document. These relationships establish the logical, spatial and temporal constraints which are needed to represent and manipulate the media objects of the multimedia document. Because the MEDIADOC

document models itself according to the relationships between its media objects, as opposed to relationships which may exist between functions within a procedure, the MEDIADOC architecture is therefore also a data flow entity. Therefore the object-oriented paradigm offers the best support to map the real-world document model to software implementation.

There are other advantages in using object-oriented techniques for implementing the data model. The main one results from the PIE principle of the object-oriented paradigm. The PIE principle, which stands for polymorphism, inheritance and encapsulation is the basic concept of object-oriented environments. Encapsulation groups parts and behaviour within type definitions while inheritance allows new types to be sub-typed from more general types. With inheritance, the sub-type can add additional parts and behaviour or modify and even disable the behaviour of the general type. Polymorphism allows the same operation to be performed by different types. The operation which is invoked depends on the object it is sent to and the arguments or signature of the method being invoked. The benefit of the PIE principle is that it maximizes software re-use by incremental development thereby allowing software components to grow from existing components and reducing the time needed to create these new components [PUG 94].

The first step in creating an object-oriented data model is to begin by identifying the problem domain of the multimedia document. Because the MEDIADOC document is destined to be the information exchange mechanism for our DMIS, the document model must be able to represent and manipulate the different kinds of media. Table 5.1 lists the different kinds of media, their parts and the operations which they must respond to. From this table, a general type is created by factoring the common parts and behaviour of all the media identified in the table.

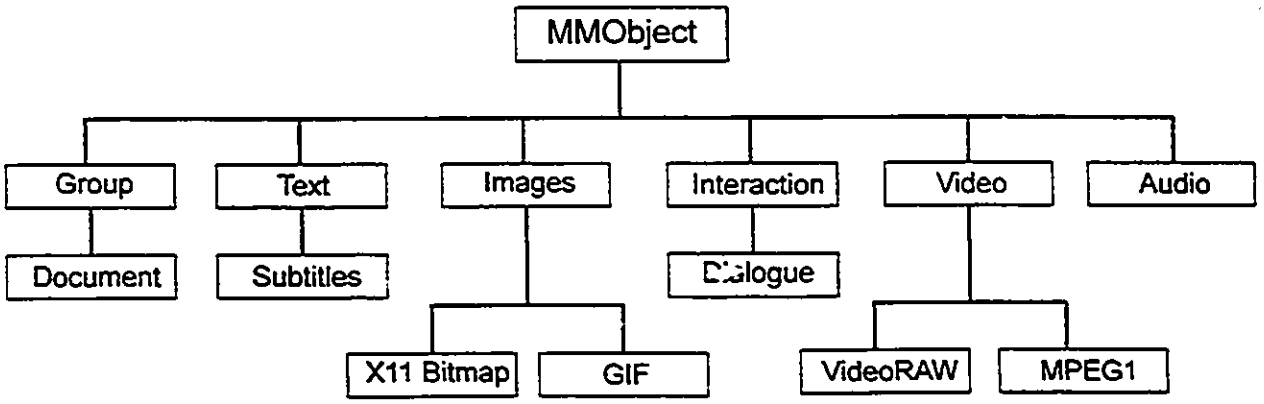
This general type or base class is the MMObject class and it defines the interface which is common for all media which are derived from it.

Table 5.1: The Multimedia Data Types

Parts			Behaviour		
Group	**Data**	**Media**	**Group**	**Operations**	**Media**
Spatial layout	Position	T, I, ,V,D,S,	Creation	new	T, I,A,V,D,S,G
	Size	T, I, ,V,D,S,			
	Orientation	T, I, ,V,D,S,	Member manipulation	Get_PART	T, I,A,V,D,S,G
Temporal layout	Start time	T, I,A,V,D,S,G		Set_PART	T, I,A,V,D,S,G
	Duration	T, I,A,V,D,S,G			
	Capture rate	, ,A,V, ,S,	Input and output	Load	T, I,A,V,D,S,G
	Sync. links	T, I,A,V,D,S,G		Save	T, I,A,V,D,S,G
Content	Data segments	T, I,A,V,D,S,G		Read	T, I,A,V,D,S,G
				Write	T, I,A,V,D,S,G
	Data size	,A,V,D,S,G		Seek	T, I,A,V,D,S,G
Information	Name	T, I,A,V,D,S,G	Rendering	Paint	T, I, ,V,D,S,
	Title	T, I,A,V,D,S,G		Play	T, I,A,V,D,S,G
	Authors	T, I,A,V,D,S,G		Synchronize	T, I,A,V,D,S,G
	Creation	T, I,A,V,D,S,G		Recover	T, I,A,V,D,S,G
	Modification	T, I,A,V,D,S,G			
	Keywords	T, I,A,V,D,S,G			
	QoS	T, I,A,V,D,S,G			
	Origin	T, I,A,V,D,S,G			
T = Text I = Image A = Audio		V = Video D = Dialogue S = Subtitles		G = Group	

Figure 5.2 illustrates the inheritance hierarchy from which all media classes are derived. This inheritance hierarchy demonstrates how the PIE principle was used to implement the MEDIADOC document and its constituent media objects. An instance of MMObject is made up of parts or data members which define the media objects in terms of their presentation, transportation and storage requirements. For example, the spatial_layout member, contains the position, size and orientation of the media object. The temporal_layout member contains the start time, duration, capture rate and the synchronization links of the media object. These two members

together represent the information required by the MPA to render the media object as it was intended to be viewed by its author.




```
class MMObject   – public interface for the MMObject class which is used to represent Multimedia objects
{
public:
// Constructors – are used to create and destroy instances of the MMObject class
        MMObject();
        virtual ~MMObject();

// Accessors  are used to get and set the data members of the MMObject instance
        const Information& GetInformation() const;
        void SetInformation( const Information& newInfo );

        const SpatialLayout& GetSpatialLayout() const;
        void SetSpatialLayout( const SpatialLayout& newSpace );

        const TemporalLayout& GetTemporalLayout() const;
        void SetTemporalLayout( const TemporalLayout& newTime );

        const GetContent& Content() const;
        void SetContent( const Content& newContent );

// Input / Output
        virtual int open ();                                    // opens a I/O stream defined in pOrigin
        virtual int close();                                    // closes the I/O stream
        virtual int write( int n );                             // writes segment n to the output stream
        virtual int read( int *nSegNum, int *time_out );        // reads segment n from the input stream
                                                                // until time_out expires
//  Presentation
        virtual void display( OutputDeviceHandle *aHandle );        // object rendering method

        virtual void pause();                                   // pause playback
        virtual void stop();                                    // stop playback
        virtual void play();                                    // playback the media object
        virtual void rewind();                                  // rewind the media object
```

```
        virtual void forward();                          // move the media forward
        virtual void scan( int nSegmentStep );           // render fwd or bkwd at accelerated rate

// Error Recovery
    int synchronize( int *time_out);                     // attempts to synchronize object within time_out
   virtual void getQos( int nParamID, QoS *aQoS );       //gets the QoS parameter identified by nParamID
   virtual int setQoS( int nParamID, QoS *aQuality );    //sets the QoS parameter identified by nParamID
   virtual int negotiateQoS(int nParamID);               // initiates a negotiation for the QoS parameter
   virtual int recoverFromError( int nErrorCode );       // error recovery routine

// Data members
private:
        Information     information;        //Author, location, QoS,..., general information of Object
        SpatialLayout   spatial_layout;    // position, size, orientation of the Media Object
        TemporalLayout temporal_layout;    // start time, playback rate and synchronization links
        Content         content;           // media data organized in variable length segments
};
```

Figure 5.2: Multimedia Object Class Hierarchy and Interface

The content member holds the actual data of the media object. The actual meaning or representation of the data demonstrates how encapsulation is used in this type of hierarchy. The data of any object of the hierarchy is segmented according to the class of the object. A VideoRAW object segments its data into individual constant size frames, while an MPEG1 video segments its data into variable size sets of frames. Each set is made up of different numbers of I, B and P frames which are grouped to optimize the transfer and playback of the MPEG1 video. ImageX11 and ImageGIF on the other hand, need only one segment to represent their entire content. Although each class inherits its data member and data accessor functions from the base class, they all override the methods used to manipulate (i.e. play, paint) the data member. From this example we see how encapsulation is used to re-define the meaning of the data member.

The information member is used to represent all the information pertinent to the media object: the object's name, its publication information (i.e. title, authors, dates of creation, modification, keywords,...), its QoS parameters, and its origin. The origin of the media object

specifies the host, the file or database, or even the input streams device or digitizer the media object is arriving from. The information member is designed to represent the complete identity of the media object.

The Document class is the class used to represent the MEDIADOC document. The Document object can be composed of any number of intermediate layer Group objects which in turn can be composed of any number of Group objects or any number and class of media objects. The reason for the Document class being sub-classed from the Group class and not vice-versa is because the Document class is a special type of Group: it is the highest level Group which must be unique within any instance of a Document object.

5.3 The Multimedia Presentation Agent

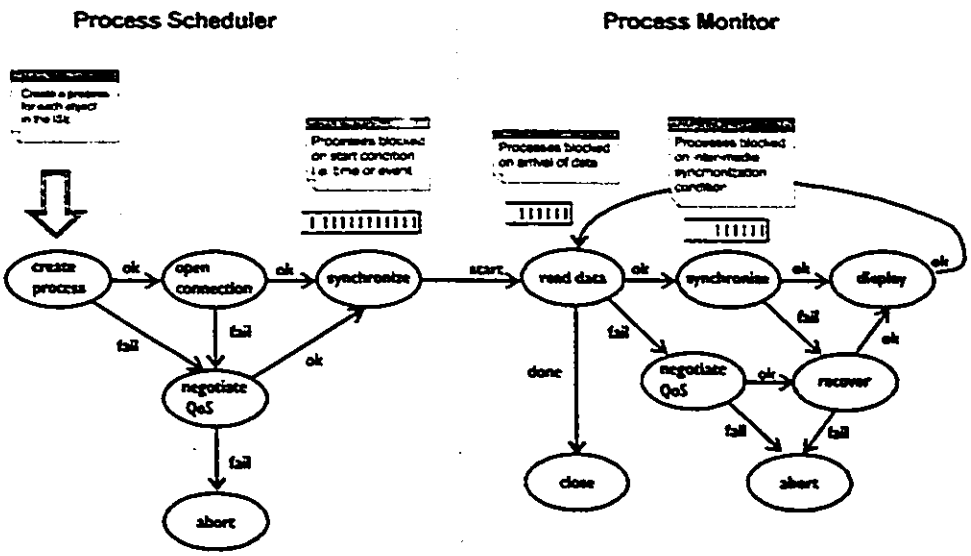
The multimedia presentation agent is the part of the system which handles the synchronization at the user interface level. Because it is the last component to exercise any control on the media before they are rendered at the user's workstation, it must guarantee the quality of service within the bounds expressed by the media author or the document viewer.

The primitives offered by the MPA interface are listed in table 5.2. The client application invokes document rendering via the **play()** primitive of the MPA server. Figure 5.3 illustrates the rendering procedure for ISEs. The process scheduler of the MPA will first identify the ISEs of the multimedia document. The ISEs in our data model are the lowest level Group objects which do

not contain any other Group objects. After the ISEs have been identified, a rendering process for each media within the ISE is created.

Table 5.2: Primitives for the Multimedia Presentation Agent

Primitive	Description	Parameters
pause	reserves a communication connection for the client application	MMObject *aMediaObject
stop	sets/requests the quality of service for the opened connection	MMObject *aMediaObject
play	gets the quality of service set for the opened connection	MMObject *aMediaObject
rewind	closes the communication connection	MMObject *aMediaObject
forward	reliable transmission of data, waits for an acknowledgement	MMObject *aMediaObject
scan	reliable reception of data, sends an acknowledgement	MMObject *aMediaObject, int nSegmentStep
recoverFromError	un-reliable transmission of data	MMObject *aMediaObject, int *nErrorCode
setQoS	un-reliable reception of data, may wait forever	MMObject *aMediaObject, int nParameterID, QoS *aQuality
getQoS	sends a segment to the receiver, may receive a negative acknowledgement for a segment which did not arrive at the receiver	MMObject *aMediaObject, int nParameterID, QoS *aQuality



Algorithm for Media Object Playback

caller - MediaObject = *the multimedia object which is currently being played back.*

```
while  ! (err = MediaObject -> open() )                    // open the source input stream
    if  ! MediaObject -> negotiateQoS( err )               // re-negotiate QoS
        return FAIL;

time_out = MediaObject -> getInformation() -> getQoS() -> Delay()  // get the start time/event
MediaObject -> synchronize( time_out ) ;                   // wait on start event.
current_segment = 0;                                       // initialize segment number

do                                                         // playback loop
   time_out = MediaObject -> getInformation() -> getQoS() -> Delay() //get the time out value – this value
                                                           //is updated in each blocking method
   if  ! ( err = MediaObject->read( &current_segment, &time_out ) )  // read the data from input stream
       if  ! MediaObject -> negotiateQoS(err)              // re-negotiate QoS
           return FAIL;
       else
           if  ! MediaObject -> recoverFromError(TIME_OUT)          // perform error recovery
   else
       if  ! MediaObject->synchronize( &time_out )         // inter & intra-media synch.
           if  ! MediaObject -> recoverFromError(TIME_OUT) // perform error recovery
               return FAIL;
   MediaObject -> display( OutputDeviceHandle )            // render the object
while err != DATA_END                                      // until no more data

MediaObject->close();                                      // close the connection
```

Figure 5.3: MPA Process Control Graph and Algorithm for Multimedia Object Playback

When a rendering process is initiated for a media object, the MPA invokes the **openConnection()** primitive of the transport manager to establish a communication channel for the media object. The connection is accepted if the communication server can negotiate an acceptable QoS request for the media being rendered. If the connection is refused, the MPA is notified which in turn informs the client application of the refused connection via the **recoverFromError()** primitive. If the connection is accepted, the process scheduler creates the playback process for the media object and puts this process on the blocked queue where it waits on its start event.

Once a start event has been signalled for a media object, the media enters the ready-to-run state. Media in this state are simply waiting for their time slice to begin their basic playback loop. An executing playback process first attempts to fetch its content part from the remote source via the network server. If the process has to wait for the data to be produced by the transport manager, the process is removed from the monitor and put on the blocked queue. This process will be awakened by either receiving the data it is waiting for, or by a time out event. The time out is set to the maximum delay time a process is willing to wait for its next data segment. For example, the time out for 30 frame per second VideoRAW objects is set to the inter-frame arrival time of 33 ms. Once the process resumes, it checks to see if it was awakened by a time out. If the process was awakened by a time out, the transport manager is informed of the missed deadline by the **recoverFromError()** primitive. This primitive informs the transport manager that it has failed to meet the requested quality of service for the rendering process. The MPA then attempts to re-negotiate the QoS of the rendering process or will abort the process if the re-negotiation fails.

When a process is awakened because the data has arrived it will attempt to "rendez-vous" with any other process (media) it is synchronized to. Processes remain blocked in the "rendez-vous" state until their inter-media synchronization conditions are satisfied. Once these conditions are satisfied, or if the process does not have any inter-media synchronization conditions, the process will remain blocked for the remainder of its time out period. The process exits the rendez-vous on two conditions: time out and failure. The time out condition is used to enforce the intra-media synchronization (i.e. constant playback for 8000 kHz audio) and to ensure that the media are not rendered faster than requested by the client application. The failure condition informs the client application that the media are out of sync. The synchronization algorithm used to implement rendez-vous is shown in figure 5.4.

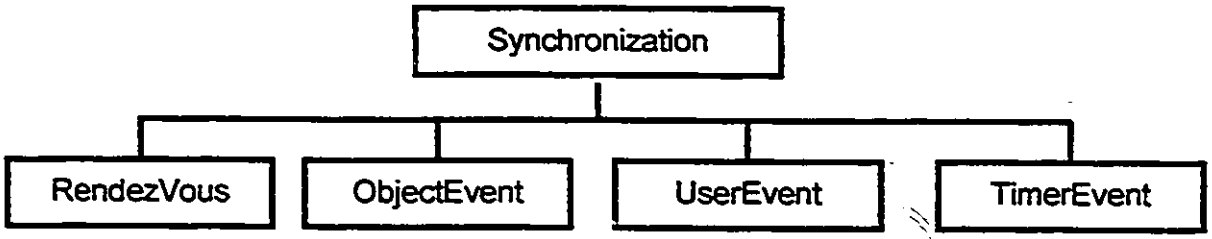
Synchronization Algorithm

caller – MediaObject = *the multimedia object which enters its synchronization state.*
input – *time_out = time left before the media object's synchronization delay has expired.*

```
// Get the list of synchronization links from the MediaObject's TemporalLayout member
pSyncCondList = MediaObject -> GetTemporalLayout() -> GetSyncConditions();

while ( pSyncCondList )                                    // loop for all sync links

    pSyncCondList -> SendSyncSignal()                      // send a sync signal to the link

    if  ! pSyncCondList -> WaitOnSyncSignal( &time_out ) // wait for reply from the link
        if ! MediaObject -> recoverFromError( TIME_OUT )// recover in case of time out
            return FAIL;

    pSyncCondList = pSyncCondList -> Next()                // get next link

end while

TimerEvent aTimer;
aTimer -> WaitOnSyncSignal ( &time_out );                  // wait for time_out

return OK;
```

Figure 5.4: Algorithm for Multimedia Object Synchronization

The MPA uses the inter-process communication technique to achieve inter-media synchronization and uses the event synchronization technique for intra-media synchronization. In order to implement these techniques a hierarchy of synchronization classes is created (figure 5.5) to represent the different types of synchronization required by the media. These synchronization classes make use of the Multi-Thread Library of the Solaris 2.3 operating system. The multi-thread library provides primitives which create threads of execution. These threads are termed light-weight processes because they carry less overhead than Unix processes created by **fork()** and **exec()** commands. Also, the multi-thread library provides simple interfaces for semaphores, mutex and condition variables.




```
class Synchronization  // public intterface for the Synchronization class
{
// Constructors – are used to create and destroy instances of the Synchronization class
public:
        Synchronization();
        virtual ~ Synchronization();

// Accessors  are used to get and set the data members of the Synchronization instance
        Link& GetLink();
        void SetLink( Link& pLink );

// Signalling  — Signals are loss less; i.e.  if a process is not waiting for the signal, the signal is still
registered by // that process, and acknowledged when the process invokes the waitOnSyncSignal method

        virtual int SendSignal ();                  // sends a synchronization signal to the link
        virtual int waitOnSignal( int &time_out);   // waits for a signal from the link until time_out expire

// Data members
protected:
        Link    *m_pLink;               // link to synchronize with
        UINT    m_nSignalCount;         // signal counter
};
```

Figure 5.5: Synchronization Class Hierarchy and Interface

The client application makes use of the media object's QoS parameter to recover failure conditions. For example, if a video frame failed to arrive in sync with its corresponding audio track, the client application could re-display the previous video frame with the current audio track and discard the video frame which is to arrive, or block the rendering of the current audio track until its corresponding video frame arrives. Here we use the QoS parameter of the media object in order to implement constraints on error recovery routines. These routines may be bound dynamically so that different error recovery strategies may be implemented as the rendering process proceeds. For example, an author may use the first strategy, frame repetition within a scene, but may also use the second strategy, suspend the rendering between scenes. Here we see how the QoS parameter can be used to create dynamic constraints to help create a satisfactory overall service to the user.

In the previous chapter we saw that the **forward()** and **rewind()** primitives relocate the relative temporal position of the multimedia document. The **scan()** primitive, on the other hand, permits the client application to render the document (forward or backward) at an accelerated rate. These primitives skip scheduled data segments, media objects or ISEs depending on the skip size defined by either the individual objects, the class defaults, or even client applications. For example, scanning a VideoRAW object could be accomplished by skipping data segments and displaying every 10th frame of the object. Because all media including the MEDIADOC document are derived from the MMObject class, polymorphism allows the client application to invoke any of these service primitives for any instance of a Group or other media class. Another advantage of this object-oriented data model, is that each class has its own way of implementing the service primitives of the MPA and therefore has its own default parameters for **forward()**,

rewind() and **scan()**. Also, when the objects are rendered, each has its own QoS member when negotiating data retrieval over the network or error recovery callbacks at the user interface. Therefore, re-negotiation can be handled automatically by incrementally adjusting the values within the default bounds defined by each class, or the bounds set by the author for each object within the document. Re-negotiation can even take place on the fly through dialogues with the user.

5.4 The Transport Manager

The transport manager handles the transfer of the multimedia information over the computer network. The goal of the transport manager is to be able to pass the information from source to sink according to the quality of service requested by the client. However, multimedia information is characterized by highly different service requirements. Therefore the transport manager must be able to offer a wide range of transfer rates, error rates, latencies and jitter.

The transmission of multimedia information will ideally be built on high performance networks which provide high bandwidth, low error rates, low latency and low jitter. These networks should also offer lightweight protocols and be able to sustain multicast communications [FUR 94]. Because multimedia communication protocols do not currently exist, an alternative solution is implemented for our DMIS. This solution attempts to implement a multimedia transport protocol which can be used by different network drivers and therefore remain independent of the communication hardware and software of the computer system. This protocol

also makes use of dynamic quality of service negotiation in order to optimize network resource utilization so that the maximum performance can be achieved from the particular communication environment.

The multimedia transport protocol is created on top of an existing transport protocol, specifically the user-datagram protocol (UDP). UDP is a transport service which is created on top of the Internet Protocol (IP). The reason the UDP protocol was selected for implementing the multimedia protocol of the transport manager is simply due to its wide use. Indeed, even ATM networks offer TCP/IP and UDP/IP emulation as an alternative to their native AAL1-5 protocols. Also, the ATM adaptation layers are still in their infancy with regards to available software APIs [MCD 95].

The primitives of the multimedia transport protocol are listed in table 5.3. These primitives are the primitives which are offered to the clients of the transport manager. The **openConnection()** primitive (figure 5.6) reserves a communication channel for the communicating client. When the client requests a quality of service for the channel (via the setQos primitive), the connection manager will determine if the transport manager can provide the service within the bounds of the DQoS parameter. If the transport manager cannot satisfy these bounds, the client may either re-negotiate or refuse (release) the communication channel. The connection manager is responsible for managing all communication channels. Because the manager "sees" all channels, it can improve the performance of any single channel by reducing the priority of the other channels. By doing this, the connection manager reduces the quality of the

other channels, yet it gains the ability to satisfy the service expectation of more client channels, albeit at the lower ends of their DQoS bounds.

Table 5.3: Primitives for the Multimedia Transport Protocol

Primitive	Description	Parameters
openConnection	reserves a communication connection for the client application	Address *destAddress
setQoS	sets/requests the quality of service for the opened connection	int nDeviceHandle, int nParameterID, QoS *pQuality
getQoS	gets the quality of service set for the opened connection	int nDeviceHandle, int nParameterID, QoS *pQuality
closeConnection	closes the communication connection	int nDeviceHandle,
sendMessage	reliable transmission of data, waits for an acknowledgement	int nDeviceHandle, char *msg, long size, int code
recvMessage	reliable reception of data, sends an acknowledgement	int nDeviceHandle, char *msg, long *size, int *code
sendData	un-reliable transmission of data	int nDeviceHandle, char *data long size, int options
recvData	un-reliable reception of data, may wait forever	int nDeviceHandle, char *data, long *size, int options
sendSegment	sends a segment to the receiver, may receive a negative acknowledgement for a segment which did not arrive at the receiver	int nDeviceHandle, char *seg, long size, int options, int nSegment
recvSegment	retrieves segments from the connection, optionally sends nacks on lost segments	int nDeviceHandle, char *seg, long *size, int *nSegment, TimeStruct *timeOut, int options

openConection() Algorithm

```
caller – MediaObject  = the multimedia object which wishes to create the connection to the remote server
input – destAddress  = pointer to address of the remote host/multimedia server

if   !(nComHandle = create_connection( destAddress ) )          // create the network connection
     return FAIL;                                               // cannot create connection

if   !connect( nComHandle, destAddress )                        // connect the handle to the address
          return FAIL;                                          // cannot make the connection

QoS  aQos = MediaObject->GetInformation()->GetQoS();            // get the QoS for the MM object

if   !setQoS( nComHandle, TRANSPORT, &aQos )                    // set the QoS for the connection
          // reset the QoS of the media object
     if   !MediaObject->GetInformation()->SetQoS( TRANSPORT, &aQoS )
          destroy_connection( nComHandle );                     // destroying the connection because
          return FAIL;                                          // we failed to get an acceptable QoS
                                                                // for data transmission

MediaObject->GetInformation()->GetOrigin()->SetHandle( nComHandle );
return  OK;
```

Figure 5.6: Algorithm to Open Connections for Multimedia Objects.

The **sendMessage()** and **recvMessage()** are reliable message transfer primitives. Because the multimedia protocol is built on top of UDP which is essentially an un-reliable transport service, a reliable message primitive had to be implemented so that communicating entities would not wait eternally for replies. This primitive uses a hand shaking protocol which re-transmits the message after a time out period until the sender receives a positive acknowledgment from the receiver. Figure 5.7 shows the implementation of the **sendMessage()** and **recvMessage()** primitives.

```
#include "mm_transport_primitives.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stropts.h>
```

```
#include <errno.h>
#include <sys/socket.h>
#include <netdb.h>

// sockfd - network connection socket
// code   - message code, i.e. OPEN, CLOSE, GET_SEGMENT, ...

char * recvMessage(int sockfd, char *msg, long *mlen, int *code)
{
  long new_msg;
  static long last_msg = -1;                          // used to keep track of messages
                                                      // received from the client so that
                                                      // re-transmissions are discarded.

  msg = mmdp_allocate_message_buffer(sockfd, msg, mlen);   // create message buffer

  do {
    mmdp_receive_data(sockfd, msg, mlen, code, &new_msg);  // get the next message from the queue
    if ( *code == OPEN )  last_msg = -1;              // if OPEN message, re-init  last_msg
  } while ( new_msg <= last_msg );                    // keep reading messages until we get
                                                      //  a new un-ACK'ed message
  last_msg = new_msg;                                 // update last received message
  mmdp_send_data(sockfd, NULL, 0, ACK, 0);            // ACK the message

  return msg;
}


long sendMessage(int sockfd, char *msg, long mlen, int code)
{
  int over = 0;
  int wait_time = 0;                                  // time to wait before re-transmission
  long size = 0;                                      // size of transmission

  char buf[1];                                        // used to receive ACK
  long bsize = 1;

  static long last_msg = -1;                          // used to keep track of messages
                                                      // received from the client so that
                                                      // re-transmissions are discarded


  struct pollfd pollfds;                              // pollfds structure is used to respond
  pollfds.events = POLLIN;                            // to the asynchronous input events
  pollfds.fd = sockfd;                                // coming over the communication
socket

  if ( code == OPEN ) last_msg = -1;                  // if OPEN message, re-init  last_msg

  last_msg++;
  do {                                                // transmission loop

    if ( poll(&pollfds, 1, wait_time) == -1 ) {       // poll the communcation socket
          perror("poll failed");
          exit(5);
```

```
    }

    switch( pollfds.revents ) {                                    // determine the type of event

        case POLLIN:                                                                    // received message
                mmdp_receive_data(sockfd, &buf, &bsize, &code, &retval);// r ead the message
                if ( code == ACK )  over = 1;                                           // check if it's an ACK
                break;                                                                  // if ACK done.

        case 0:                                                                         // timed out
                size = mmdp_send_data(sockfd, msg, mlen, code, last_msg);               // re-send the msg
                wait_time = DEFAULT_TRANS_TIMEOUT;                                      // reset the time out
                break;

        default:                                                                        // socket error
                perror("Poll returned error event");
                fprintf(stderr, "Returned error: %d\n", errno);
                exit(6);
                break;

    }

  } while ( !over );                        // loop until the message has been ACK'ed

  return size;
}
```

Figure 5.7: Implementation of the Transport Protocol Primitives for Sending and Receiving Messages

The **sendData()** and **recvData()** primitives on the other hand are un-reliable message services. These primitives are used in cases where re-transmission of information would impede the performance of the communication system. Consider for example the transmission of 30 frames per second video. Since the inter-frame arrival time is 33 ms, the re-transmission period would have to be less than 33 ms. In this case, re-transmission is not a reasonable solution since the sender will flood the network with many copies of the same frame until it receives an acknowledgement. Also, if we factor in the propagation delay and the reconstruction delay, it is very likely that the client will have exceeded the inter-frame arrival time if the first copy of the UDP message is lost.

The **sendSegment()** and **recvSegment()** primitives are designed specifically for multimedia streams like Audio, Video and Subtitles. The **recvSegment()** (figure 5.8) primitive attempts to read an expected, numbered segment. If the segment is not found on the read queue, a negative acknowledgment is sent to the sender, and the sender will re-transmit from that segment onward. Alternatively, the negative acknowledgement may not be sent by setting the SEND_NACK to false. Setting SEND_NACK to false is desired for the reasons explained earlier with regards to video transmissions. On the other hand, some communication channels may wish to send negative acknowledgements in order to guarantee low packet loss rates to the transmission client.

```
long recvSegment( int sockfd, char *seg, long *size, int *nSegment, TimeStruct *timeOut, int options)
{
  int code = 0;                                          // code of the incoming transmission
  long retval = NOT_RECEIVED;                            // return value
  long new_msg;
  struct pollfd pollfds;                                 // pollfds structure is used to respond
  pollfds.events = POLLIN;                               // to the asynchronous input events
  pollfds.fd = sockfd;                                   // coming over the comm. socket

  do {

    if ( poll(&pollfds, 1, updateTime(timeOut)) == -1 ) {    // poll the communcation socket
        perror("poll failed");
        exit(1);
    }

    switch( pollfds.revents ) {                                  // determine the type of event
        case 0:                                                  // TIME_OUT
          retval = TIMEOUT;
          break;

        case POLLIN:                                             //  data arriving on the socket
          mmdp_receive_data(sock, seg, size, &code, &new_msg);   // read the data
          if ( ( code == ACK ) || ( *sz <= 0 ) ) {               // is data valid
                if ( code != ACK )                               // if 0 sized message
                   mmdp_send_data( sock, NULL, 0, ACK, 0 );      // send ACK, i.e. no more segs
                * nSegment = new_msg;                            // update the segment number
                retval = DATAEND;                                //  no more segments
          } else {
            if ( new_msg == * nSegment) {                        // if the incoming segment is
              retval = new_msg;                                  // the one we expect, OK
            } else if ( new_msg > * nSegment) {                  // else if it is greater...
```

```
                if ( options &  SEND_NACK ) {                          // if SEND_NACK
                   mmdp_send_data(sock, NULL, 0, NACK, * nSegment);    // NACK the expected seg
                } else {                                               // else
                   retval = new_msg;                                   // accept the segment
                }
              }
            }
            break;

          default:
            break;
    }
  } while ( retval == NOT_RECEIVED );                                  // keep polling untill segment arrives or
                                                                       // timeOut gets triggered.
  return retval;
}
```

Figure 5.8: Implementation of the Transport Protocol Primitive for Receiving Segments

5.5 The Multimedia File Server

This section discusses the implementation of a multimedia file server which was used as the source of multimedia information for the DMIS. This file server is layered into two parts. The first part is the transaction manager and the second part contains the set of service primitives which are offered to the file server's client.

The transaction manager is responsible for coding and decoding messages going to and coming from the transport server. It is also responsible for handling the synchronized retrieval of multimedia information. The reason the transaction manager is treated separately is because the transaction manager is a component which may be used for either a database server or a multimedia file system. Therefore creating these new types of multimedia services requires only the replacement of the set of service primitives.

Table 5.4 lists the set of service primitives offered by the file server. These primitives establish an asymmetric service protocol with the client applications. This type of protocol is also called a client/server protocol. That is, the file server waits for a service request from the client. When the service request arrives at the communication endpoint of the server, the server accepts the message, spawns a new server to service this message and continues listening for incoming requests. Requests are not queued and processed sequentially because transactions involving multimedia information are typically long transactions which often require concurrent processing (i.e. audio/video playback). Sequential transaction processing is only suitable for transactions which have short service durations and do not have strict response times.

Table 5.4: Multimedia File Server Primitives

Primitive	Description	Parameters
open	opens the file	MMObject *pObject
close	closes the file	MMObject *pObject
load	loads the MMObject, i.e. retrieves the first segment of the MMObject	MMObject *pObject
save	saves the entire object to the opened file descriptor	MMObject *pObject
read	reads the entire object, a series of segments or a number of bytes from the opened file descriptor	MMObject *pObject, int firstSegment, int lastSegment,
write	writes the entire object, a series of segments or a number of bytes to the opened file descriptor	MMObject *pObject, int firstSegment, int lastSegment,
seek	positions the file descriptor to the position represented by the segment number of the offset number of bytes	MMObject *pObject, int nSegment OR long offset_bytes

Figure 5.9 shows the implementation of the transaction manager. First the communication endpoint is created so that the file server can listen for incoming requests. When the request is accepted, the new server is spawned in order to service the request. When creating a new server, a function table of primitives is loaded for the appropriate server. Therefore, the transaction manager can be re-used to implement different kinds of source servers by simply loading different function tables. In fact, one transaction manager per host can be used for running all the source servers, i.e. multimedia file and database servers, located on that host as long as the function primitives are defined for that source server.

```
/* Transaction Manager implementation */

#include "mm_transport_primitives.h"

#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>

int main( int argc, char *argv[] )
{
  int sockfd, newfd, clilen;
  struct sockaddr_in  cli_addr;                    // client address structure
  char serverName[64];                             // name of server which the client
                                                   // wants to connect to
   mmdp_init_manager();                            // initializes the transaction manager

        /* Open a UDP (Internet datagram) socket */
  if ( (sockfd = mmdp_create_server_socket()) < 0 )   // create the listening port
        exit(1);

  clilen = sizeof( cli_addr );
  while (1) {                                      // wait for connection requests

        // accept the client connection
    newfd = mmdp_accept(sockfd, (struct sockaddr *)&cli_addr, &clilen, serverName);

        // if the connection is accepted, create a server process to service the client
    if ( newfd > 0 ) spawn_server(sockfd, newfd, serverName);

  }
```

```
        /* Never Reached */
  mmdp_destroy_manager();                                   // on termination destroy the manager
  exit(0);
}


int spawn_server(int sockfd, int newfd, char *serverName)
{
  switch ( fork() ) {                   // Spawn new process
        case -1:                                // System Error

          perror( "Fork error." );  // Diagnostic message
          exit(1);
          break;

        case 0:  // Child process

                // Make sure SIGINT is set to DEFAULT action
                // mmdp_init_manager() overrides the SIGINT action.
          if ( signal(SIGINT, SIG_DFL) == SIG_ERR )
            perror("SIGINT error");

                // close parent sockfd in the child process and do service for the client.
          close(sockfd);
          break;

        default: // Parent process

                //  close child newfd and then go up and listen for more connection request
          close(newfd);
          return 0;
          break;
  }


  run_server(newfd, serverName);        // run the server
  exit(0);
}


int run_server( int sockfd, char *server )
{
  char *buf = (char *)NULL;                     // incoming message buffer
  int code;                                     // message code
  int retval = 0;                               // return value
  long nbytes = 0;                              // size of message

  struct pollfd pollfds;                        // pollfds structure is used to respond
                                                // to the asynchronous input events
                                                // coming over the communication socket

  extern int (*SourceServerFunctions[])();              // function table of the file server
```

```
extern void make_source_server_function_table();      // external function

make_source_server_function_table(server);            // load the function table

pollfds.events = POLLIN;
pollfds.fd = sockfd;

while (retval >= 0) {

  if ( poll(&pollfds, 1, -1) == -1 ) {                // poll the communcation socket, infinite wait time
       perror("poll failed");
       exit(5);
  }

switch( pollfds.revents ) {                           // determine the type of event

       default:                                               // error
            perror("Poll returned error event:");
            break;

       case 0:                                          // time out...should never occur
            retval = -1;
            break;

       case POLLIN:                                     // message arrived
                 // read the mesage
            buf = recvMessage(pollfds.fd, buf, &nbytes, &code);
                 // if valid message, look up the function from function table and invoke it
            if ( nbytes >= 0 )
              retval = SourceServerFunctions[code](pollfds.fd, nbytes, buf);
            break;

  }

}
return retval;                                        // Done
}
```

Figure 5.9: Implementation of the Transaction Manager

Once the function table is loaded, the message is interpreted and the server invokes the function from the function table which corresponds to the client's request. This server remains active until the client terminates the service request. The transaction manager offers the **setQoS()**, **getQoS()** and **recoverFromError()** primitives to the file server, because the transaction manager is the one responsible for controlling all the server processes which it has created. Consider for

example a retrieval process which is retrieving video frames from the file server. If the retrieval process misses its deadline because it cannot retrieve frames fast enough from the persistent store, the server process will invoke the **recoverFromError()** primitive. In response to a file server invoking the **recoverFromError()** primitive, the transaction manager can increase the resources allocated to the server process, i.e the memory buffers and CPU. Alternatively, the transaction manager may suspend lower priority servers which it has created, or even reset the service quality within the tolerable limits set in the client's quality of service parameter. For example, a client application may have requested 30 frames per second of video data, but may be willing to accept a lower limit of 10 frames per second. If the transaction manager cannot successfully re-negotiate the quality of service for the file server, that is if it cannot provide the lower limit client request of 10 frames per second requested for video data, the transaction manager will inform the client that it cannot satisfy the client's service expectations and it will terminate the transaction altogether. Figure 5.10 shows the algorithm used to implement the transaction manager's **recoverFromError()** primitive at it applies to missed deadlines by the file server.

Algorithm for Error Recovery at the Source Level

```
caller - MediaObject = the multimedia object being retrieved by the Multimedia File Server
input – error code

recoverFromError( int nErrorCode )                    // this function is used to dispatche
                                                      // the proper error handler only.
    ...
   if  ( nErrorCode == TIME_OUT )
       return  MediaObject->recoverFromTimeOut()      // error handler for TIME_OUTs

caller - MediaObject = the multimedia object being retrieved by the Multimedia File Server

recoverFromTimeOut()

  QoS aQoS;                                           // Get the Shared Buffers component
  MediaObject->getQos(ALL, &aQoS)                     // of the source server's QoS paramter
```

```
if buffersAboveHighWaterMark( aQoS->GetBuffers() )          // If the buffers are almost full
    if MediaObject->setQoS( INCREASE_BUFFER, NULL )         // add more buffers
        return OK                                           // continue

if !maxPriority( aQoS->GetBuffers() )                       // If the server's priority is not maximum
    if MediaObject->setQoS( INCREASE_PRIORITY, NULL )       // increase the priority
        return OK                                           // continue

  // The process table belongs to the transaction manager.
  // The process table is used to monitor all source server processes created by the transaction
manager.
processTable = GetProcessTable()                           // Get the process table

  // The transaction manager then attempts to suspend all processes which are suspendable. i.e
processes
  // of equal priority which are meeting their deadlines and may possibly have their priorities reduced.
if  suspend_processes ( processTable )
  return OK

if MediaObject -> negotiateQoS()                            // re-negotiate QoS of the media object
  return OK

return MediaObject -> TerminateTransaction()                // All's failed...notify client and stop service
```

Figure 5.10: Algorithm for Error Recovery at the Source Level

The inter-media synchronization technique implemented at the file server is the timeline technique explained in chapter 3. This technique is used because fine grained synchronization is not required at the source level. Because the media have to propagate over asynchronous computer networks, and because of the un-deterministic behaviour of these networks and their underlying protocols, it is more than likely that the synchronized media will arrive skewed at the client location. Ensuring the lip-synch quality of inter-media synchronization becomes the responsibility of the client MPA. Because the transport manager cannot guarantee the delivery of synchronized media, the IPC rendez-vous technique provides an un-necessary amount of computational overhead for source level synchronization. Therefore, the timeline technique offers a computationally economical solution which is capable of delivering the media to the transmission manager with an acceptable level of inter-media synchronization.

5.6 The User Interface and Mode of Operation

In this section, the document creation and document playback processes will be explained. The multimedia document is created with the help of the multimedia editor. This editor was implemented using the Simple User Interface Toolkit (SUIT ver 2.3) [PAU 91]. SUIT is a graphical user interface toolkit developed by the department of computer science at the University of Virginia. Suit provides a C subroutine library that allows programmers to create and modify graphical user interfaces interactively.

The multimedia editor which was developed is similar to many WYSIWYG graphical user interfaces which are widely available in text and image editors. This editor is made up of a menubar, a toolbox, scrollbars and a whiteboard or drawing area (figure 5.11). The whiteboard represents the actual spatial layout of the document.

In order to create a document, the author uses the toolbox to outline an area on the whiteboard. Once an area has been outlined for an object, the author can use simple point and click methods to cut, paste, move and re-size the object. By creating outlines and editing these outlines, the author is essentially creating the document's spatial layout which will be presented during playback.

The logical structure of the document is created via the "Paragraph" tool. This tool is used to group the media objects and/or other paragraphs, hence creating a hierarchical tree structure where the document is the root node and the leaf nodes are the basic media objects. Although the

tool is called "Paragraph", the object which is created resembles the Group objects explained earlier. The author may rename these paragraph objects either section, chapter, part,... or any other name which best describes the semantics of the grouping. The author moves objects into and out of the paragraph by the same point and click methods described for creating the spatial layout.

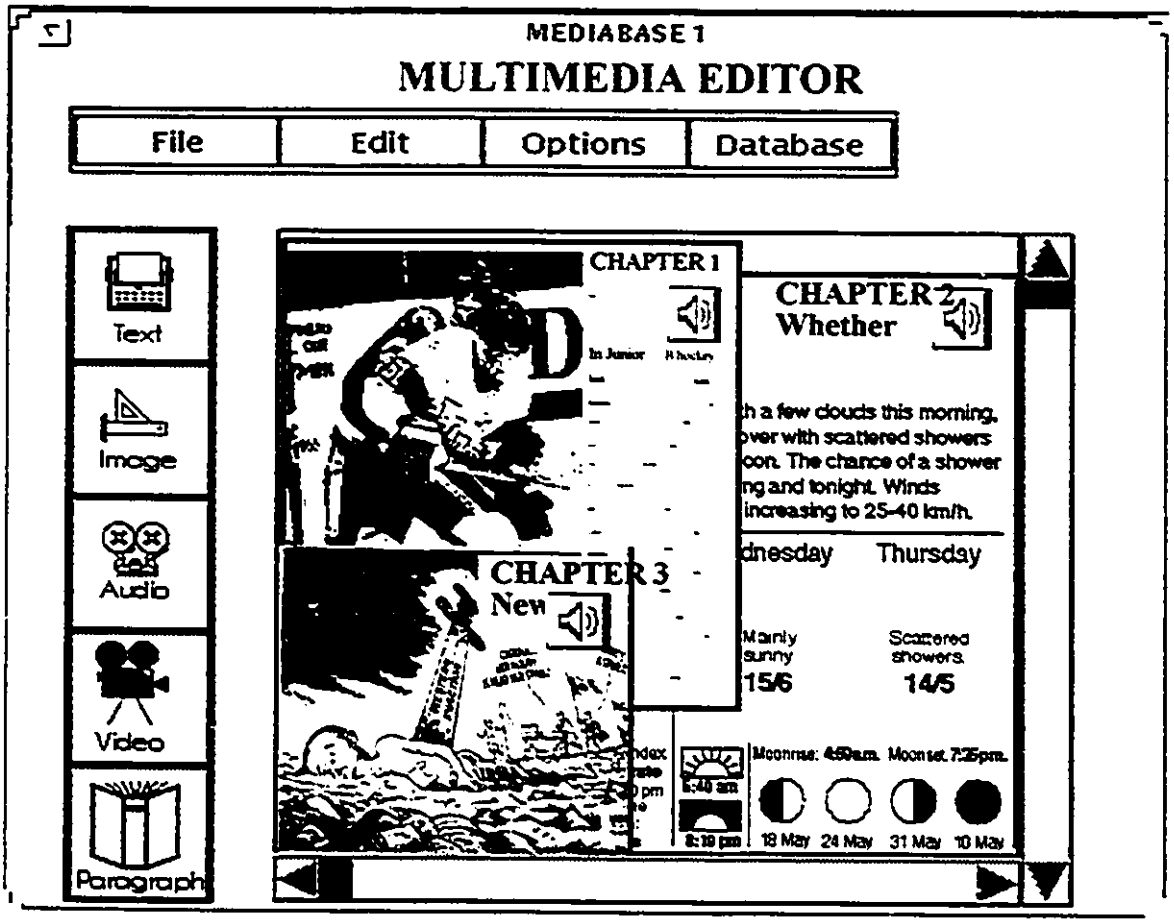


Figure 5.11: The Multimedia Editor

The Multimedia Editor uses dialogues to create the temporal layout. The dialogues are invoked by the Options command of the menubar. These dialogues prompt the author for the start

times and durations of the objects as well as the temporal links (if any) to any other object in the current document.

The content structure for the media object is added by using the File and Database functions of the menubar. When these functions are invoked, the author browses the database or file system for data or files of the same type as the object outlined. This data is first created using media editors (e.g. word processors, image editors, digital audio & video recorders,...) which make up the basic tools of a production server. The data can remain in the file system or be stored within the remote database. The author then selects the data or file desired for the outlined object which the Multimedia Editor will then load into it. This gives the author a preview of what the document will look like. The author can then modify the layout according to personal preferences. In the case of stream objects, only the first segment or frame is loaded into the object. Because audio objects have no visual information, an icon is used to represent audio objects, and therefore no data is loaded into the audio object.

The document now contains all the spatial and temporal information needed by the presentation agent to render the object, as well as the logical information required to identify and schedule ISE retrievals. The author now stores the document to the database or file, and the creation process is complete.

Figure 5.12 shows a client application, the Media Player application, which was created to test the synchronization system. Media Player was built using the Motif graphical API [HEL 91]. Media Player consists of a menubar, command bar, whiteboard, scrollbars and time bar. Again the whiteboard represents the actual spatial layout of the document while the scrollbars permit the

viewer to move about the spatial boundaries of the multimedia document. The time bar permits the user to jump to various temporal positions by sliding the indicator bar to relative positions in the document. Document rendering is controlled by the command bar.

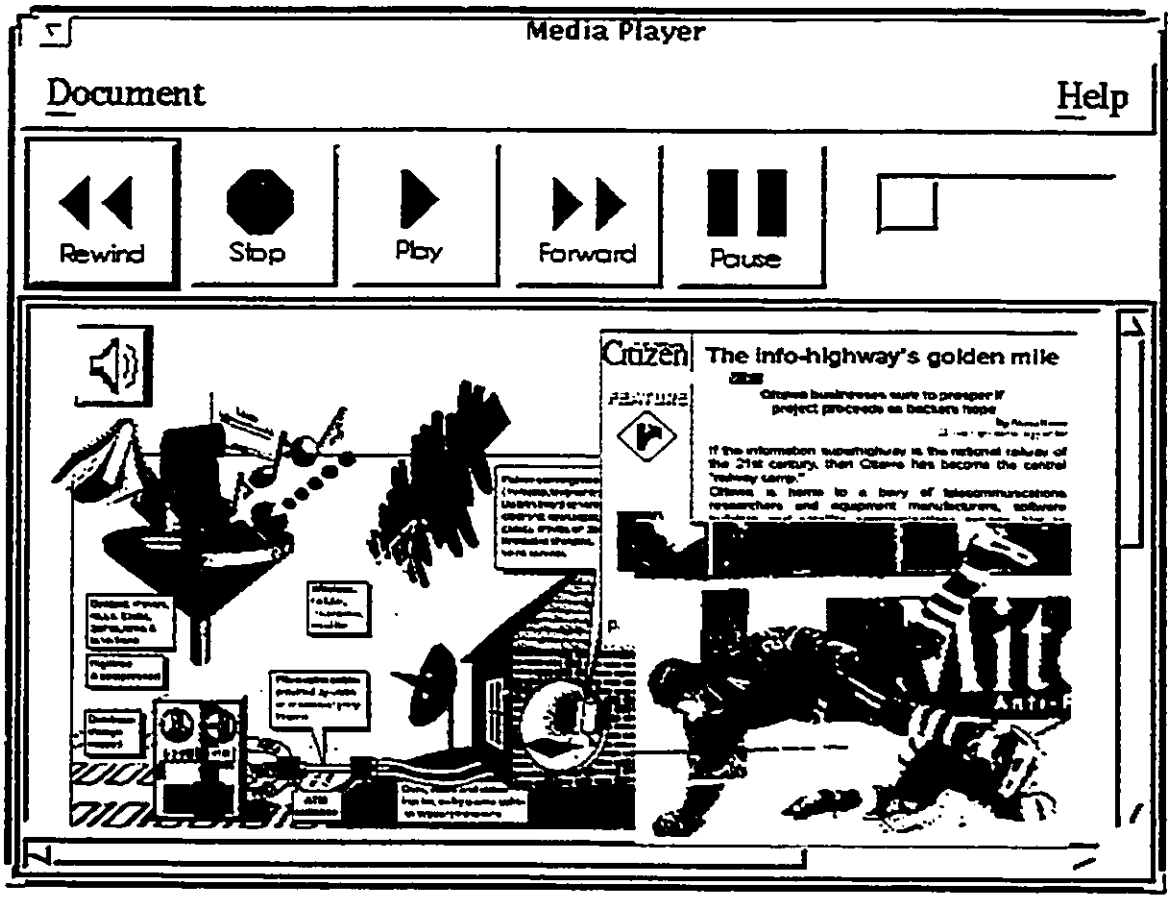


Figure 5.12: The Media Player

The command bar is the tool used to invoke the MPA primitives for the multimedia document. These primitives are rewind, stop, play, forward, pause and scan. The document is rendered once the viewer clicks on play. Play invokes the process scheduler which schedules the ISEs for playback, opens the connections to the remote file server and begins the transfer over the network as described earlier. The pause button suspends all communications and playback while the stop button terminates all network transfer and closes the connections.

The viewer can use the forward and reverse buttons to move to the desired temporal location of the document. A visual indication of the relative location of the document is echoed on the time bar. The viewer can also scan the document in either the forward or reverse direction by pressing the forward or rewind button while play is invoked. In this case, visual indication of the temporal motion is displayed on the whiteboard as the media objects change with time. Any of the functions described above can be invoked on individual media objects or groups by selecting individual objects (clicking on the object) and then clicking on the function button.

The viewer can use the forward and reverse buttons to move to the desired temporal location of the document. A visual indication of the relative location of the document is echoed on the time bar. The viewer can also scan the document in either the forward or reverse direction by pressing the forward or rewind button while play is invoked. In this case, visual indication of the temporal motion is displayed on the whiteboard as the media objects change with time. Any of the functions described above can be invoked on individual media objects or groups by selecting individual objects (clicking on the object) and then clicking on the function button.

Media Player can be used as a stand alone application. In this case, users load a multimedia document via the Open command of the File pull-down menu. The Open command pops up a file and directory dialogue which allows the used to select the file to be rendered. Alternatively, Media Player can be invoked from other client applications. These applications could be database clients, hypermedia browsers or multimedia news clients [FAL 95]. These client applications can retrieve the document without the content portion, and pass this light document as an argument to MediaPlayer application which will then be used to render the document for the client application.

5.7 Related Work

[LIT 90] have proposed the Object Composition Petri Net (OCPN) data model with an object retrieval and presentation algorithm as the basis for a presentation agent. The authors have identified certain problems with their data model. For instance, the OCPN model does not provide any spatial or layout support. Applications using this model must either specify where the objects will be displayed or let the window manager specify object positioning at the display. Another problem is that user input is not well described by the model and therefore user interaction with the multimedia document is not supported.

[OGU 90] discuss the RAVI system. This system is designed to represent the interchange of audio/visual interactive applications. RAVI provides a representation format and a set of functional operators which are used to control interchange of a multimedia application within a distributed environment. Although RAVI does integrate heterogeneous data from distributed sources, it does not control the arrival of the data from the distributed sources and therefore relies on the execution agent of each independent media stream to handle the media synchronization. Because each execution agent is proper to its own media stream, RAVI does not possess the means to enforce strict inter-media synchronization of independent data streams.

The MAEstro system presented in [DRA 93], attempts to create a user-friendly multimedia document authoring environment. It uses non-blocking remote procedure calls (RPCs) to create a control protocol for remote media editors (presentation agents). Because

of the latencies involved in the RPCs and setup time of the remote presentation agents, only coarse-grained synchronization (on the order of 1 second) is available in this system. Therefore MAEstro is not suited for lip-synching applications. Also MAEstro does not support interactive multimedia applications.

QuickTime [DRA 93, BUF 94] is a multimedia authoring and playback application developped by Apple Computer Inc. QuickTime uses the QuickTime Movie File (QMF) format for storing the content of the QuickTime presentation. The QFM is composed of a set of tracks, each track representing a specific, time-ordered media type, divided in segments of media clips. QuickTime synchronizes the presentatiion by aligning the media clips of each track and is able to keep in step with the real-time presentation by dropping frames (or segments) when necessary. QuickTime, however, does not support interactivity, nor does it provide dynamic QoS negotiation.

OMFI and MHEG [BUF 94] are two multimedia interchange formats which attempt to standardize the representation of time-based media. Like QuickTime, OMFI represents multimedia data using the track model paradigm. However, OMFI provides and extendible hierarchy which allows an application to define a new type of track. The disadvantage of OMFI is that it does not directly support real-time delivery over networks, nor does it support interactivity [BUF 94].

MHEG defines an object-oriented model for document presentation. MHEG includes an interaction model to support interactive multimedia documents. MHEG also supports real-

time network and continuous media synchronization. MHEG, however is a "final-form representation" [BUF 94]. That is, MHEG keeps the spatial and temporal relationships of the multimedia document, but does not keep the logical content of the document which may be needed for document editing, browsing and querying.

Chapter 6
Conclusion

6.1 Summary

In this thesis, the architecture and implementation of a synchronized delivery system for a distributed multimedia information system was presented. The aim of this development was to test the ideas and theories of multimedia temporal integration at both the data definition and data delivery levels. Five components were designed and implemented in order to realize this system. They are the multimedia object model, the multimedia presentation agent, the multimedia transport protocol, the multimedia source server and the multimedia editor.

The multimedia object model was created to implement the multimedia document based on the MEDIADOC architecture [EME 93]. This object model is an object-oriented design which is used to encapsulate the parts and behaviour for the multimedia document, the ISEs and also the individual media objects. This object model can be used to create complex document hierarchies and scenarios as well as provide support for interactive documents.

The multimedia presentation agent handles the synchronization at the local workstation. This presentation agent offers service primitives to client graphical user interfaces. These primitives control the rendering of the multimedia document as well as the quality of service requirements between the client application and the presentation agent. The MediaPlayer application was implemented as the client user interface for the multimedia presentation agent.

The transport manager is the component of the system which is responsible for the transfer of multimedia information over the computer network. This component manages the connection with respect to its bandwidth, delay, jitter, bit error and packet error rates. These communication parameters are communicated by the clients of the transport manager via dynamic quality of service management. The transport offers reliable positive acknowledgement, reliable negative acknowledgement and un-reliable data delivery primitives. The transport manager currently builds upon the UDP transport protocol.

The multimedia file server is the component of the system which is used as a source server for the distributed multimedia information system. The file server is built upon the UNIX file system. The file server makes use of the more generalized component, the transaction manager,

which handles message decoding and encoding, and is responsible for transaction scheduling and monitoring. The transaction manager can be used as the core component for database servers, production servers or any other type of server which responds to user queries or transactions.

The multimedia editor is the first implementation attempt for the DMIS which is presented in this thesis project. Although the multimedia editor is no longer used, many aspects of this editor provided the foundations from which the rest of the system is derived. The rendez-vous synchronization technique was initiated for this system and improved upon in the DMIS. The general multimedia object type from which other types could be sub-typed also began on this system. The multimedia editor provided some insight into how multimedia document editors could be designed so that complex document creation can be made more efficient.

6.2 Test Results

A number of tests were conducted in order to evaluate the performance of the MPA and the synchronized retrieval system. These tests were conducted for the source level, the presentation level and the transport level in order to determine which parts of the system functioned as expected and which parts constituted the weak points of the overall system.

At the source level, the first test was intended to determine if the local source could physically retrieve the multimedia information at the rates required by client applications. Therefore the tests conducted at this level were designed to measure the speed at which

applications could read data from the local disk and display this data at the local workstation. The first part of the test was constructed to measure the bit retrieval rate using the **read()** methods written for the media objects of the system. Because uncompressed video requires the highest throughput (~20 Mbps for 320x240, 8 bit colour) bit retrieval tests were conducted only for video objects. It was determined that the SPARC/10 IPC workstation could achieve bit retrieval rates of apporximately 40 Mbps second which easily satisfies a 30 frame per second retrieval rate required for video objects.

The next step is aimed at determining how many frames can be retrieved and consumed by the application. Although this test was not necessary to determine if the source system could provide satisfactory retrieval services for client applications it was useful in determining if the X11 graphical system would reduce the achievable frame rate. This test determined that only the disk retrieval rate limited the display of the video media and for tests where the video was stored in live-memory, between 60 and 70 frames per second were achievable for video objects.

The tests conducted at the presentation level were designed to verify the performance of the MPA. The main focus of these tests was to determine if lip-synch quality synchronization was being achieved by using the "rendez-vous" inter-media synchronization technique. In order to determine if lip-synching was achievable between an audio and video objects, two tests were conducted to ascertain if it were possible for one media to run-away from another. That is, if one media were artificially slowed down, and if the error recovery routines were set to suspend rendering processes waiting to rendez-vous; would a rendering

process continue its presentation even if it did not receive an acknowledgment from the process it is attempting to synchronize with?

In order to test this, we first slowed down the playback of the video object from 30 frames per second to 1 frame per second and set the error recovery routine to suspend the audio playback instead of dropping video frames. The second test consisted in doing the inverse, that is to slow down the audio playback rate from 8000 samples per second to 800 samples per second and to suspend the video playback instead of dropping audio samples. By slowing down an object's playback rate, the object would remain in its synchronize state much longer. Recall from section 5.3, that objects wait in the synchronize state until their time out value expires. This is intended to ensure that objects are not rendered faster than requested by the user, in this case artificially set to 10 frames per second and 800 samples per second. The results of these tests were as expected; that is the audio was rendered in bursts of samples followed by periods of silence. These bursts coincided with the display of a new video frame at the computer screen. A failure of either test would have been noticed by one process continuously playing at its defined playback rate while the other would play sporadically.

The next stage for testing the rendez-vous synchronization technique was similar to the first except that the number of media which we were attempting to synchronize with each other increased from 2 to 3, 4, and 5 simultaneous media objects. The additional media that were added were subtitles and video. Additional audio was not added since the purpose of the test was to evaluate if processes could be rendered simultaneously and because the audio

player can only be used by one process at a time. As before one media was slowed down well below its playback rate, and the rendering of the other media were observed. The tests were successful for 3, 4 and 5 simultaneous rendering processes and since there was no reason to believe that the system would fail as more media were added it was concluded that the rendez-vous synchronization process implemented for this system was able to ensure accurate and reliable inter-media synchronization.

The next stage in evaluating the performance of the synchronization system was to determine if the system could satisfy lip-synch quality synchronization. From the results of the previous tests we know that media which are synchronized together cannot run-away from each other. However, the rendez-vous synchronization scheme implemented in this system ensures that corresponding media segments are rendered simultaneously; it does not guarantee satisfactory lip-synching within the presentation time of the media segment. [STE 95] reports on a study conducted to evaluate the human perception of media synchronization. The researchers asked some 107 human subjects whether they could notice when the presentation of audio and video media were skewed and if so, was it tolerable? The results of the research determined that the subjects were willing to tolerate skews of up to ± 80 ms between the audio and video presentation. Therefore in order to guarantee lip-synch quality synchronization, it sufficed to ensure that inter-media synchronization points occurred at least at every 80 ms for synchronizing audio and video media.

In order to verify the performance of the transport system, tests were designed to determine if the transport manager was capable of retrieving the media information according

to the playback rates specified by the client application. Therefore tests were designed to use the retrieval method of a media object to count the number of time outs, that is the number of times segments failed to arrive by their specified deadlines, and also to count the number of re-transmissions for media with large time out values. At the time when the tests were conducted the ATM network was not fully functional; therefore all tests were conducted on the ethernet network.

The synchronized retrieval of a video object (160x120, 8 bit colour, 30 frames per second) was used to test the number of time outs encountered by the system. For some test trials, no time outs were encountered therefore demonstrating that the system seemed capable of retrieving the video object with absolutely no loss. On other trials, more than 90% of the video frames failed to arrive before their time out values expired. Because ethernet networks have unbounded medium access times, it is not surprising that such a high percentage of the video frames would miss their deadlines if the network should experience heavy traffic conditions. Also, since the ethernet network has a bandwidth of 10 Mbps, and since the video object requires only about 4.5 Mbps, it is also no surprise that it is possible to encounter no time outs under low (zero) traffic conditions.

In order to measure the number of re-transmissions required by the system, the same video object was used. This time however, we allowed frames to be re-transmitted and set the time out value to be a very large value such that time out should never occur. Because the ethernet network was a private local network, there was very little occasion for video frames to get lost within the network. Therefore the number of re-transmissions measured was

always 0. In order to test if the re-transmission system actually did work, we had to force the server to send the video frames at a speed which was much faster than the speed at which the client would read the video frames. By doing so, the client's socket receive buffer would get filled and force new video frames to be discarded at the receiving end of the system. As the client would read the sequential video frames from its socket receive buffer, it would find gaps where frames were discarded and the client would request the re-transmission of the video frame following the last in-sequence video frame.

From these tests it was determined that the ethernet network was the weakest point of the overall system. Because the multimedia transport protocol of the transport manager behaved as expected, a high performance network such as ATM would no doubt improve the performance of the overall system. The next shortcoming of the system was determined to be the multimedia file system. Although it was understood that the multimedia file system could handle high bandwidth retrieval, the test was conducted for only one 320 x 240 video object. The file system could not handle larger video objects or even multiple video object retrievals. The most succesful aspect of the system is the MPA's abitlity to handle lip-synch quality synchronization of multiple media objects using the rendez-vous synchronization technique.

6.3 Futur Improvements and Direction

This section offers a few suggestions on how the system implementation can be improved. These improvements are not necessarily aimed at improving the performance of the system per se but are also aimed at extending the capabilities of the current implementation.

The current object model currently provides direct support for handling distributed multimedia documents. The model can also be used to indirectly support distributed multimedia objects. That is, if a video object were fragmented within a distributed database, the current model can handle the temporal integration of these video fragments. However, this support is not direct. The current model would use a Group object to integrate the fragments of the entire video object by creating video objects whose data components are the distributed fragments and synchronizing their temporal presentations according to their video segment order. Although this may achieve the desired end result of integrating fragmented media, the meaning of the fragmented objects are replaced by the semantics of the group object. Therefore object manipulation may become ambiguous in large communication systems. Object-oriented systems are not simply about re-use, but also about the preservation of object semantics.

The MPA can be made more versatile by extending the types of media it can support. Animation types could be added, as well as format specific, text, audio, video and image types could be added. For example, only X11 and GIF images are currently supported. Classes which can be used to represent TIFF, CGM, postscript, JPEG, etc., should be added to offer more possibilities for multimedia document authors.

The transport manager can be improved by offering primitives which permit broadcast and multicast protocols. Although broadcast and multicast are available in the current system, this is an indirect consequence of the UDP protocol and may not offer the type of multicasting required for multimedia communication. The multicast primitives would offer more versatility to the communication system so that new applications may be created.

The most notable area where performance improvement may be achieved is in the implementation of the transport manager. The current implementation of the transport manager is based on the UDP protocol. The UDP and IP protocols have certain limitations which inhibit the performance of the transport manager. First of all, the maximum message size using the UDP protocol is 65 KB. This is obviously much too low for multimedia traffic. Chapter 2 outlined the fact that high quality images require between 64 KB and 75 MB. In order to communicate these images, the transport manager must handle its own message packetization and reconstruction and must use some of the data bits of the UDP message for its own reconstruction protocol. This wastes both CPU time and UDP message space. Therefore creating protocols tailored for these problems should improve the performance significantly.

Another problem with the UDP protocol is due to the medium access protocol used by IP. This protocol which was designed for Ethernet physical mediums has an infinite medium access time. Therefore, there is no way the transport manager can guarantee a minimum delay or even jitter. The fact that most LANs can generally offer reasonable delay and jitter is a happenstance of the communication environment which is guaranteed to fail as the level of network traffic increases to satisfy the commercial demand for multimedia communication. Protocols based on

the ATM AAL layers offer much promise to alleviate these problems. ATM also aims to support variable bit rate traffic which is not supported by Ethernet networks. The main problem with ATM AAL-5 which is the variable bit rate protocol is that it also has a maximum message size of 65 KB and therefore multimedia communication will have to rely on software solutions for message packetization and fragmentation. This will probably reduce the performance achievable by ATM protocols which attempt to offer many simultaneous connections like the system presented in this thesis.

Improvements at the source level can be two-fold. First more source servers should be created to offer more functionality to the DMIS. Production servers, database servers, source profilers, hypermedia browsers and improved file servers can be integrated into the current system to make it complete. The second way to improve the performance of the source system is to use multimedia storage systems as opposed to the operating system (UNIX, NFS) file systems. File systems like the MEDIAFILE file system proposed by [WAN 93] aim to improve the way continuous media are retrieved by using disk rotation speeds to decide on the optimal storage location for the data of the continuous media. Such systems should improve the performance of source servers which access persistent media.

A multimedia document editor based on the new object model should be created so that multimedia documents can be created efficiently. The old MediaEditor application is not based on the current object model and cannot be used for this purpose. Also the MediaEditor application is too simple and naive to be used for the creation of complex documents hierarchies and scenarios.

Appendix A

Publications

1. O. Megzari, M.B. Brahmanandam, J. Rody, G. Warnock and A. Karmouch, "MEDIABASE - An Experiment in Multimedia Information and Communication Systems", Broadband Islands '94: Connecting with the End-User, ed. W. Bauerfield, O. Spaniol and F. Williams, c. Elsevier Science B.V., pp. 435-438.

2. O. Megzari, M.B. Brahmanandam, J. Rody, G. Warnock and A. Karmouch, "MEDIABASE: A Distributed Multimedia Information System", Proceedings of the 17th Biennial Symposium on Communications, pp 223-226, Kingston, Ontario, CA, May 1994.

3. J.A. Rody, A. Karmouch, "A Remote Presentation for Multimedia Databases", Proceedings fo the International Conference on Multimedia Computing and Systems, pp 223-230, Washington DC, May 1995.

4. J.A. Rody, A. Karmouch, "A Presentation Agent for A Distributed Multimedia System Over High Speed Networks", IEEE International Conference on Communications, Seatle, Washington, pp 568-572, June 1995.

Bibliography

[BAS 94] F. Bastian, P. Lenders, "Media Synchronisation on Distributed Multimedia Systems", Proceedings of the IEEE International Conference on Multimedia Computing and Systems, pp. 526-531, Boston, MA, May 1994.

[BAT 94] J. Bates, J. Bacon, "A Development Platform for Multimedia Applications in a Distributed, ATM Network Environment", Proceedings of the IEEE International Conference on Multimedia Computing and Systems, pp. 154-163, Boston, MA, May 1994.

[BES 94] L. Besse, L. Dairaine, L. Fedaoui, W. Tawbi, K. Thai, "Towards an Architecture for Distributed Multimedia Applications Support", Proceedings of the IEEE International Conference on Multimedia Computing and Systems, pp. 164-172, Boston, MA, May 1994.

[BUF 94] J.F. Buford, C.B. Gopal, "Standardizing a Multimedia Interchange Format: A Comparaison of OMFI and MHEG", Proceedings of the IEEE International Conference on Multimedia Computing and Systems, pp. 463-472, Boston, MA, May 1994.

[BUL 91] D.C.A Bulterman and R. van Liere, "Multimedia Synchronization and Unix", Proceedings of the Second International Workshop on Network and Operating System Support for Digital Audio and Video, pp. 108-119, Heidelberg, Germany, Nov. 1991.

[CCI 88] CCITT Recommendation I.350, "General Aspects of Quality of Service and Network Performance in Digital Networks, Including ISDN", Blue Book, 1988.

[DRA 93] G.D. Drapeau, "Synchronization in the MAEstro Multimedia Authoring Environment", ACM Multimedia '93, pp. 331-339, June 1993.

[EHL 94] L. Ehley, B. Furht, and M. Ilyas, "Evaluation of Multimedia Synchronization TechniqueS", Proceedings of the IEEE International Conference on Multimedia Computing and Systems, pp. 514-519, Boston, MA, May 1994.

[ELL 91] C.A. Ellis, S.J. Gibbs and G.L. Rein, "Groupware: Some issues and experience", Communications of the ACM, Vol. 34, No. 1, pp. 39-58, January 1991.

[EME 93a] J. Emery and A. Karmouch, "A Multimedia Document Model for Continuous Media", CCECE/CCGEI '93, pp. 640-643, Vancouver, B.-C., CA, Sep. 1993.

[EME 93b] J. Emery and A. Karmouch, "A Multimedia Document Architecture and Rendering Synchronization Scheme", Proceedings of the Second International Conference on Broadband Islands, pp. 59-67, June 1993.

[FAL 95] B. Falchuk, "A Multimedia News Delivery System over an ATM Network", Proceedings of the International Conference on Multimedia Computing and Systems, pp 56-63, Washington DC, May 1995.

[FUR 94] B. Furht, "Multimedia Systems: An Overview", IEEE Multimedia Magazine, pp. 47-59, Spring 1994.

[HEH 90] D.B. Hehmann, M.G. Salmony and H.J. Stuttgen, "Transport Services for Multimedia Applications on Broadband Networks", Computer Communications, Vol. 13, No. 4, pp. 197-203, May 1990.

[HEL 91] D. Heller, "Motif Programming Manual for OSF/Motif Version 1.1", O'Reilly & Associates, Inc. 1991.

[HEP 94] F. Hepp, L. Rehbein, E. Hinostroza, E. Laval, C. Dreves, and M. Ripoll, "Enlaces: A Multimedia Based Educational Network", ACM Multimedia Systems 1994, pp. 329-336, San Francisco, CA, Oct 1994.

[HOD 89] M.E. Hodges, R.M. Sasnett, and M.S. Ackerman, "A Construction Set for Multimedia Applications", IEEE Software, pp. 37-43, Jan. 1989.

[HOD 90] M.E. Hodges, R.M. Sasnett, and Assoc., "Multimedia Computing: Case Studies from MIT Project Athena", Addison-Wesley Publishing Company. 1993.

[ISO] ISO, Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects (MHEG), Part 1, Draft International Standard 13522.

[ISO 88] ISO, Information Processing - Text and Office Systems. Office Document Architecture (ODA) and Interchange Format (ODIF). Parts 1-8, International Standard 8613, March 1988.

[KAR 90] A. Karmouch, L. Orozco-Barbosa, N.D. Georganas and M. Goldberg, "A Multimedia Medical Communications System", IEEE JSAC, Vol. 8, No. 3, pp. 325-339, April 1990.

[KAR 93] A. Karmouch, "Multimedia Distributed Cooperative System", Computer Communications, Vol. 16, No. 9, pp. 568-580, September 1993.

[LAR 94] P. Large, J. Beheshti, A. Breuleux, and A. Renaud, "The influence of Multimedia on Learning: A Cognitive Study", ACM Multimedia 1994, pp. 315-319, Oct 1994.

[LIT 90a] T.D.C. Little and A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects", IEEE JSAC, Vol. 8, No. 3, pp. 413-427, April 1990.

[LIT 90b] T.D.C. Little and A. Ghafoor, "Network Consideration for Distributed Multimedia Object Composition and Communication", IEEE Network Magazine, pp. 32-49, November 1990.

[LIT 91] T.D.C. Little and A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services", IEEE JSAC, Vol. 9, No. 9, pp. 1368-1382, December 1991.

[MCD 95] D.E. McDysan, D.L. Spohn, "ATM: Theory and Application", McGraw-Hill Series on Computer Communications, New York, NY. 1995.

[MEG 94a] O. Megzari, M.B. Brahmanandam, J. Rody, G. Warnock and A. Karmouch, "MEDIABASE - An Experiment in Multimedia Information and Communication Systems", Broadband Islands '94: Connecting with the End-User, ed. W. Bauerfield, O. Spaniol and F. Williams: C. Elsevier Science B.V., pp 435-438.

[MEG 94b] O. Megzari, M.B. Brahmanandam, J. Rody, G. Warnock and A. Karmouch, "MEDIABASE: A Distributed Multimedia Information System", Proceedings of the 17th Biennial Symposium on Communications, pp 223-226, Kingston, Ontario, CA, May 1994.

[MIL 93] G. Miller, G. Baber, M. Gilliland, "News On-Demand for Multimedia Networks", ACM Multimedia '93, pp. 383-392, June 1993.

[NEW 91] S.R. Newcomb, N.A. Kipp and V.T. Newcomb, "Hytime", Commununications of the ACM, Vol. 34, No. 11, pp. 67-83, November 1991.

[NIC 90] C. Nicolaou, "An Architecture for Real-time Multimedia Communication Systems", IEEE JAC, Vol. 8, No. 3, pp. 391-400, April 1990.

[OGU 90] F. Oguet, C. Schwartz, F. Kretz, and M. Quere, "RAVI, A Proposed Standard for the Interchange of Audio/Visual Interactive Applications", IEEE JSAC, Vol. 8, No. 3, pp. 428-436, April 1990.

[PAU 91] R. Paush, N.R. Young II, and R. Deline, "SUIT: The Pascal of User Interface Toolkits", Proc. of the ACM Symposium on User Interfaces Software and Technology, pp. 117-125, Hilton Head, South Carolina, USA, Nov 1991.

[PUG 94] J. Pugh, Lecture Notes for "Object-Oriented Systems: course 95.514". School of Computer Science, Carleton University. Winter 1994.

[RAV 93] K. Ravindran R. Steinmetz, "Transport-Level Abastracions for Multimedia Communications", Proceedings of the 4th IEEE ComSoc International Workshop on Multimedia Communications, pp. 220-231, Monterey, CA, USA, April 1993.

[REI 94] A. Reinhardt, "Managing the New Document", Byte Magazine, pp. 91-104, Aug. 1994.

[STE 90] R. Steinmetz, "Synchronization Properties in Multimedia Systems", IEEE JSAC, Vol. 8, No. 3, pp. 401-412, April 1990.

[STE 92] R. Steinmetz, T. Meyer, "Multimedia Synchronization Techniques: Experiences Based on Different System Structures", Proceedings of the 4th IEEE ComSoc International Workshop on Multimedia Communications, pp. 306-314, Monterey, CA, USA, April 1992.

[STEV 90] R. W. Stevens, "UNIX Network Programming", Prentice Hall Software Series, Englewood Cliffs, NJ, 1990.

[STEV 92] R. W. Stevens, "Advanced Programming in the UNIX Environment", Addison Wesley Professional Computing Series, 1992.

[TAW 93] W. Tawbi, L. Fedaoui, and E. Horlait, "Dynamic QoS Issues in Distributed Multimedia Systems", Proceedings of the Second International Conference on Broadband Islands, pp. 69-74, June 1993.

[WAL 91] J. Walrand, "Communication Networks: A First Course", Richard D. Irwin, Inc., Boston, MA, 1991.

[WAN 93] R. Wang, A. Karmouch, "A Multimedia File Structure For Continuous and Discrete Media", CCECE/CCGEI '93, pp 644-647, Vancouver, BC, CA, Sep. 1993.