National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre reference

Our file Notre reference

NOTICE

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

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

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

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

AVIS

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

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

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

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

Canada

Implementation and Performance Analysis of a Multimedia Synchronizer

Renaud Brimont

A thesis submitted to the

School of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE

Ottawa-Carleton Institute of Electrical Engineering

Department of Electrical Engineering

Faculty of Engineering

University of Ottawa

July, 1995

ISBN 0-612-07838-8

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Renaud Brimont

I further authorize the University of Ottawa to reproduce the thesis by photocopying or by other means, in total or in part, at the request of the other institutions or individuals for the purpose of scholarly research.

Renaud Brimont

Abstract

Synchronization is a crucial problem for distributed multimedia systems and has been the subject of a great many research investigations at all levels of a multimedia system, and on several of their components concerned with the issue of synchronization: the databases, the communication system, the operating system, the documents, applications themselves... Indeed, in the case of multimedia presentational applications, an efficient management of communication resources and the eventuality of having the data of different media types stored on distributed media-storing database servers requires the use of independent network connections for the transmission of each medium to a remote workstation. During a presentation, the multiple data streams have to be synchronized, using synchronization specifications for the multimedia document, and network traffic predictions, among others, in order to properly display that document and optimize resources such as the network connections and the buffers at the user's site.

In this thesis, the implementation of a complete software synchronization control system for presentational application is described. In the target application —a News on Demand Application, a user wants to retrieve a multimedia document from a distributed database. Each medium present in the article is stored independently from the others and retrieved by its own media server. The synchronization among the media is achieved in two steps. Using a pre-defined scenario and QoS guarantees supported by ATM based virtual connections, a delivery schedule is derived and passed to each server, so that they can transmit the appropriate media objects at the scheduled times, compensating for the delays and delay variations introduced by the network, and eventually the decoders. Additionally the Stream Synchronization Protocol (SSP) performs rescheduling operations at the client site to recover from jitters on the network and provide the user with a better and acceptable service. The various components (database, client application, graphical interface) of this system, their software modules, and their interactions are described thoroughly.

In parallel, we develop a DSPN model of the SSP, which is used to obtain representative results on the behavior of the system during a presentation such as the qualitative effect of the SSP, the number of synchronization errors, etc. These results can be used as general indicators for the design of presentational applications.

Acknowledgments

I would like to thank my supervisor, Professor Nicolas D. Georganas, for his guidance, support and encouragement throughout my studies at the University of Ottawa, and Dr. Tet Yeap, who critically examined parts of my research work.

I also gratefully acknowledge the financial support of the Canadian Institute for Telecommunication Research, the University of Ottawa, and Telecom Paris.

While many people in the past and present MCRLab research team deserve my gratitude, I extend special thanks to those who collaborated in the development of the Multimedia News on Demand prototype, especially Jeff Brinskelle, Grant Henderson, and Louise Lamont, with whom I learned so much.

I am also grateful to Professor Ionescu for allowing me to use his computer facilities, together with Professor A. Karmouch, for their words of encouragement.

Finally, my sincere thanks go to my family, as well as Mr. and Mrs. Heath, for their strong support at the toughest times of this thesis, and to my fiancé, Tanya, for always being there and believing in me.

Table of Contents

List of Figures and Tables

Acronyms

ATM	Asynchronous Transfer Mode
B_ISDN	Broadband Integrated Services Digital Network
CITR	Canadian Institute for Telecommunications Research
CMSC	Client Media Synchronization Controller
CSCW	Computer Supported Cooperative Work
DSPN	Deterministic and Stochastic Petri Net
FDDI	Fibre Distributed Data Interface
FIFO	First In First Out
GUI	Graphical User Interface
I/O	Input/Output
IPC	Inter-Process Communication
LAN	Local Area Network
MAN	Metropolitan Area Network
MPEG	Moving Pictures Expert Group
MSC	Media Synchronization Controller
PN	Petri Net
QoS	Quality of Service
RPC	Remote Procedure Call
SMSC	Server Media Synchronization Controller
SSP	Stream Synchronization Controller
TFG	Time Flow Graph
VCR	Video Cassette Recorder
LDU	Logical Data Unit
ODA	Open Document Architecture

Chapter 1

Introduction

1.1 Multimedia Systems and Applications

The advent of multimedia technology in the last five to ten years represents a major step forward for our computer based informational society. Not only will it improve dramatically existing systems, but it sets a new framework in which a wide range of potential applications are made attainable by integrating a variety of information formats, also called media, such as voice, graphics, animation, images, audio, and video. Nowadays, as this technology is maturing, multimedia systems are becoming technically, but also economically, feasible.

Computer technology has enormously improved, with the development of very powerful workstations dedicated to multimedia, storage devices that can handle hundreds of gigabytes with fast access times and high transfer rates [Fur94], and compression techniques that reduce the vast amount of data related to such media as audio, video and images. This leads to the realization of a number of successful prototypes of multimedia systems operating on a single workstation. Some of these systems are now commercially available [Bla91]. On the other hand, networking and communication technology has known dramatic advances too. For example, high speed integrated

services digital networks operating at more than hundred Mbits/s over long distances such as FDDI and B-ISDNs based on the ATM technology [Leo94]. Despite breakthroughs in the domain of high performance multi-service networks, distributed multimedia applications, which additionally involve the distribution of a system over a digital network, are not as advanced as stand-alone systems. Distributed systems offer much more promise and perspective than the latter, but they are more complex and introduce many specific problems which are currently the focus of many research teams.

Many multimedia systems are already available commercially. They are multimedia mailing systems, computer supported collaborative work systems (CSCW), and multimedia conferencing systems. Many applications in the fields of tele-education, travel, real-estate, banking, tele-medicine, advertising, etc., are now emerging [IEEE92]. One area of active research is the development of presentational applications, and especially on-demand multimedia services [Ran92, Lam94.6]: video rental services, digital multimedia libraries, such as the Electronic Library of the Future, interactive entertainment or multimedia news services [Fur94]. Many business alliances between entertainment, cable, phone, and computer companies have been formed targeting especially the soon-to-come market of video on-demand. Others, such as broadcasting, newspaper or business companies, are looking into general or specific news on-demand systems [Mil93].

One of the main problems in the design of a distributed multimedia system is to integrate the different data types stored in distributed multimedia database servers, and sent in parallel on multiple virtual connections. The need for this integration is a consequence of timely dependencies between the information coded in the different media objects. When a composite multimedia document is created, each of its components is occupying a defined place in the 'space' represented by the screen of the computer (the layout relationships), and each component must be presented at a certain time according to its temporal relations to other components. To reconstruct the multimedia document after its transmission over the network requires proper spatial and temporal integrations. The latter is also referred to as media

synchronization.

1.2 The Multimedia Synchronization Problems

The problems of synchronization of different media elements arise from the distribution of the servers over a network that requires the data belonging to different media to be transmitted on different virtual data connections. This problem is made worse by the necessity to transmit the data according to their traffic characteristics and Quality of Service (QoS) requirements, even when the data could be multiplexed on a single channel and thus synchronized at the source. The broadband network supporting the data retrieval will introduce random delays and delay jitters during the transmission. Data that were scheduled to arrive simultaneously at the receiver's workstation are thus likely to arrive at different times, very often introducing a loss of synchronization in the display at the receiver's workstation.

A multimedia system supports different types of media. A distinction is made between time-dependent and time-independent media. A time-dependent media object will usually be presented as a sequence of presentation units: in a video clip, for example, a sequence of frames will be presented frame by frame at a certain frame rate. In contrast, a time-independent media object will be presented as one presentation unit, e.g., a page of text or a graphic [Bla96].

Synchronization can be divided into two different classes: *intra-media* (or *serial*) synchronization, and *inter-media* (or *parallel*) synchronization. Intra-media synchronization takes place within a time-dependent media stream, and it will determine the rate at which events must occur within a single data stream. Inter-media synchronization on the other hand determines the relative schedule of separate streams, and will take place between either time-dependent or time-independent media. The inter-media synchronization can be further classified into *continuous* and *point to point* synchronization. *Continuous* synchronization corresponds to the permanent synchronization of lengthy events belonging to time-dependent media streams,

and *point to point* synchronization is the synchronization at a specific time between two events (e.g., the presentation of two media objects) [Fur94].

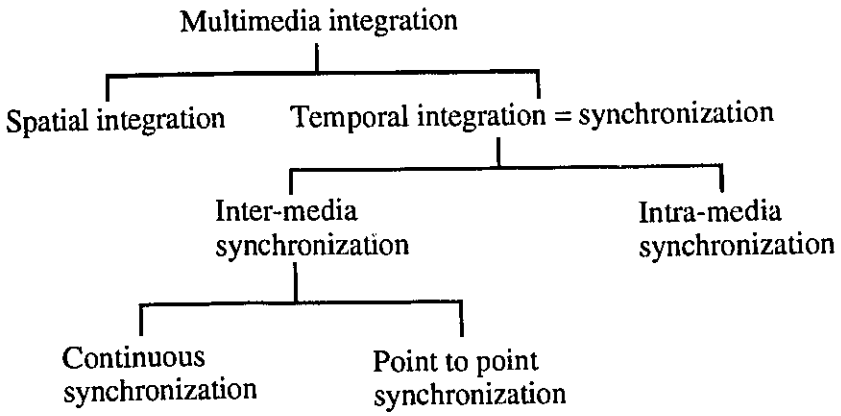

Figure 1.2 Multimedia integration

There are three major synchronization issues to be solved in a multimedia system: the temporal modeling of synchronization constraints during a multimedia presentation, the algorithms performing the synchronization based on the temporal model, and the extension of transport protocols, with QoS guarantees, for transferring multimedia data while taking into account their synchronization requirements.

The research at the Multimedia Communications Research Lab. (MCRLab) between 1990 and 1993 covered all of the synchronization aspects mentioned above [LiLi92-LiLi94]. Later studies in the case of multimedia presentational application have been conducted [Lam94.5, Lam 94.6], which present synchronization architectures and protocols for this particular case. [Lam94.6] targets more specifically at a Multimedia News On-Demand Service application. These studies on presentational applications were developed in relation to a major Canadian Institute for Telecommunications Research (CITR) research project on "Broadband Services". The work presented in this thesis is a continuation of these studies and is a part of this CITR project. The architecture and protocols presented in [Lam94.5] have been used in the

implementation of a prototype for Multimedia News On-demand application.

1.3 A Multimedia News On-Demand Prototype

The CITR "Broadband Services" Project

Through one of its major project, denominated "Broadband Services", the CITR is intending to "develop a thorough understanding of the hardware, software, networking and database issues related to the development of efficient and flexible multimedia presentational applications". The result of this research is expected to have an impact on the design of video servers, multimedia databases and telecommunication softwares in the near future [Dra94]. The project especially puts a strong emphasis on the development and implementation of an experimental common prototype (see Figure 1.3.2 which describes the rough architecture of this prototype) , which is to be demonstrated on an ATM test bed. The implementation and tests on the ATM network are very important, in that they will help us to fully understand various research issues where models are non-existent or not always reliable. The target application being chosen is an on-demand multimedia news service, which will allow users to browse through items of current-affairs stored in digital form in a remote multimedia database with distributed servers. Users will be able to interactively retrieve documents coupling newspapers articles, radio and TV news, etc.

This major project has been split into six complementary constituent projects, each of which is being investigated by a different university research team. A brief overview of each of the constituent projects is given hereafter, with the project's name and the related university teams [CITR 1994]:

- i) Multimedia Data Management, University of Alberta (Prof. T. Oszu).
- ii) Scalable Video Encoding for Database Storage, INRS-Telecommunications (Prof. E. Dubois).

- iii) Quality of Service Negotiation and Adaptation, University of Montreal (Prof. G. Bochmann).
- iv) Distributed Continuous-Media File System, University of British Columbia (Prof. G. Neufeld).
- v) Synchronization of Multimedia Data for Presentational Applications, University of Ottawa (Prof. N. D. Georganas)
- vi) Project Integration, University of Waterloo (Prof. J. W. Wong)

Projects i) and iv) examine database issues in relation to storing and retrieving the media involved in a presentational application over a high-speed B-ISDN. Project i) focuses rather on characteristics and requirements of a multimedia database due to the nature of the information to be stored (e.g. benefits of an object management system over a relational database), while project iv) looks into developing a file system that supports the time-sensitive nature of multimedia data (video and audio), such as a very high aggregate throughput, guaranteed real-time rate-based quality of service despite a large number of users... Project ii) is concerned with the flexible representation of compressed still pictures and video sequences in a multimedia database, thus taking into the fact the user's terminal capabilities (e.g. resolution). Project iii) is studying methods for QoS management such as negotiation, re-negotiation, and monitoring within a distributed environment, over networks such as ATM. The methods should handle the QoS violations and give the application the means to dynamically adapt to a variety of QoS. Project v) is handled by our research group at the MCRLab. Its goal is to develop a complete software control system for synchronizing the multimedia objects, originating from distributed media-servers of a multimedia database. Finally, project vi) is coordinating the other projects so that the shared goals of the major project are eventually achieved.

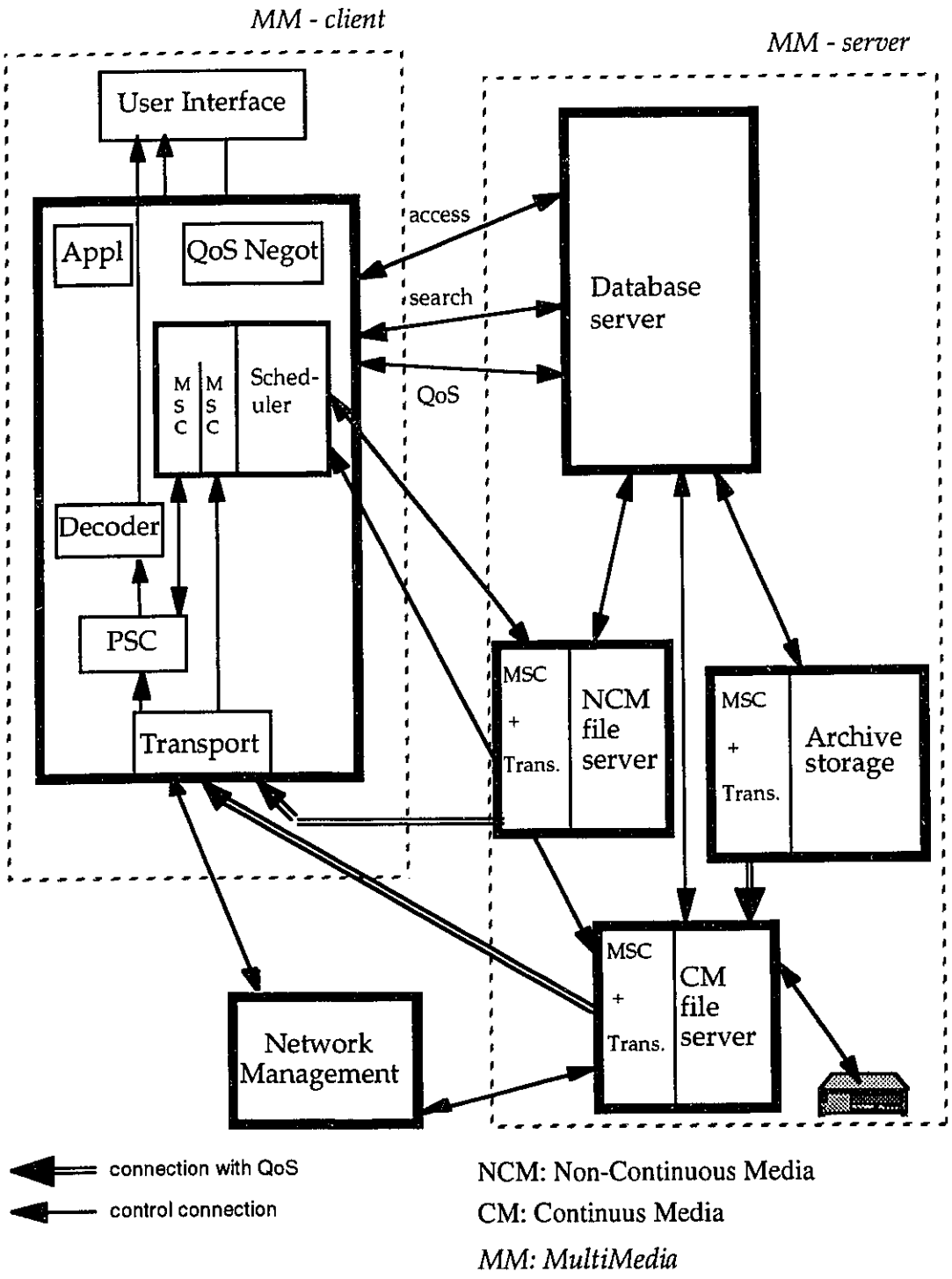

Figure 1.3.1 General architecture of the integrated prototype

The MCRLab Prototype

The final goal of this major CITR project was the development of a multimedia presentational application. The target application that was finally chosen is a Multimedia News On-Demand service, and the six university teams have been collaborating to implement a common prototype of such a system. A first integrated version of this prototype has been demonstrated in March 1995 on a local ATM network. A later version will be demonstrated in August 1995 on an ATM test bed.

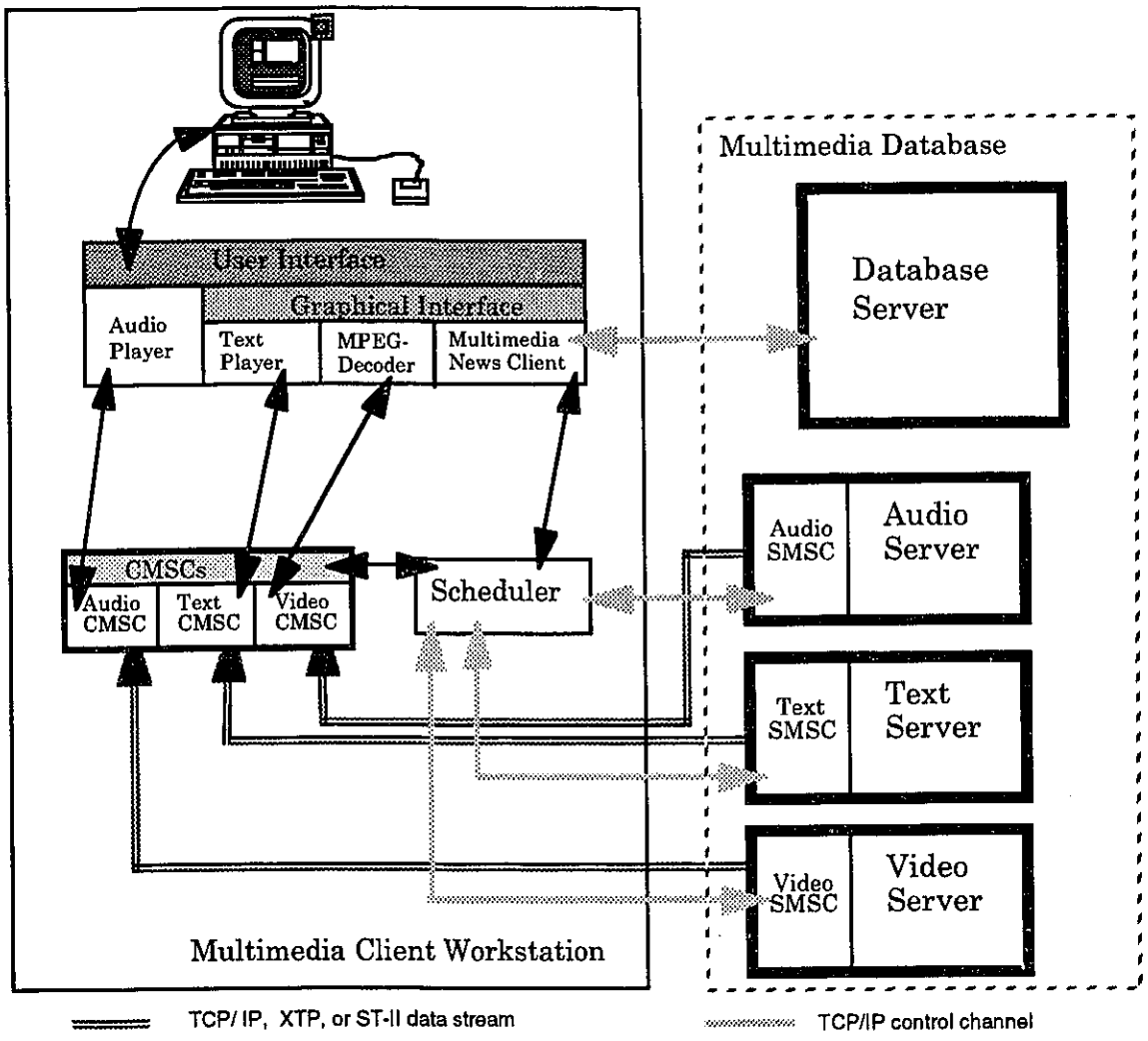

Figure 1.3.2 Structure of the 'local' Multimedia News On-Demand prototype

Though our task was limited to the implementation of a software control system for media synchronization, and to later integrate it into the common prototype built by the six university teams, we found it necessary to develop a 'local' prototype. This allowed us to better appreciate certain implementation issues. This prototype will also be used for testing purposes on the OCRINet, the first wide-area ATM test bed in Canada, sponsored by the Ottawa Carleton Research Institute (OCRI) and the industry. In this prototype, we focused on the synchronization control components. Many other components, such as the database server, the media file servers or the GUI were kept as simple as possible, since they may not be used in the integrated version, and their are not affecting the synchronization performance of the all system. Figure 1.3.2 shows the simplified architecture of the 'local' prototype and its components.

In any system design, it is necessary and very important to have an evaluation phase. The performance evaluation of a system can be conducted either through the use of a prototype of the system once it has been build, or through a model of the system. In performance modeling and evaluation, a model is build that reflects the characteristics and behavior of the studied system. The model is used afterwards to obtain performance results through simulations or mathematical analysis. The results provide us with estimates of the behavior of the real system, and the closer the model is to the real system, the better the estimates. Performance modeling is useful to predict the performance of a system before it is build and thus influence its design to meet its various requirements. But modeling, even when a system's prototype is available, is very often the best way to understand a system's behavior; it is the best suited mean to determine the influence of certain parameters on its performance. It is in this frame of mind that a performance analysis of the Stream Synchronization Protocol has been conducted.

1.4 Thesis Outline

The thesis is organized as follows:

Chapter 2 presents a general review on multimedia systems and the multimedia synchronization technology. The general requirements of multimedia information systems are assessed. The specific constraints that media synchronization imposes on such systems is explicitly detailed. Various methods for specifying the synchronization requirements are presented. Finally, we explain the approach taken in different multimedia systems to achieve the required synchronization.

Chapter 3 describes the implementation of a prototype of a media synchronization control architecture for a Multimedia News On-Demand system. The concept of Multimedia News On-Demand service is explained. The design of the synchronization control systems is presented. The rest of the chapter is devoted to giving a thorough description of the prototype's implementation: the various components of the prototype (servers, client application, graphical user interface) and their functionality are described; specific implementation issues (inter-process communication, client-server architecture, buffer management, and user interactions) are explicitly detailed.

Chapter 4 focuses on the performance of the Stream Synchronization Protocol (SSP), which performs the recovery of asynchrony between media during the presentation of a document. A short review on Petri Nets is done, before a model for the SSP is build using Deterministic and Stochastic Petri Nets (DSPNs). Performance results obtained through simulations are later discussed.

The thesis ends with chapter 5 which concludes on the prototype's implementation and the performance analysis that have been conducted. Suggestions for improvements and future work are made.

1.5 Main Contributions and Publications

Main Contributions

The main contributions of this thesis are summarized below:

- In chapter 3, which describes the implementation of the media synchronization architecture, section 3.3 to 3.6 contain original contributions of the author, either exclusive (section 3.5) or important (sections 3.3, 3.4 and 3.6).
- In chapter 4, the modeling of the SSP (section 4.2), the extension of the notion of buffering delay (section 4.4) and the simulation results (section 4.5) are all original contributions of the author.

Publication Arising from this Research

L. Lamont, Lian Li, Renaud Brimont, and N. D. Georganas, "Synchronization of Multimedia Data for a Multimedia News on Demand Application," *IEEE JSAC*, Jan. 1996 (to appear).

Chapter 2

Multimedia Systems and Media Synchronization Issues

Technological advances have made storage and communication of image, audio, and video information possible in the last few years. The processing of all these different forms of information using digital technology allows us to bring these disparate but complementary sources to one unique multimedia platform. We have come a long way since the time computing was only handling alphanumeric information, and now, in all the aspects of computing, from networking to user interfaces, the multiplicity of representation of information is addressed, making computers a transparent tool to the user [Jai94].

However, to get a multimedia environment, one needs more than just being able to handle a large variety of media to convey information. The various media must be *integrated* in a single framework, and, during the integration process, the characteristics of each medium must be addressed and used in a multimedia system to optimize its transparency and its efficiency. This leads to a number of requirements for multimedia systems, that intervene both at the computer

and the communication levels. Apart from the integration requirement, we can name the other problems of real-time performance, dynamic and flexible management, group communication handling, and synchronization, each of which will be shortly presented in section 2.1. The next section describes the requirements imposed in term of synchronization during the presentation of multimedia data. Finally sections 2.3 through 2.5 respectively describe various models for describing the synchronization specifications, review different approaches for transporting the synchronization specifications to the end-user system, and detail two of the many steps where synchronization control operations are necessary, between time of the acquisition of the data and the time they are displayed.

2.1 Requirements of Multimedia Systems

2.1.1 Integration

As discussed previously, the term multimedia covers a large number of data types. A certain variety already exists, but new data types will appear in the near future which multimedia systems should be able to deal with. Each medium presents certain characteristics that impose specific constraints on the processing, presentation, and transmission devices. Time-dependent media obviously have different requirements than time-independent media. Their integration is indeed an essential and new aspect in information processing, and it represents the main challenge for multimedia systems.

The constraints imposed on a system are not only medium dependent but also application dependent. Each data type will accept different quality of service from different applications. And if we want to use multimedia systems at full efficiency, we need to integrate the different media and their qualities of services into a unique framework. Integrating the media means that the different media types remain independent but they can be processed and presented together. A definite step toward higher degrees of integration is brought by integrated digital systems, since

they allow to process any type of information in digital form.

2.1.2 Real Time Performance

Another essential requirement of multimedia systems is the need for real time performance to be able to handle continuous data types such as video or audio. The problems linked to real time performance are numerous, challenging, and they are stressing all the components of a computer system. Continuous media necessitate a very high *storage* capacity, very fast access times and high transfer rates for database servers. New file formats need to be created within multimedia file systems to optimize the access times and take into account the time-sensitive nature of some multimedia data. Very high speed networks and protocols are necessary, that provide guaranteed QoS parameters, such as high bandwidth, low latency, and low jitter required for the *transmission* of multimedia data. Compression algorithms need to be further developed, that reduce enormously the required bandwidth for continuous media, thus reducing the load of the network. A large *processing* power is then necessary to implement software codecs able to code or decode the data streams in real time. The handling of such media also requires the operating system to support real time scheduling, deterministic delays and fast-interrupt processing, and their presentation imposes an architecture with efficient I/O and high bandwidth bus.

2.1.3 Dynamic and Flexible Management

In a multimedia system that transfers data over a network, the system management has to allocate the requested resources, at call setup, for the media transmission. The resources are allocated according to a number of QoS criteria, and the objective is to keep a sufficient transmission QoS to maintain the most stringent reception QoS requested by the user and the application. Dynamic and flexible management is necessary to optimize the available resources. Re-negotiation of the transmission QoS might be required when new connection requests are made. A reduced service

can be provided to applications that are willing to accept a certain degradation. Resources allocated to requests with low priority can be preempted. The major requirement of a multimedia system is that the agreed QoS must be maintained to keep the quality of the presentation to the end-users at any moment.

2.1.4 New Specific Application Demands

Multimedia applications bring new and specific demands on the underlying technology. Among them, two key areas require a specific system support: *groups* and *synchronization*.

Many target multimedia applications stretch out the importance of groups: Computer Supported Cooperative Work (CSCW) is one of them. In some multimedia systems, the same multimedia information needs to be sent simultaneously to many different users, which requires multicast facilities to be implemented, even for handling of continuous data. Another issue with groups is the need for coordination, e.g., the access to the shared material in a group conference has to be controlled to maintain the integrity of the information for every participant.

The requirement for multimedia synchronization, on the other hand, arises from the handling of multimedia data with different QoS requirements, and from the eventual distribution of multimedia systems which implies that many data streams have to be handled in parallel, even though they have to be presented with precise relationships towards each other. This implies that synchronization constraints must be expressed across different media types, and these need to be enforced with the use of appropriate synchronization control algorithms. These specific aspects of media synchronization will be discussed in details in sections 2.3 to 2.5.

2.2 Presentation Requirements

When one designs a system, the concern that should be placed above any other is the final use that the users will make of the system. In the case of a multimedia system, synchronization is

the main concern because a loss of synchronization during the presentation to user can degrade the retrieved multimedia information to the point where the user is unable to appreciate any information from any media. Figure 2.2.1 [Lit90.11] gives, as an example, a high level view of a video telephony system and shows where asynchrony can be introduced between the sources of the data and the display devices.

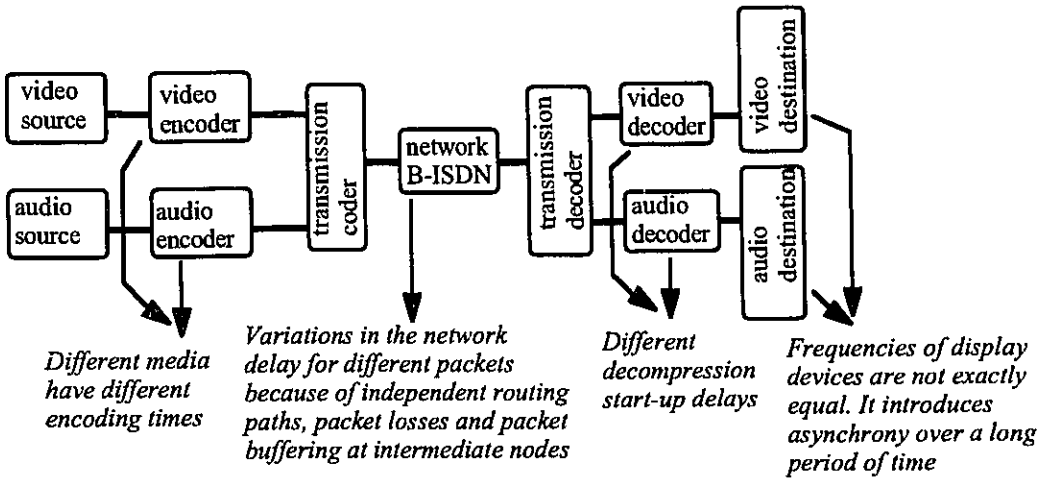

Figure 2.2.1 Various causes of asynchrony in a video telephony system

The user will be able to detect disturbances that impose two types of presentation requirements [Bla96]: the intra-object (equivalently intra-media) synchronization which refers to the time relation between two successive presentation units of the same time-dependent medium, and inter-object (eq. inter-media) synchronization which refers to the accuracy of parallel presentation of media objects. On the one hand, *gaps* may occur within a data stream because of jitters in the data transmission which affect the rate of presentation of the Logical Data Units (LDUs) [Ste92]. This corresponds to a failure of intra-media synchronization. For example a gap in an audio stream will be perceived as a sudden silence; for a video stream, the video window will freeze on the last frame displayed. On the other hand, a failure in inter-media synchronization, which is a consequence of differences in the delays experienced by different media streams, results in an asynchrony in the parallel presentation of media-objects. In a typical video and

audio presentation, the user is then disturbed by the lips of the person talking on the screen, that do not correspond to the words of the sound track, i.e., there is no lip synchronization.

However, it is difficult to objectively judge the quality of a synchronization. Different users will not react the same way since they perceive the surrounding information differently. Moreover, the synchronization requirements will depend on the medium that suffers a gap, or on the media that suffer an asynchrony in their parallel display. A gap of 30 ms in an audio stream will be detected very easily because of the 'crack' it will produce, whereas the same gap in a video stream with a rate of 30 frames per second will be perceived as two consecutive video frames that are identical since the video display will freeze on the last frame (unless an alternative action is been taken).

Media		Mode, Application	QOS
video	animation	correlated	+/- 120 ms
	audio	lip synchronization	+/- 80 ms
	image	overlay	+/- 240 ms
		non overlay	+/- 500 ms
	text	overlay	+/- 240 ms
		non overlay	+/- 500 ms
audio	animation	event correlation (e.g. dancing)	+/- 80 ms
	audio	tightly coupled (stereo)	+/- 11 us
		loosely coupled (dialog mode with various participants)	+/- 120 ms
		loosely coupled (e.g. backgroung music)	+/- 500 ms
	image	tightly coupled (e.g. music with notes)	+/- 5 ms
		loosely coupled (e.g. slide show)	+/- 500 ms
	text	text annotation	+/- 240 ms
	pointer	audio relates to showed item	- 500 ms + 750 ms

Table 2.2.2 Quality of services for inter-media synchronization [Ste93]

In the particular case of inter-media synchronization, a number of experiments have been done at the IBM European Networking Center [Ste93] that study the maximum acceptable skews such that two media presented in parallel are considered as 'in sync', i.e., synchronized, by a majority of users. The skew is the time difference between two media streams; a skew of 0 second is equivalent to a perfect synchronization. Partial results of this study are presented in Table 2.2.2. These results give the skew tolerances between media streams, depending on the media involved, but also on the type of application in which these media are used. These results, even if they are influenced by the environment in which they where done, can be considered as a strong basis for determining the synchronization QoS parameters that should be used to suit the presentation requirements of the application.

2.3 Specifying the Synchronization Constraints and Modeling the Temporal Relationships

2.3.1 Synchronization Specifications

As explained earlier, there are both intra- and inter-object synchronization constraints in a multimedia document. The specification of the synchronization constraints must describe all temporal dependencies that are part of the objects in the document. The specifications can be divided into four main types [Bla96]:

- Intra-object synchronization specification for each object of the multimedia document.
- Descriptions of acceptable QoS for the intra-object synchronization
- Inter-object synchronization specification between different object of the document.
- Quality of service description for inter-object synchronization.

QoS for intra-media synchronization relate to parameters such as the tolerated error rates, jitters, the frame rate ... In the case of inter-media synchronization, the skew between two media

objects is the primary factor (see Table 2.2.2). Among these specifications, specifying inter-object synchronization constraints has been the subject of many research projects and gave birth to numerous methods for modeling the temporal relationships between media objects.

2.3.2 Temporal Modeling

The temporal knowledge about the objects involved in a multimedia document presentation is always relative. Therefore, the temporal information about a media object is always related to another object. In describing the synchronization between two media object one can take either *intervals* or *synchronization points* in time [Hoe92] as primitives.

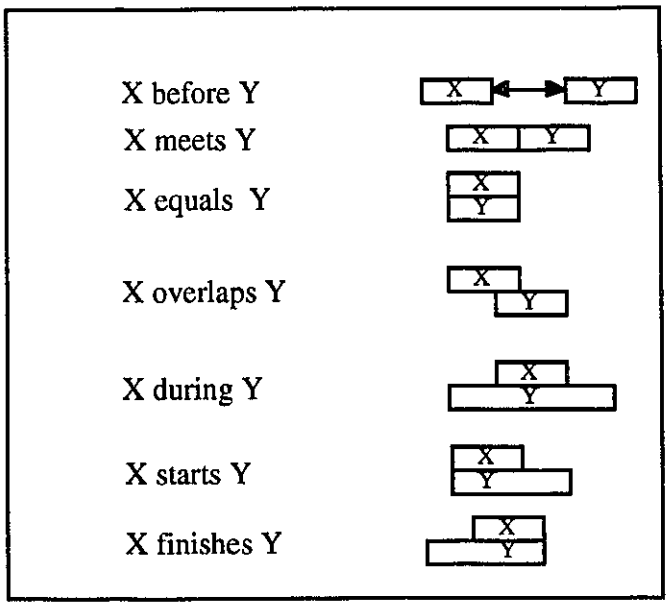

Figure 2.3.2.a Possible basic temporal relations between two media objects

In the case of *interval* based primitives, each media object represents the duration of an interval. If we exclude these which are invertible, there is seven types of basic relations between two time intervals (see Figure 2.3.2.a). We can use these seven relations to describe any type of synchronization constraints between two objects. Among the interval based temporal models, we

find the Object Composition Petri Net (OCPN), one of the very first models of this type, and the Time Flow Graph (TFG), which is used in our prototype.

OCPN

OCPNs were first presented by Little in [Lit90.4]. This is a specification method based on Timed Petri Nets, where the time is associated with the places instead of to the transitions. Each place represents a media object and its duration, or an idle object (a place that does not correspond to any media and that represents an intermission). Each vertical bar represents the transition to a following object and helps in describing the simultaneous start or end of two or more events symbolized by the connected places, since a transition will fire only when all the input connected transitions contain a non blocking token (i.e., a token that remain in a place for as long as the duration associated with that place). Each token is associated with a resource (the audio display device, for example). The presence of a token in a given place means that the corresponding object is using the resource corresponding to the token. The token will remain in a place for the place's associated duration (i.e., the duration of the media object). Figure 2.3.2.b shows, as an example, how part of a slide show presented along with a text page could be specified using OCPNs. The seven basic temporal relations presented earlier can be easily modeled using OCPNs as shown on Figure 2.3.2.c.

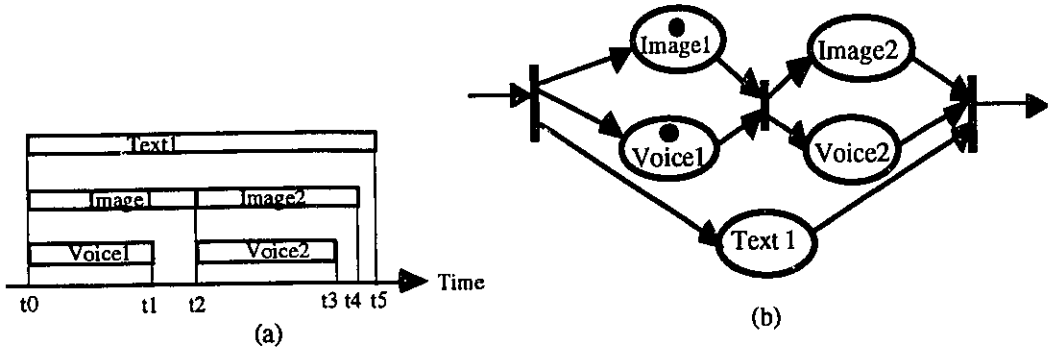

Figure 2.3.2.b (a) Timeline of a multimedia document (b) Corresponding OCPN representation

This kind of model allows the description of all levels of detail, and of all kinds of synchronization specifications. Very detailed synchronization constraints can be modeled in subnets. Then, a hierarchy can be constructed by assigning these subnets to places, for which the associated time is the longest path of the subnet. However, a big drawback of this method is the fact that in order to specify synchronization concerning the content of media objects, these need to be split in sub-objects (e.g., a video sequence needs to be split into frames)

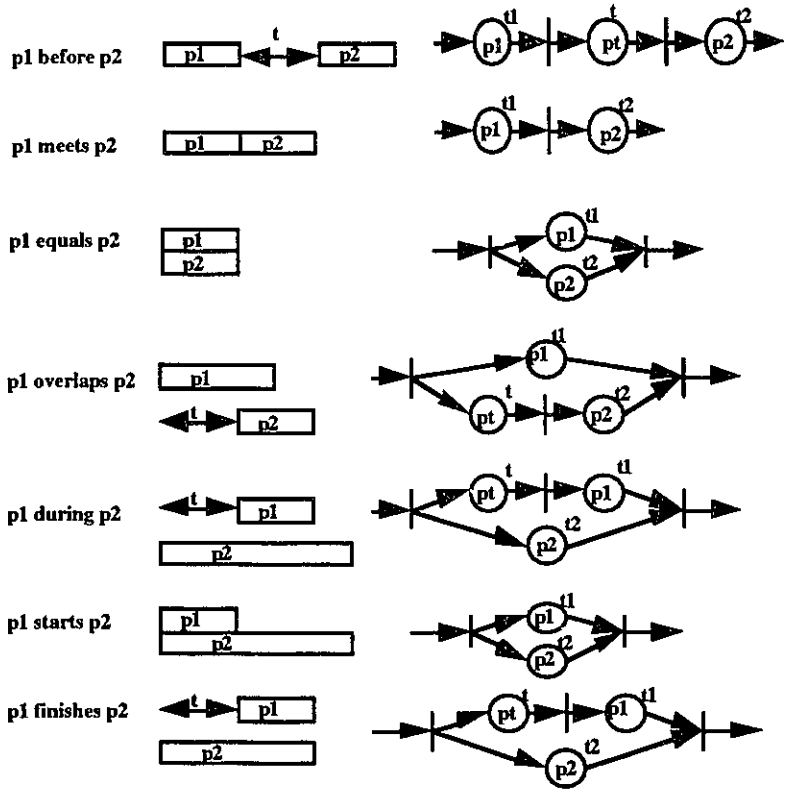

Figure 2.3.2.c OCPN modeling of the basic relations between two temporal intervals [Lit90.4]

TFG

The Time Flow Control method was introduced by Li Li in [LiLi94.2-1]. It is the temporal model that we use in our implementation to describe the presentation scenario.

The TFG model is an interval-based temporal model. A TFG is a directed graph,

containing [LiLi94.2-1]:

- Δ_N, a set of nodes for the set of interval object (each node is associated a transition time equal to the duration of the interval object),
- Δ_t, a set of transit nodes (each transit node as a transition with zero duration),
- E, a set of directed edges, used to connect the nodes and thus represent the sequential ordering of the presentation in time.

The model of any interval object is represented by a interval node N_X of Δ_N representing the actual duration of the object, and as many nodes ∂_X of Δ_N as necessary to represent the flexible intermission between this object interval and other object intervals. For an object interval the model is therefore one of:

$$(\partial_X^{(p)}, N_X, \partial_X^{(f)}), \qquad (N_X, \partial_X^{(f)}), \qquad (\partial_X^{(p)}, N_X), \qquad (N_X),$$

where (p) stands for *previous* and (f) for *after*. The existence of a ∂_X node before or after the node N_X means that either the starting time or ending time (or both) are not precisely specified. ∂_X nodes do not exist independently but only associated with a N_X node.

N_τ nodes of Δ_N are used to describe fixed intermission of duration τ between two interval objects. Such nodes are used in relations of the type *before* and *overlap*. N_τ nodes do not have any associated ∂_X nodes.

There are two types of transit nodes in Δ_t: N_S and N_e. They are used to delimit the start and the end of either the whole TFG or one of its activity (the TFG can be divided into sub-graphs, called activity, which are forming the whole TFG when they are aggregated).

Figure 2.3.2.d shows how the basic temporal relations can be modeled using a TFG

The TFG is a sophisticated specification schema. In contrast to the OCPN and other specification methods, the TFG takes into account the fact that the available temporal knowledge about the objects involved in a presentation, such as their duration, is strictly relative. Therefore, the representation of the scenario (which contains the temporal relationships between all the objects of a presentation) should allow uncertainty of information. The TFG does not require

precise synchronization constraints, such as exact event occurring times or the duration of objects' presentation, to represent all types of interval relationships. This leads to a so called *fuzzy* scenario. The *fuzzy* scenario can then be used at the proper time (i.e., when the necessary temporal knowledge has been acquired) to perform the necessary temporal scheduling.

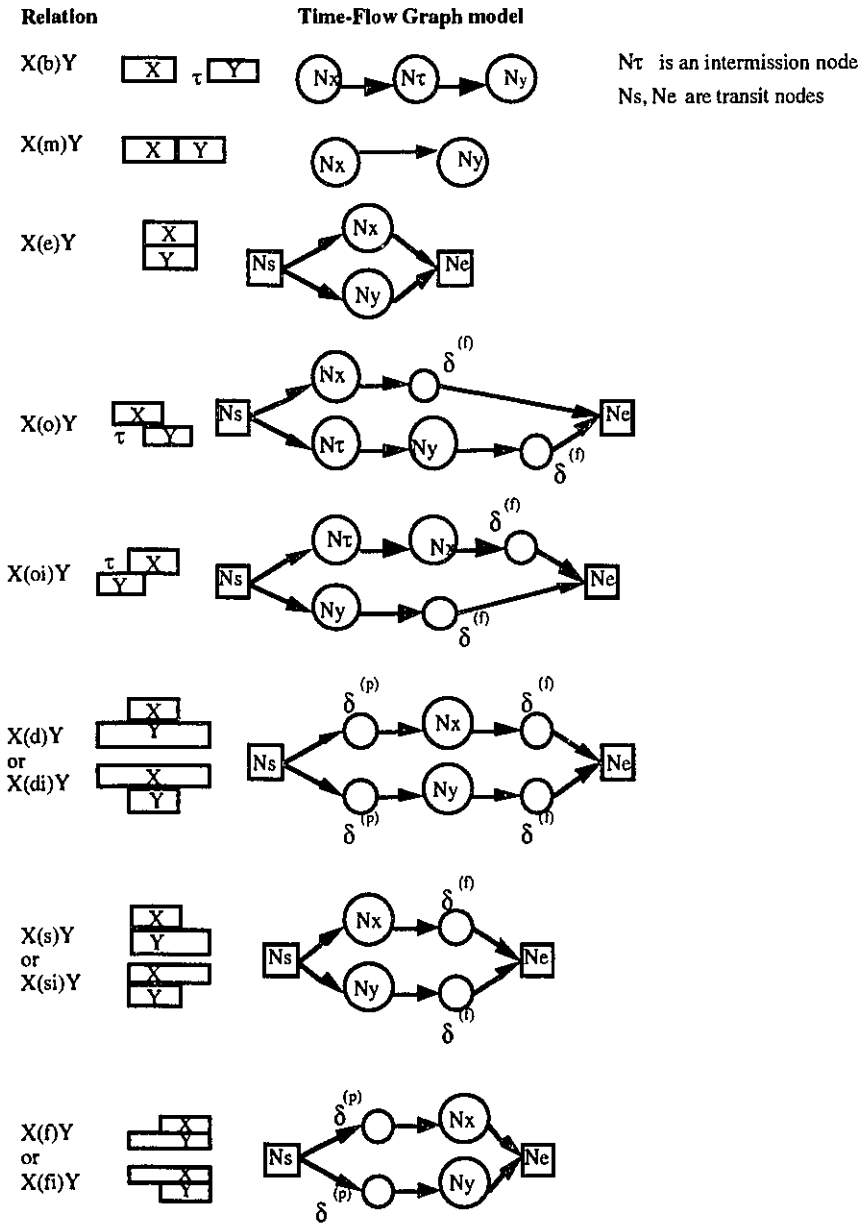

Figure 2.3.2.d TFG modeling of the basic relations between two temporal intervals.

Because the TFG model supports the creation of a temporal scenario with uncertain and relative

information, it makes it possible for additional information to be added to that scenario to derive a more precise delivery schedule. The TFG was indeed created to support a synchronization dialogue among multiple information sources necessary in certain distributed multimedia applications, dialogue which was implemented in the synchronization algorithm presented in [LiLi94.2-2].

Other Specification Methods

Many other specification methods have been the subject of research developments in the last five years:

- The hierarchical specification is presented in [She90] and is based on two main synchronization operations: serial synchronization of actions and parallel synchronization of actions. The range of possible synchronization relations can be extended by the introduction of delay as a possible action [Lit90.4]. This is an interval-based specification model.
- The synchronization via *reference points* is a point-based model presented in [Ste90]. The synchronization of media objects is done by connecting together references points of media objects. The reference points are the start and end times of the object, as well as the start times of LDUs within the time dependent media objects.

A thorough review of many other specification models is provided in [Bla96].

2.4 Transport of the Synchronization Information

In order for the receiving system to perform the necessary synchronization, it needs to receive the synchronization specification for an object before that object is to be displayed —or at least at this very moment. In the following, we describe various approaches that were the subject of extensive research.

2.4.1 Multiplexing the Media Data at the Source

If different data types are merged together into a single 'document' and transferred as such to the user, then no inter-media synchronization problem exists anymore. Indeed, the advantage of multiplexing the LDUs of different media stream on the same virtual connection is that the synchronization between streams is delivered with the LDUs (see Figure 2.4.1). For example, if two audio and video streams have to be synchronized using the multiplexing method, we just need to interleave the LDUs of the audio stream with these of the video stream. No additional information needs to be passed to the user's workstation. This is a method used in certain standards, such as ODA (Open Document Architecture) [She90].

However, multiplexing all the data in one channel makes it very difficult to select a QoS for the communication channel that matches the requirements of the various multiplexed media. For each QoS parameter, the most stringent requirement of all involved media should be guaranteed. Besides, when the source is distributed, multiplexing is not straightforward anymore and not very interesting. Actually, the advantages of transmitting the document in parallel on different connections are the following [Bla91]:

- Gain in parallelism. This can reduce the transmission time in a significant way.
- Allow 'lazy transmission', i.e., fetching of data on demand only.

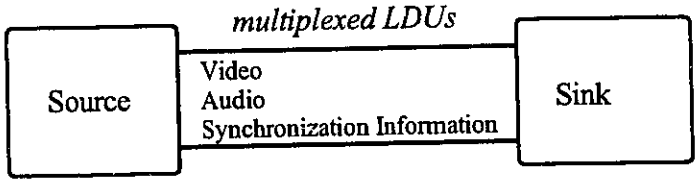

Figure 2.4.1 Multiplexed data streams

- Easy integration of disjoint sources (distributed servers, different workstations).
- Natural and efficient use of the resources. Each medium is transmitted on a B-ISDN channel

with a class of traffic that suits the required QoS for the medium (cell loss rate, cell transfer delay...) and the characteristics traffic generated by the source (peak rate, burst length...).

However, even if the multiplexing approach presents disadvantages at the transport layer, different media can be multiplexed at a higher level, in coded standards [Bla96]. This is the case of MPEG which multiplexes audio, video, and the synchronization information in a single bit stream. In this case, though, we tend to consider the multiplexed data as a new type of medium.

2.4.2 Synchronization Marker

When the media data are sent on separate channels, a way of synchronizing the data is to insert 'synchronization markers' (SM) into the data streams at the source, such that the LDUs that need to be displayed in parallel receive corresponding SMs. This approach is simple in the modifications of protocols and overhead it requires. The receiving system can then the data from early streams until all corresponding SMs have arrived, thus forcing synchronization between the streams.

Nevertheless, this method discussed in [She90] [Lit91.12] and [Ste92] presents the disadvantage of modifying the media data which disables the possibility of attaching output devices easily. This might be difficult to realize for coded streams in certain applications. Also , the better synchronization needs to be, the smaller the LDUs and the larger the number of necessary markers. This causes the transmission overhead to rapidly increase. But the main drawback is the fact that this scheme only supports parallel synchronization between the media object to the detriment of more complex synchronization schemes.

2.4.3 Synchronization Channel

The concept of synchronization channel was developed in [She90]. A synchronization channel is required, in addition to the connection for the media data, to pass the synchronization information

in real-time (see Figure 2.4.3). The synchronization commands and parameters, as well as the references to the matching points in the data streams are transmitted on this channel. This approach is very suitable when performing live and sophisticated synchronization, even when the whole synchronization information is not known beforehand. Besides, it allows the description of complex synchronization mechanisms. It leaves the media data unchanged and does not introduce additional delays. However, this channel conveys the synchronization information in real-time. Thus, it introduces an overhead in complexity of the protocols to be used and in network resources.

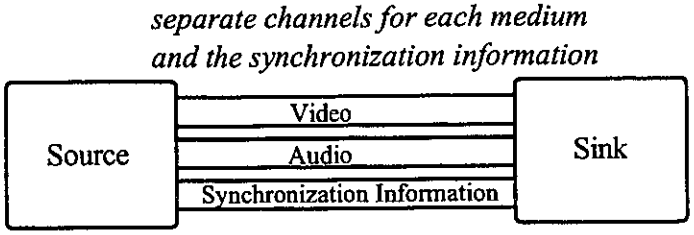

Figure 2.4.3 Use of a separate synchronization channel [Bla96]

2.4.4 Delivery of a Scenario Before the Presentation Starts

This approach is used exclusively in the case of synthetical synchronization where a document is edited and the synchronization specifications between media object are 'synthesized' during the editing of the multimedia document, in contrast to live synchronization that takes place in CSCW systems. Typically, a presentational application permanently stores the multimedia documents and their synchronization specifications after they have been edited. The specifications are often called scenarios. When the presentation of a multimedia document is required, the scenario is sent to the end-user site. Even though sending the scenario before the presentation can start is introducing a significant delay, this approach represents a simple and effective way of passing the specifications. In addition, this is a very effective solution in the case of multiple sources. This approach is used in [Lit90.11], [Lam94.5] and [Lam94.6], among others.

2.5 Synchronization Operations

Synchronizing different media types is certainly a multi-step process, especially when the system is distributed. All components of the system have a potential for introducing delays. It is thus required to perform or control the synchronization at many stages [Bla96]:

- 1) Synchronization during the acquisition of the media object e.g. 'lip synchronization'.
- 2) Synchronization during the retrieval of the media objects, e.g. from a database.
- 3) Synchronizing the delivery of the media object to the network.
- 4) Synchronization during the transport, especially for continuous media, with the use of isochronous protocols.
- 5) Synchronization at the end-user, when delivering the data to the display devices.
- 6) Synchronization within the display device (and between display devices if necessary).

Step 1) concerns the source workstation in the case of live creation of the multimedia data (e.g. conferencing systems) or a multimedia document editor in the case of document that are created for presentational applications. Step 2) deals with the file system or the database storing the documents. Transport services and underlying network have the responsibility of step 4). Step 6) is embedded in the design of the media players, either hardware or software.

The following sub-sections will detail steps 3) and 5).

2.5.1 Synchronization Scheduling

Synchronization scheduling refers to the scheduling of the times when the media objects are delivered to the network. In a stored-data application (typically, a presentational application) by opposition to one that has live-data sources, the system has more flexibility in scheduling the times at which the data are communicated to the application through the network.

In [Lit90.11], using an OCPN-based temporal model, Little and Ghafoor designed an

algorithm for scheduling the data delivery. The algorithm was performed in a centralized manner, either at the destination node or in some intermediate node. This node retrieves the temporal specifications and derives the playout deadlines for each media object. Using traffic predictions on the virtual connections, a sub-schedule —retrieval schedule— was computed for each information source to compensate for important delay mismatches between separate data streams.

Other algorithms have been developed. In [LiLi93.4], Li Li describes the necessary dialogue between the sources which arise as a new requirement, when there are multiple independent information sources (e.g. distributed database servers) in a presentational application, in order to convert the relative and uncertain temporal requirements into the absolute and precise synchronization controls. Li Li and Lamont presented in [LiLi94.2] and [Lam94.6], respectively, a distributed scheduling algorithm for a presentational application. The algorithm was based on the TFG model described earlier. It allowed to get rid of the eventuality that the centralized scheduler becomes a bottleneck, which adds to the robustness of the algorithm. In addition, the traffic predictions are based on the guaranteed QoS parameters provided by an ATM-based communication network.

2.5.2 Synchronization Recovery at the End User

Due to the probabilistic nature on which the traffic predictions are based, synchronization scheduling is not sufficient to obtain a good synchronization. Variations on the delay introduced on each data connection alter the performance of the delivery scheduling and the quality of the simultaneous data delivery of the multimedia data at the user's site. A mismatch between different streams will appear as a synchronization error for the user. Thus, it appears necessary to control the quality of the synchronization of the LDUs of media to be synchronized, to filter out possible gaps and smooth jitters. These compensation mechanisms are called error recovery algorithms.

One of the numerous researches on that aspect is developed in [Ran91], where Rangan *et al.* presented a feedback technique which can inform the sender about the synchronization errors in the playback. The sender receives feedback data units sent by the receiver and thus detects existing asynchronies, if any. The sender can then adjust the data transmission of the different data streams, e.g., speeding up the slow stream by skipping some data or slowing down the fast stream by repeating media information. This scheme does not require any global clock or any specific control message exchange between the sender and the receiver, and appears therefore very suitable for applications with simple client terminal, such as video-on-demand, where heavy computing cannot be done at the receiver. However, this method might not be very robust for applications with stringent real-time characteristics since the correction of the asynchronies is not performed at the receiver (where the asynchronies are detected), but instead at the sender.

Other methods are developed in [And91] and [Lam94.6] among others. Anderson introduces the Logic Time System (LTS) for controlling the synchronization of continuous media. Each LDU of a continuous media stream is associated with a time-stamp. LDUs with the same time-stamp are displayed at the same time. Asynchronies are adjusted at the receiver by skipping data for speeding up the stream or pausing the data presentation for slowing down the rate. This method is more suitable for real-time applications. It may however present a problem for coded streams which are very sensitive to data loss. The approach developed in [Lam94.6], called Stream Synchronization Protocol, is thoroughly explained in section 3.2.

2.6 Summary

In this chapter we described the general requirements of multimedia applications. We then developed the specific problem of synchronization within these applications. We presented quantitative synchronization requirements from a user's point of view. Then we described some possible methods for specifying synchronization between the media objects, and how these specifications could be sent to the node performing the synchronization. Finally, we detailed two

important steps that intervene in the whole process of synchronization.

This review on multimedia systems and their synchronization issues is not exhaustive; many synchronization issues that are the subject of numerous research have not been described here: group synchronization issues in cooperative applications (need for coordination and causal ordering between the participants [LiLi93.6]), issues related to the ATM communications services, QoS negotiation, coded streams, databases, transport protocols...

Chapter 3

Implementation of a Media Synchronization Control Architecture for a Multimedia News On-Demand Prototype

In this chapter, the concept of a Multimedia News On-Demand system will be presented. Then the general design of the prototype will be explained, describing the various entities involved and how they interact with one another, as described in [Lam94.5, Lam94.6]. In particular, certain decisions will be discussed that have been made during the implementation and slightly modify the previous architecture. The rest of the chapter is devoted to giving a thorough description of the 'local' prototype's functionality and its components.

3.1 A Multimedia News On-Demand Service

Information is one of the most important components of our contemporary society. The need for news has various motivations. Among these are decision making, curiosity, and culture. The institutional media used to convey information have changed with time. At first newspapers

printed text and graphics, then pictures. Then, the radio brought the public audio news. Later, with the TV and the VCR, the news became an alliance of audio and video. Today, the technological advances in electronics, networking and computers are about to bring together the newspaper and broadcasting industries into the field of multimedia news over computer networks.

A multimedia news article will possibly contain a combination of the media stated above: video, audio, text, images, graphics, etc. In a multimedia news system, an article will be created by integrating the various information sources (video clips, newspaper articles, pictures...) into a single document. The integration of the sources, i.e., the edition of the multimedia article, corresponds to defining the document structure, with both the spatial (relative position of the document's components on the computer screen) and temporal structures. From here on, the temporal structure is referred to as the document's *scenario*. Finally, the multimedia document is stored in a multimedia database along with its scenario.

Typically, a person who wants to access a multimedia news on-demand service will first need to be registered with such a service. At registration time, the user's profile will be defined:

- The amount of available news is now so enormous that this service must provide the user with a means to filter the different topics, and only present those which are of interest for him/her.
- The multimedia news service will deliver its news articles to the user through a high-speed broadband network able to maintain a guaranteed quality of service required for the presentation of media such as audio or video. The cost of the news service will depend on what quality the user desires for the presentation. A default 'quality of presentation' needs therefore to be chosen by the user, and it will be recorded as a part of the user's profile.

The user's profile is thus a set of default parameters used to provide a better and faster service. Both aspects of the profile can be of course re-negotiated, either definitively or only for occasional presentations.

The service must provide the user with a very user-friendly user interface which allows

one to browse the topics and articles available in the database in a very intuitive and fast manner. Once an article has been requested for retrieval, the presentation is done in a highly interactive environment, providing the user with all the controls of a VCR: the presentation can be started and paused at will, scanned backward or forward, and stopped at any time.

In order to use the multimedia news service, the user also needs to have a terminal or a workstation which can handle a multimedia presentation. This implies the presence of both hardware and software components. The hardware components range from the loudspeakers, necessary for the audio retrieval, to hardware decoders if possible (e.g., for real-time video decoding). Required software components range from the synchronization control software, provided by the news service, to software decoders. Finally, one of the requirements for the user's workstation supporting a multimedia news presentation is the buffer space, since the data sent from the database need to be temporarily stored before being displayed.

Depending on the schemes used to perform the synchronized presentation of the media, the required buffer space will be larger or smaller. A complete store-and-forward approach where the whole document to be viewed is sent at once before the presentation starts obviously constitutes the worst case in terms of required buffering. In order to minimize this buffering, we need to minimize the time spent by the data in the buffers at the user's workstation, and thus transmit the multimedia data according to a "just-in-time delivery" scheme. The following section presents a software control architecture designed to meet this goal.

3.2 The Synchronization Control Architecture

The synchronization architecture that will be discussed in this section was first presented in [Lam94.5] and [Lam94.6].

We consider a Multimedia News On-Demand system, in which the multimedia database is composed of a database server and distributed media servers. Each media server delivers the

data of a multimedia news article to a remote user's workstation using independent data connections of an ATM network, without the use of a global clock.

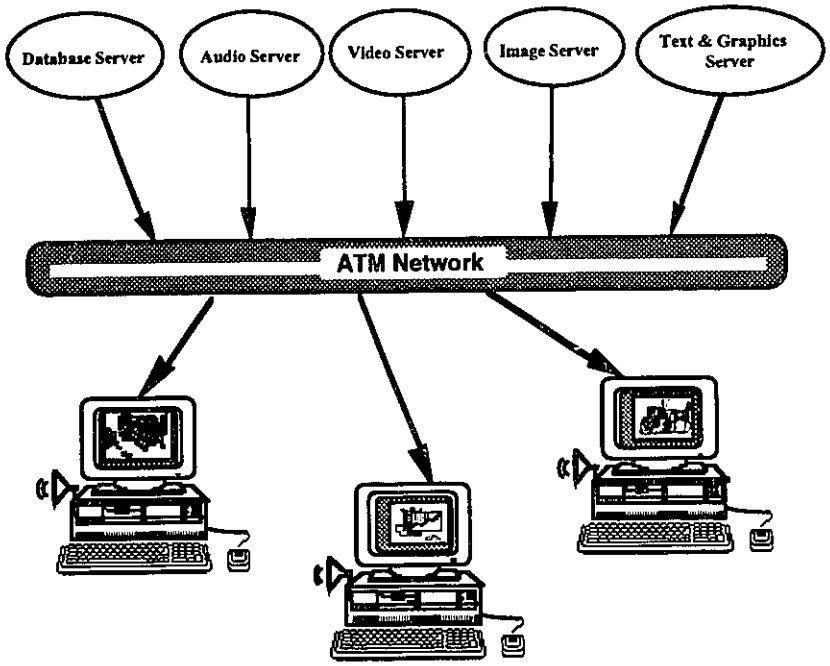

Figure 3.2.1 Architecture's Main Components

The database server stores the multimedia articles along with their presentational scenario, which represents the temporal characteristics of the media objects and the inter-object synchronization parameters. The media servers store the objects of their own medium type and retrieve them at presentation time. Each medium can have its own server, but a configuration can be envisaged where continuous media such as audio, video and voice would be stored in a continuous media server, whereas non-continuous media would be stored along in another server. The underlying communication network is an ATM —connection oriented— network. The data for different media are transferred on different virtual connections, each with different traffic characteristics and QoS requirements adapted to the medium they are transmitting.

In such a system, each data connection is introducing random delays during the transmission. However, the QoS parameters of the connections only provide us with estimates

of the mean delay and delay variation of each connection, and we can only use these two parameters to partially compensate for the random delays. As a consequence, the continuity of the data may be interrupted within a single data streams because of the delay jitter on the connection, thus introducing gaps in the presentation of the corresponding medium. Moreover, different delays on different connections may create a mismatching in the presentation, since data packets of different media are transferred on parallel connections. Both the discontinuity and the mismatching destroy the temporal relationships specified in the scenario, and address respectively the problems of intra-object and inter-object synchronization.

In order to preserve the temporal presentation of the multimedia objects, a synchronization method in two steps is proposed:

- 1) A *stream delivery scheduling* computes, for each media server, the delivery schedule of the corresponding media objects, taking into account the known QoS parameters of the connection from the server to the user's workstation.
- 2) The *Stream Synchronization Protocol* (*SSP*) performs *synchronization recovery* at the receiver when network delay variations make it necessary.

Stream Delivery Scheduling

The stream delivery scheduling is based on the knowledge of the presentational scenario, of the network delay characteristics for each data connection (which are part of the negotiated QoS at connection setup time), and of the decoding delays for encoded streams. The algorithm consists of computing a delivery schedule using the above knowledge. The latter specifies the times at which media servers should deliver their objects on the data connections in order to meet the temporal requirements of the scenario concerning the presentation of the media objects at the user's workstation. The times given in the delivery schedule are off-sets to the absolute starting time.

Initially, the user requests to view an article. The presentational scenario is returned by the database server to the scheduler. The latter determines which media types are present in the

scenario, and creates the corresponding Client Media Synchronization Controllers (CMSCs) (Fig. 3.2.2). Still using the scenario, the scheduler computes a Time Flow Graph (TFG) and the presentation time of each object. When the control schemes were first designed, the CMSC of coded media streams was supposed to stand, in the data flow, between the decoding of the data and their presentation. This provides a better control on the presentation of the data —the decoders also introduce random decoding delays— however it degrades the buffering performance of the synchronization schemes. In the implementation, the data are decoded just before being presented to the user, and therefore just after the control of the CMSC. This implies that the presentation schedule that is computed as explained previously takes into account the decoding delays for media coded streams.

A three-party QoS negotiation takes place between the transport system, the media servers and the client, trying to find an acceptable compromise between the media server constraints, the user's profile and the transport system constraints: i) the scheduler opens a control channel to each media server, and notifies the corresponding Server Media Synchronization Controller (SMSC) to open connections with the required QoS (according to the user's profile); ii) depending on the actual capacities of the media servers and the available resources of the network, the connections are opened, eventually with a reduced QoS if no more can be provided and if the user is ready to accept the degradation. The SMSCs return the negotiated QoS to the scheduler. The mean network delay and delay variance between each media server and the user's site are then used to derive a delivery schedule for each SMSC, from the presentation schedule. The scheduler sends each delivery schedule to the appropriate SMSC. Each delivery schedule contains the starting time off-sets for all the objects of a medium.

Once the user requests to start the presentation, the request is sent to the SMSCs, which start to execute the delivery schedule. It is the responsibility of the SMSCs to send each media object according to its relative starting time. At the client site, the CMSCs are used during the retrieval of the document to perform the second step of the synchronization control.

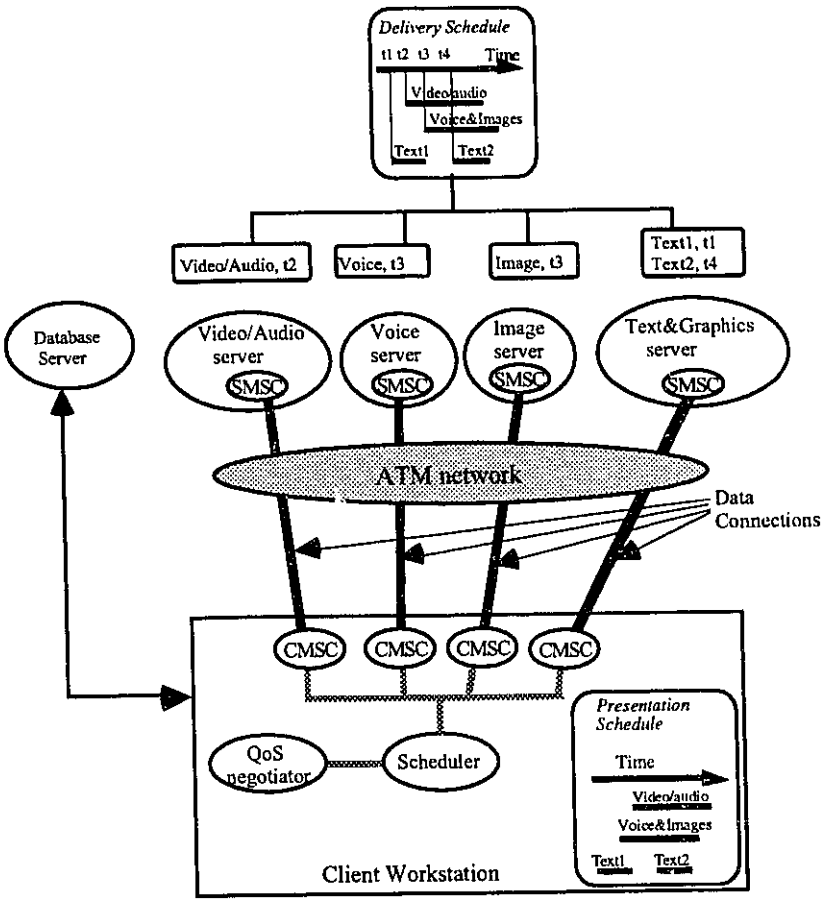

Figure 3.2.2 Deriving the presentation and delivery schedules

The Stream Synchronization Protocol

Due to the presence of random fluctuations in the network delays, the stream delivery scheduling proves to be insufficient to guarantee the media synchronization during the playback. Other compensation mechanisms are therefore necessary at the client's site to recover from the asynchronies, on the fly. This is the role of the Stream Synchronization Protocol (SSP) which is performed on each media object just before it is played back. Before a media object is displayed, the CMSC controlling the given media type will compare the actual time to the presentation time of the object. The presentation schedule containing the presentation time off-set of all the media objects involved in the document is shared between the CMSCs. The first packet received on

any stream triggers the CMSCs to start executing the presentation schedule, by transforming the time off-sets of the presentation schedule into absolute presentation times. From the derived TFG the CMSCs deduce data that should be received simultaneously and the number of packets that should be received in an interval of time. When the delay experienced by an object is so important that this object cannot be played back at the scheduled time, there is a risk that the media streams will not be displayed in synchrony anymore. The tolerated mismatch between media in the playback varies with the media that are to be synchronized; they are referred to as the skew tolerance parameters or Synchronization QoS values, and they are based on experiments conducted by Steinmetz in IBM Heidelberg [Bla96]. If the delay encountered by a CMSC on its stream is below the tolerance parameters, it is passed to the other CMSCs by rescheduling the presentation of any media objects scheduled after the late one (the presentation times are postponed by the value of the delay). If the delays encountered on a stream are much larger than the tolerated skews, QoS re-negotiation should be done by the system if possible. The presentation is aborted otherwise.

3.3 General Information on the Prototype

3.3.1 Hardware and Software Equipment

The development of the prototype has been done on IBM RISC/6000 workstations (IBM PowerStations 360), running the version 3.2 of the IBM-AIX Base Operating System. Technically the prototype is flexible enough to be run on a single machine, but it is essentially designed to run on many machines: one for the user's workstation, the other(s) for the servers' site(s).

The servers' site(s) and the user's workstation are connected through various types of networks: Ethernet, Token Ring and ATM networks. The transport protocol that is used at this

point of the development is TCP (Transmission Control Protocol) on top of IP (Internet Protocol) for each of the network types previously stated. Some other transport protocols such as the eXpress Transport Protocol (XTP) and ST II are to be made available in the near future.

For the development of most of the software, we used the IBM C Set ++ (version 2). The IBM AIXwindow Interface Composer version 1.2 (AIC) was used to create the major part of the graphical user interface (GUI). The text player and the MPEG1 decoder/player have been developed independently from the rest of the user interface. The text player was created from scratch using the C Set ++ and the Motif libraries. For the MPEG1 decoder/player, we used a public domain MPEG1 video software decoder — developed in the Computer Science Division of the University of California at Berkeley.

3.3.2 The Various Software Components

The prototype has two main entities, the multimedia database with its distributed servers, and the client application installed on the user's workstation. As stated earlier in section 3.3.1, the implementation allows the media servers and the Database server to stand on the same or on different workstations. For practical reasons, all the components of the multimedia database are on the same workstation, which we will call the server's workstation. On the server's site we therefore have four main components: the Database server, and the text, audio and video servers. When a user requests the presentation of a document, each media server involved creates a SMSC that will control the delivery of the media objects to the transport system. On the second workstation, we find the user application. It is composed of the initial process (mainly controlling the graphical user interface), the scheduler, the Socket_MSCs and FIFO_MSCs and the three media players. Each media type has one Socket_MSC and one FIFO_MSC, each of the two implementing a specific part of the CMSC. Each of the mentioned components corresponds to a Unix process. The processes that are indeed participating in the synchronization control software are the scheduler, the Socket_MSCs and FIFO_MSCs, and the

SMSCs.

Communication between some of the components of the prototype is needed. Control channels are especially required between the SMSCs and the scheduler or between the graphical user interface and the Database server. On the other hand inter-process communications (IPCs) are also necessary among the many processes running at the user workstation, either to perform the stream synchronization protocol or to pass the user interactions to the appropriate processes.

3.4 Inter-Process Communications

The Inter-process communications (IPCs) have been achieved through many different means. This section will shortly describe each of the IPC means that have been used and what functionality they have.

- RPC routines have been used for all the communications between two processes on different workstations, i.e., between a process at the server site and one of the processes of the user application.
- Within the same workstation, i.e., between processes of the user application, we are using the classical Unix IPC facilities such as signals, semaphores, FIFOs (First In First Out), and shared memory, to pass information or share certain data.

3.4.1 Remote Procedure Calls (RPCs)

A RPC is a high-level network application model that allows to hide the details of the underlying network to the different components of a system. It is a common extension over a network of a common procedure call. The local system —called the *client*— invokes a procedure on a remote system where the *server* stands. During a RPC, the *client* passes arguments to the *server* which

returns some values to the *client*, as for a simple routine. Figure 3.4.1 [Ric90] shows the different successive steps involved in a RPC, and shows in particular that both the client routines and the server routines are hidding the existence of a network communication service since there are making local procedure calls to their local stubs (whose role is to marshal or unmarshal the arguments of the local call, i.e., put them in a standard format).

In the implementation, the transport system used to build the RPC is TCP/IP. RPC's are used in two cases: between the Database server and the initial process at the client site (see Figure 3.7.2.a) and between the scheduler and the media servers (see section 3.6.1) .

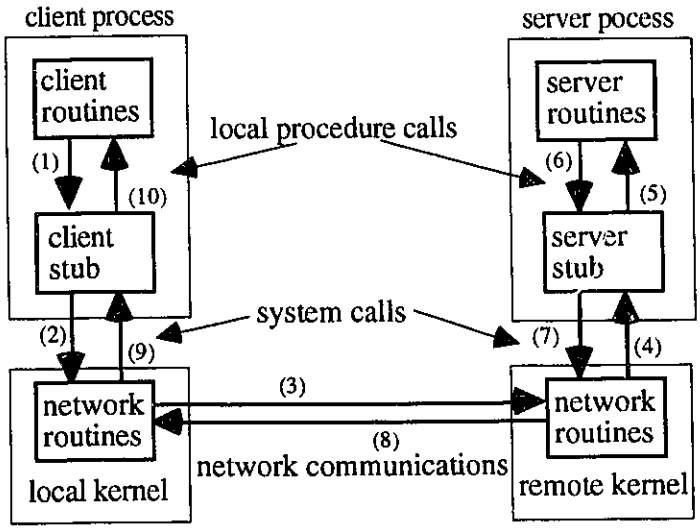

Figure 3.4.1 The Remote Procedure Call model

3.4.2 IPC Techniques for a Local Host

Signals

Signals are used by a process to send a warning to another process. Upon reception of a signal, the receiving process can either ignore, perform a default action that depends on the signal's number, or execute an operation specified in the signal handler, before resuming its normal

operations. We use signals between all the processes running on the client's site to inform all the processes that the user has selected either of the PLAY, PAUSE or STOP buttons, so that each process performs the necessary operations. We also use it to synchronize certain operations of different processes: for example, the display process signals the FIFO_MSC after creating the FIFO so that the latter process can open it for writing.

Shared Memory

The shared memory is a memory segment that is not part of the local memory of any given process but that can be accessed by any process detaining the identifier of the memory segment. Since multiple processes have to share a piece of memory, they must coordinate their action to guarantee the integrity of the data stored in the shared segment: this is done by using semaphores or flags stored in the shared memory itself.

In the implementation, we use shared memories in two cases:

- We store the shared presentation schedule in a memory segment shared by all the FIFO_MSCs. Each of them uses it as a reference table to check their objects' presentation time, and to update the presentation time of all media objects whenever a re-scheduling operation is necessary. The exclusive access to the shared memory is guaranteed by a semaphore (see below).
- The media data received from the network connection by the Socket_MSC are stored in a buffer and can be accessed immediately by the FIFO_MSC since the buffer is a shared memory segment. The integrity of the data stored in th buffer is guaranteed by flags associated with units subdividing the buffer space.

Sub-section 3.6.2 gives more details on the implementation of these shared memory segments.

Semaphore

A semaphore is a set of non negative integer values stored in the kernel and accessible by any process that knows the unique semaphore identifier (the size of the set is chosen according to the

programmer's need within system defined limits). Each element, commonly called semaphore as well, is usually used to provide synchronization between many processes or to control the number of processes simultaneously accessing a shared resource. The integer values represent the number of resources associated with the semaphore. A null value means all the resources are being currently taken by some processes.

In the implementation, we use binary semaphores. With a binary semaphore we can ensure an exclusive access to a resource (only one process at a time is granted this access).

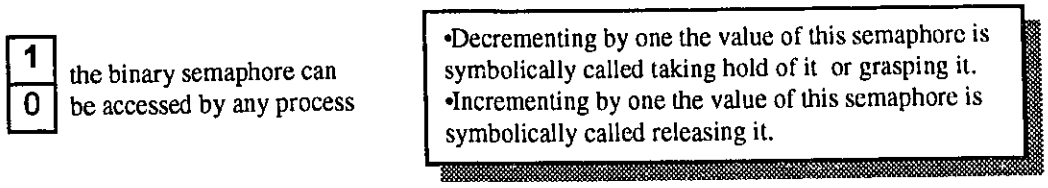

Figure 3.4.2.a The binary semaphore

We used binary semaphores in three occasions:

(1) A set of binary semaphores (the `ready_to_start` flags), with as many values as objects in the scenario, is used by the FIFO_MSCs to indicate whether a given object of its media type is ready to be presented. The semaphore, in this case, is used simply because of the information that it stores and the fact that the information is shared between different processes (i.e., a shared flag). This indication is done when an object has arrived to its FIFO_MSC, which checked the shared schedule and decided that it was time to pass the object to the media player. The other FIFO_MSC then get this information by checking the value of the semaphore associated with the object.

(2) The access of the shared schedule is controlled by a single semaphore: this is the typical case of the use of a semaphore to grant an exclusive access to a shared resource. The semaphore is associated with the shared memory storing the shared presentation schedule. When a process accesses the presentation schedule, the semaphore is decreased from 1 to 0, blocking the access for any other process (see Figure 3.4.2.b); when exiting, the process increments back the

semaphore to 1. The first process waiting in the queue to grasp the semaphore then decrements it and accesses the shared memory.

(3) Synchronizing the Socket_MSCs and the FIFO_MSCs: as on Figure 3.4.2.b, we want the Socket_MSC (process A) to block the FIFO_MSC (process B) until PLAY is pressed by the user. To avoid certain problems linked to the interruption of Input/Output operations by a signal, the Socket_MSC uses a semaphore instead, to indicate the FIFO_MSC that the presentation has been started. It holds the semaphore until the PLAY is selected and then releases it. The FIFO_MSC can then take hold of the semaphore go on with further tasks.

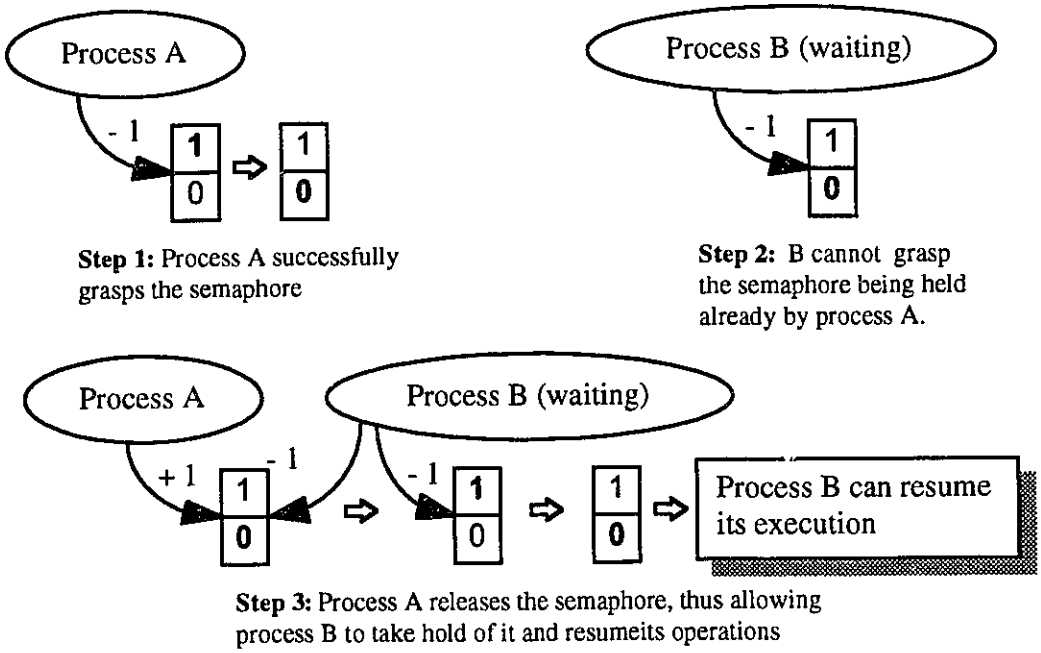

Figure 3.4.2.b Synchronizing two processes with a semaphore

FIFO

A FIFO (First In First Out), also called *named pipe*, is a communication path between two processes used to transmit data from one to the other. One of the processes is reading at one end of the FIFO while the second is writing at the other end. In our implementation, a FIFO is opened between each pair of FIFO_MSC and the associated media player The latter only reads

the data, while the first one only writes them on the FIFO. Implementation details are given in sub-section 3.6.2.

3.5 The Server Site

3.5.1 The Database Server

The Database server is exclusively designed to support the MCRLab Synchronization prototype. In the larger demonstration involving the 6 universities, all the multimedia database aspects of the prototype are the responsibility of the two research teams of the University of Alberta and University of British Columbia .

Thus, this database is really a simple, read-only database with a minimal RPC interface. Adhoc queries are therefore not supported. The database is conceived as a file system. The database is in fact using two information files, in which meta-data are stored: the `mmdb_datafile` file, which contains all the information on the articles that can be displayed, including the presentational scenario and the `000000xy.dat` files (where x and y are each a one digit number) which describe the media objects involved in the articles whose article id are xy (as shown on Figure 3.5.1).

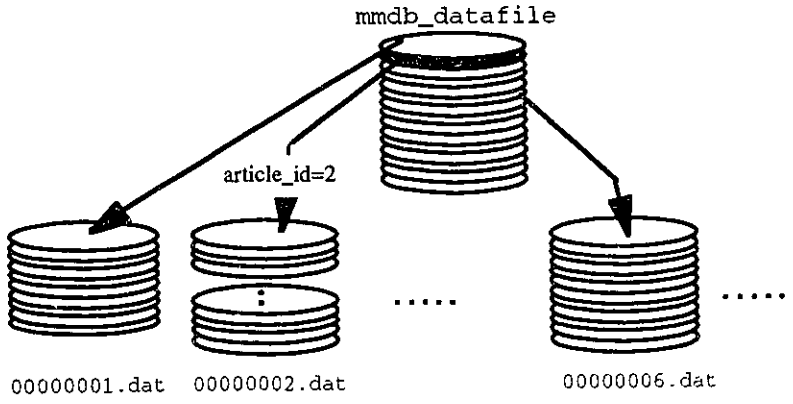

Figure 3.5.1 Structure of the database

The mmdb_datafile

The `mmdb_datafile` can be seen as a table in a database. The information that it stores concerns all the articles that are available in the database. Each line of text describes an article; it contains a fixed number of fields, each of them having a fixed or maximum length. The fields are, in the same order as implemented:

field [length of field] or *field [≤ maximum length of field]*

-the article id [8]

-the title [≤ 50]

-the date of creation [6]

-the date of last modification [6]

-keywords [≤ 40]

-duration [6]

-author [≤ 17]

-copyrights [≤ 36]

-the coded QoS values[≤ 24]

-the scenario [≤ 2047]

Each line describing a specific article can be seen as a tuple in the database terminology. The following example shows what the 4th line of our `mmdb_datafile` looks like:

```
00000006Terrorist.Group's.Chemical.Attack.Deadly••••••••••19059520
0595Terrorism,.Japan,.Chemical.Weapon•••••••000312Phil.Donahue••••
•1995.News.Inc.••••••••••••••••••••Coded.QoSValues•••••••••1:91g
lob_v010009802glob_a010009801glob_v020010002glob_a020010001glob_v0
30010002glob_a030010001glob_v040010002glob_a040010000glob_t0100071
70glob_t020008460m2??0s10000000080002m4??2s30000000080004m6??4s500
00000080006s70000000080008s10000000240008m9??
```

Each "." represents a space and each "•" represents a character (a byte) of a field that is not being used. The few bold characters have just been added for readability purposes, to indicate

the first byte of a field.

The 000000xy.dat Files

Each `000000xy.dat` file is containing the necessary additional information about the media objects in the article with article id '000000xy'. This additional information is the length of the media objects in bytes that is used by the CMSCs to differentiate the data of one object from the data of its predecessor or successor. Each object is referred to by its media type and its object id. For example, the 'first' article of our database (i.e., the one with the article id 00000001) will relate to the file `00000001.dat`, which looks like:

(object id)	*(media id)*	*(length in bytes)*
glob_v01	1	682076
glob_a01	2	215600
...	...	...
glob_v18	1	246384
glob_a18	2	82134
glob_t01	0	143
...		
glob_t24	0	37

The Server Operation

Just after having been started the Database server first reads the `mmdb_datafile` file line by line and populates a linked list of articles by invoking the routine `mmdb_load(1)`. Each element of the linked list is a structure containing the same fields as a line of the `mmdb_datafile` file. For each of these elements, a linked list of media objects presented in the corresponding article is loaded using the routine `get_objects(2)` and the `00000000xy.dat` files, thus keeping track of the length of each of the media objects.

After the database is initialized, the Database server basically waits for any user to make a request through a Remote Procedure Call (RPC). The database server is ready to traverse the linked list to provide information to the RPC clients. Three types of RPCs can be issued to the Database Server, all coming from the GUI at the user's site. There are therefore three database/RPC interface routines, which traverse the linked list of articles and return the appropriate data to the client (more exactly to the Multimedia News Client process):

`get_abstracts()`	It returns a linked list of article id's and titles.
`get_info(article_id)`	It returns the most detailed information about a given article.
`get_article(article_id)`	It returns, for a particular article, the QoS parameters, its scenario, and the list of presentation objects.

3.5.2 The Media Servers and the SMSCs

Each media server is started individually on the server site and does not got through any initialization stage. The media servers are initially waiting for a client that needs their service.

On the site of a user that made a request to the Database server to view an article, the scheduler deducts from the scenario the types of media involved in the document. For each media type, it opens up a RPC connection to the appropriate media servers and passes them the appropriate delivery schedule using the RPC routine

```
open_stream_1(struct stream *stream) ,
```

where the structure `stream` contains all the information about the CMSC, as well as the QoS information necessary to open the data stream to the CMSC, and the delivery schedule that will be used to retrieve the media objects in time:

```
struct stream {
        char      client_ip[15]      /*client host IP address      */
        int       client_port        /*client TCP port             */
        dsched    *schedule          /*delivery schedule           */
        int       throughput         /*stream throughput           */
        int       dela               /*mean stream delay           */
        int       jitter             /*mean stream jitter          */
        int       TSDU_size          /*transmission data unit size*/
        }
```

With the information on the CMSC and on the required QoS parameters, the media server opens the required connection to the CMSC. However, since TCP is the only transport protocol fully available in the prototype at the time, the QoS parameters cannot yet be taken into account.

Once the connection is set up, the media server waits for the presentation to start. At presentation starting time, the media server receives a RPC issued by the scheduler:

```
play_1(int port).
```

As soon as the call is received, the media server forks the SMSC which begins sending the media objects according to the delivery schedule.

Three RPC routines are implemented for accessing the SMSC functionalities:

```
play_1(int port), pause_1(int port), stop_1(int port)
```

They are used to transmit any of these three actions of the user: begin the playback, i.e., start the presentation of the document (PLAY), pause or resume the presentation (PAUSE) and stop the presentation (STOP). The SMSC receiving the instruction to pause is just interrupting the transmission of the data no matter at what point in the transmission it is, or what media object is being delivered. The transmission resumes when the `pause_1` RPC routine is issued while the SMSC is in a paused state.

The presentation of an article will be considered to be finished only when the user clicks on the STOP button of the GUI (also in the case where the end of the article has been reached). In this case, the `stop_1` RPC routine closes the socket stream (data connection for TCP) and

the SMSC terminates. The media server continues running nevertheless and the user can start again with any other article.

3.6 The Client Site: the User Application

The architecture at the users site consists of the following: the initial process that controls the Graphical User Interface (GUI), the scheduler, the Socket_MSCs and the FIFO_MSCs for each of the involved media, as well as one display process per medium.

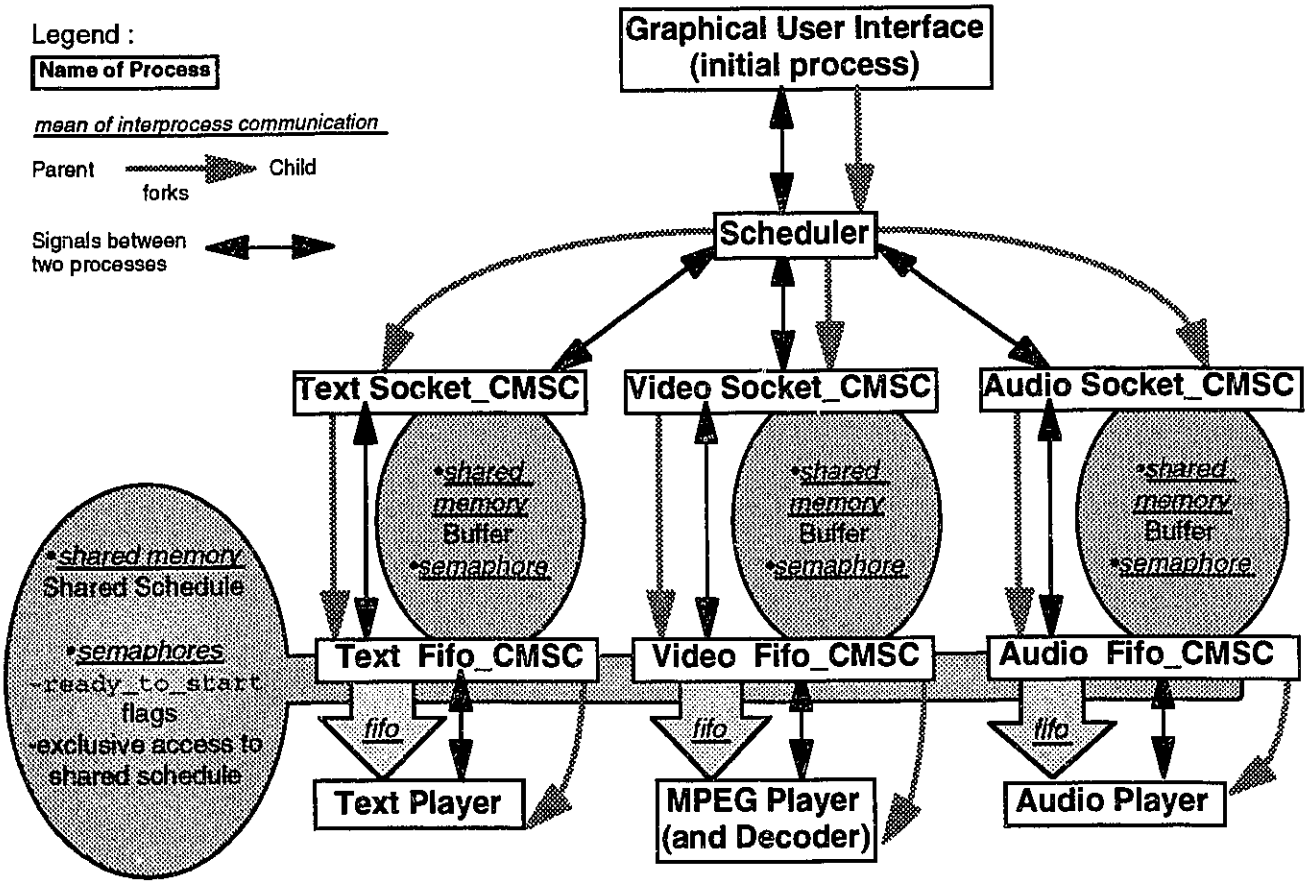

Figure 3.6.0 Parent/Child Relationship between Processes and Inter-Process Communication within the User Application

Figure 3.6.0 shows the various processes that are running on the user's workstation when the multimedia document involves all the media allowed in our prototype: audio, video, and text. This figure also shows the 'forking' hierarchy between processes, i.e., what parent/child relationships exist between the various processes, which process forked what other process. The last information contained in the figure is the different means used by the processes to communicate with one another, which will be examined at the end of this section.

The processes that are part of the synchronization architecture, and the display processes, will be described in this section. The graphical user interface (GUI), and the initial process that handles it, will be the subject of section 3.7.

3.6.1 The Scheduler

The scheduler computes a Time Flow Graph (TFG) from the presentation scenario. It then derives a local presentation schedule for the CMSCs, as well as a delivery schedule for the media servers and SMSCs, taking into account the variable network delays. It also creates the necessary CMSCs. Finally, the scheduler is the entity that informs the CMSCs and the SMSCs of any user interaction, such as PLAY, PAUSE, or STOP.

When the user requests to view an article (i.e., when the user presses START, see Figure 3.7.1) the `start_1()` RPC is sent to the Database server, which has to retrieve the temporal *scenario* of the document as well as QoS parameters and the duration of the media objects. Upon reception of the *scenario* the scheduler is forked. Then, the scheduler:

- puts the *scenario* into a usable format (see Figure 3.6.1),
- generates the TFG from the *scenario*, using `scen_TFG()`,
- generates an absolute presentation schedule using `obj_sch_bis()`, which does not take into account the delays introduced by the decoder.
- then, it queries the QoS parameters and the decoder delay values; these are all hard-

coded in our implementation, instead of using application programming interfaces (API) with the QoS negotiator, to the decoders, and to the media players, as would be required in a complete prototype.

- with these values, the scheduler can derive a presentation schedule that takes into account the decoders' delays (this is necessary since the decoders are placed after the CMSCs), and the delivery schedule by calling `delivery_sch()`.

From the scenario, the scheduler knows what media types are involved in the document and therefore the CMSC types that should be forked (using `create_client_MSC()`). Before forking the CMSCs, the scheduler creates the shared memory (with `init_shmem()`) used to store the shared schedule, as well as the 'ready_to_start' semaphore set that allows the CMSCs to signal the arrival of a given media object (calling `create_ready_to_start()`).

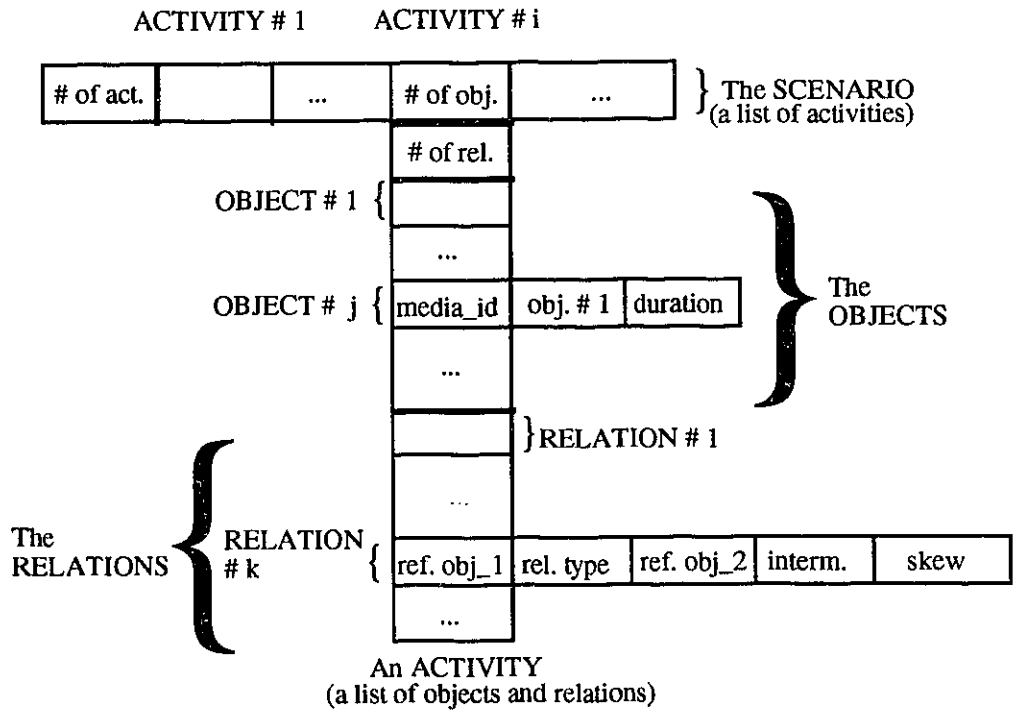

Figure 3.6.1.a Example of the structure of a temporal scenario

The scheduler now creates TCP control connections to each media server involved, or, more precisely to their SMSCs, since these are created by their corresponding media server upon

creation of the control connection. It uses the connections right away to send the delivery schedule to the SMSCs invoking `sched_create_stream()`. When this is done, the scheduler is paused until it receives a signal from the GUI indicating that the user pressed PLAY, i.e., that the playback should be starting.

During the delivery of the data, the scheduler will not have any other responsibility than coordinating the CMSCs and the SMSCs. When commands are pressed by the user: PLAY, PAUSE, or STOP (as well as the commands FAST-FORWARD and FAST-REWIND if they were implemented), the scheduler receives the appropriate signal from the GUI and is responsible for signaling all the MSCs in return: signals are sent to the CMSCs, while a RPC is issued to each SMSC (see Figure 3.6.1.b). Figure 3.6.1.c shows the typical cascade of signals that occurs for each user interactions from the GUI down to the media players, with the example of the STOP button been pressed.

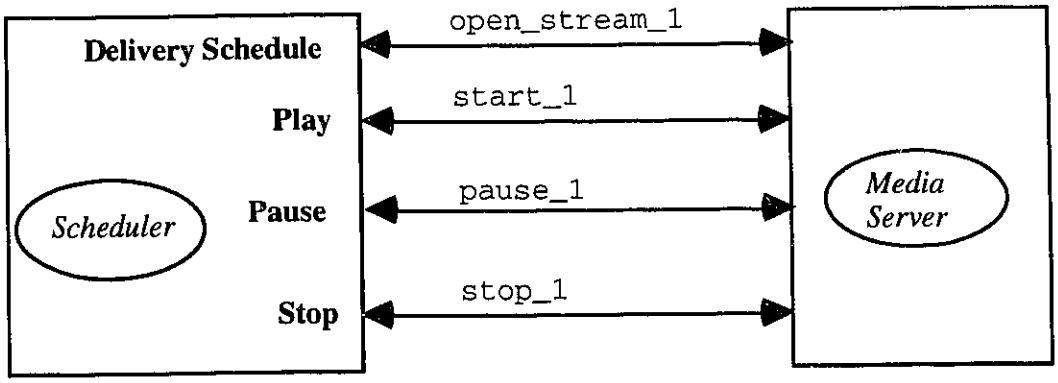

Figure 3.6.1.b RPCs between the scheduler and each media server

In this particular case, the scheduler has to ensure —as well as all the other processes— that all allocated resources are indeed de-allocated and that its child processes terminate properly, before terminating itself, so that the session is ended cleanly.

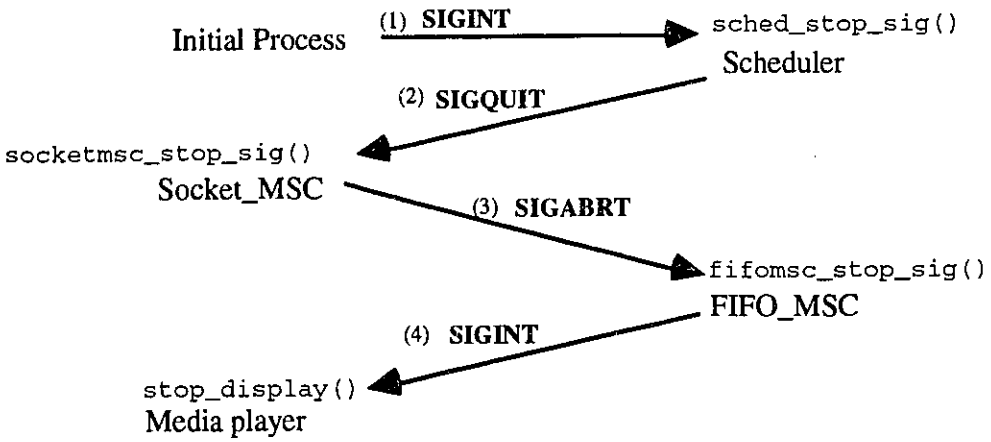

Figure 3.6.1.c Signaling STOP between processes

3.6.2 The Client Media Synchronization Controllers (CMSCs)

In the theoretical approach [Lam94.5, Lam94.6], the synchronization of each medium at the user's workstation was dealt by a single CMSC process. For practical implementation reasons, the CMSC has been split into the Socket_MSC and the FIFO_MSC. Indeed, the CMSC must both read the incoming data on the socket opened to the SMSC and write these data at the scheduled time to the FIFO in order to pass the media objects to the display process. Since the operations of reading and writing might happen concurrently, the implementation with a single process for the CMSC would have to decide of a way to alternate between reading on the socket and writing to the FIFO without creating data starvation in the FIFO when data should be displayed, or data loss on the socket because no data was read for a long time (thus filling up the buffer of the socket). Therefore, we have decided, for simpler implementation, to have one process deal with each of the two aspects mentioned above, and thus let the Unix' CPU sharing policy solve the problem for us. This is of course not an optimal implementation. We have therefore a Socket_MSC process whose action is only to go and read data from the socket, and a FIFO_MSC whose action is to perform most of the functionality of the MSC: send the media

objects to the media player at the scheduled time, and, if necessary, perform the recovery algorithm.

We will present here the actions that the CMSC process performs before it is split into the FIFO_MSC and the Socket_MSC:

- Each CMSC has been created, i.e., forked, by the scheduler. Thus it inherits all the variables declared in the scheduler process, among which are the shared memory where the shared schedule will be put, the 'ready_to_start' semaphore set that will be used to signal the arrival of the media objects, and the array called "node" containing all the information about the media objects, including their presentation schedule (more precisely, the off-set to the presentation starting time).

- From `node`, it derives `obj_list`, the list of its own media objects, using: `init_object_list(2)`.

- It resets the flags `ready_to_start` with `reset_ready_to_start ()`.

- It creates and sets up a stream, a TCP socket between the appropriate SMSC and itself.

- It initializes the buffer for the media data, in shared memory, with `init_buffer(2)`.

- It eventually forks the FIFO_MSC. At this point the CMSC process becomes the Socket_MSC process and the child process becomes the FIFO_MSC.

The Buffer

Before describing the Socket_MSC and FIFO_MSC processes, it is useful to have a better view of how the data sent by the SMSC are stored by the Socket_MSC, and then passed to the FIFO_MSC.

For each media type, the two components of the CMSC share a common buffer. Since they are two different processes, this common buffer must be placed in shared memory. This buffer is a set of *MEDIA*_MAX_BUFF buffer units, where *MEDIA*_MAX_BUFF depends only on the medium. Each buffer unit can store up to *MEDIA*_UNIT_SIZE bytes, where *MEDIA*_UNIT_SIZE varies with the media type only: TEXT_UNIT_SIZE,

VIDEO_UNIT_SIZE and AUDIO_UNIT_SIZE are defined as 160 bytes, 5 kbytes, and 1 kbytes, respectively. However, any of these constants can be modified at any time (the prototype needs to be recompiled afterwards) for values that would have shown to be more appropriate.

To each unit is associated a number of bytes and a flag. The flag indicates whether the unit has been written or not. The Socket_MSC will write data in a unit only if this unit has not been already written, i.e., if written_flag is FALSE. Otherwise, it will wait until the unit is being emptied by the FIFO_MSC (see Figure 3.6.2.b). After writing data in a unit, if this unit is full, the Socket_MSC will set the flag to TRUE. Otherwise it will continue to write the same unit until it is full, unless the end of the media object is reached.

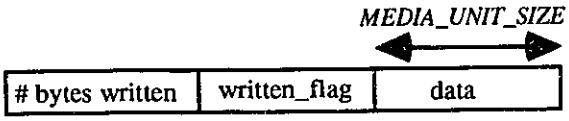

Figure 3.6.2.a Structure of a buffer unit for media objects data

The FIFO_MSC will read data from a unit only if the unit is written already, i.e., written_flag is TRUE. Otherwise (there is no data in the buffer yet) it waits for the Socket_MSC to fill up the unit. After reading a unit, it sets the written_flag back to FALSE (see Figure 3.6.2.c).

The number of bytes written indicates how many bytes have indeed been written in the unit. This way, if the Socket_MSC has not filled up a unit, it knows exactly how many bytes remain to be put in the current unit. The FIFO_MSC uses this number to know how many bytes can be written from the current unit to the display process; there are generally *MEDIA*_UNIT_SIZE bytes, except for the unit containing the last bytes of an object, if the number of bytes of an object is not a multiple of *MEDIA*_UNIT_SIZE.

In addition, the management of the buffer gives this buffer a circular structure: when,

either the Socket_MSC of the FIFO_MSC reaches the last unit of the buffer, which is implemented as an array of buffer units, it goes back to the first unit of the array, allowing the re-use of previous units in a circular fashion.

To add more flexibility in the buffer management, a buffer with dynamic allocation should be preferred to the actual implementation. It would allow the buffer to be expanded at run-time when necessary. However the AIX Base Operating System, version 3.2, does not support dynamic allocation of variables in shared memory.

The Socket_MSC

After its creation (i.e. when the CMSC process becomes the Socket_MSC process), the Socket_MSC process is ready for the document presentation to start. It pauses until it receives a start signal from the scheduler (that indicates the user has pressed PLAY). When the presentation is started, it executes N_i times (N_i being the number of media objects in the document for media type i) the loop in which it calls the routine `recv_data(5)`. This routine does the following:

- It reads data from the socket into the buffer until the number of bytes read is equal to the length of the object; the data are read by blocks of up to *MEDIA*`_UNIT_SIZE` bytes, depending on the number of bytes already written in the current buffer unit.

- It waits before writing to the buffer when this one is full. This happens if the FIFO_MSC needs to delay the presentation of an object because of the SSP). However, the size of the buffer should be chosen so that no data are lost because of buffer overflow.

- It returns the position of the writing pointer (the index of the unit in the array of units), to keep track of what unit should be used for the next object.

In addition to this routine, the Socket_MSC has three signal handlers `socketmsc_start_sig()`, `socketmsc_pause_sig()` and `socketmsc_stop_sig()` to handle the user interactions: START, PAUSE, and STOP. The START signal handler releases the Socket_MSC from its initial paused state, and allows it to go and read the data on the socket

connection. The PAUSE signal handler interrupts (rec. reactivates) the process of reading when the presentation is going on (rec. paused). Finally, the STOP signal handler ensures that the child process (the FIFO_MSC) terminates properly, before closing or releasing its own allocated resources (socket, shared memory, semaphores...).

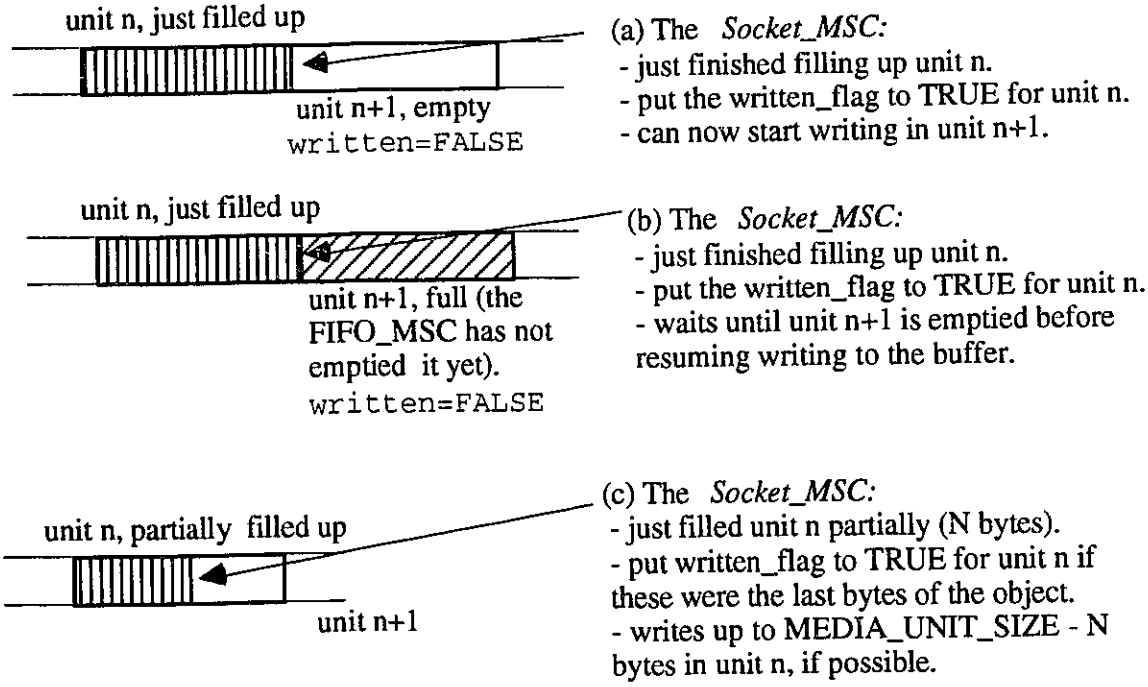

Figure 3.6.2.b Various situations where the Socket_MSC is writing to the buffer

The FIFO_MSC

After the FIFO_MSC is created, it performs the following actions:

- It creates a name for the FIFO (First In First Out) using the `mknod()` system call.

- It forks the display process.

- Upon reception of a signal from the display process which indeed created the FIFO, it opens the FIFO for writing and signals back the display process to indicate the FIFO is opened.

- The FIFO_MSC is then ready for the presentation. It waits for the latter to start by trying to grasp a semaphore that is released by Socket_MSC only when it reads its first bytes on

the socket, after the PLAY button has been pressed by the user.

The FIFO_MSC process function needs then to read the data from the buffer to the FIFO (i.e., pass the data to the display process) according to the shared presentation schedule, and at the same time to perform the stream synchronization protocol (SSP). It executes N_i loops (as many as the number of media objects of this media type).

The presentation is starting with a certain number of media objects that should be displayed in parallel. In relation to all the other FIFO_MSCs, there will be a FIFO_MSC process, named $FIFO\_MSC_{first}$, whose first object will arrive first, and another FIFO_MSC, named $FIFO\_MSC_{last}$, whose first object will arrive last. In the first loop, $FIFO\_MSC_{first}$ accesses the shared presentation schedule first, and has therefore to initialize it. Indeed, before any data has arrived to any CMSC, the presentation schedule only contains the scheduled off-set to the presentation starting point, for each object. The presentation starting point is set at this initialization moment t_{init}. The initialization consists of adding the current time t_{init} to each off-set value. When this is done the 'ready_to_start' semaphore of this object is set to 1. For each object that a FIFO_MSC is controlling, it will have to check what objects of other media types are to be presented in parallel with its current object, and whether the object has already arrived or not by checking the corresponding ready_to_start' semaphore. Each other FIFO_MSC that must present its first object in parallel with the first object of $FIFO\_MSC_{first}$ will set the 'ready_to_start' semaphore of its first object to 1. The last of these FIFO_MSC, $FIFO\_MSC_{last}$, is auto-determined by the fact that the 'ready_to_start' semaphores of all the other first objects are set to 1. $FIFO\_MSC_{last}$ will access the shared schedule at a time $t > t_{init}$, and it will update the shared presentation schedule by adding the time $T = t - t_{init}$ to all the scheduled presentation time. The presentation of the first object of each media type does not begin before the last one has arrived. If only one object has to be presented initially, it is sent to the display unit right after the initialization of the shared schedule.

In each of the N_i-1 remaining loops, the FIFO_MSC goes through the following steps:

- It checks the current time before reading the data from the buffer, and compare it with

the presentation schedule. If the object is too early, the FIFO_MSC has to stay idle until it is time to read the data from the buffer and write them to the FIFO according to the schedule. If the object is late, the FIFO_MSC updates the presentation time of the object with the current time in the shared schedule. It then checks the time difference Δt with the arrival time of the first of the objects that must be displayed in parallel. This time difference is later compared to two threshold values (the rescheduling threshold and the abort threshold) to decide of the actions to be taken:

- i) Δt is smaller than the rescheduling threshold. It this case no special action is taken. The asynchrony Δt will not be detected on the display since the rescheduling threshold is very small compared to the synchronization QoS parameters. This rescheduling threshold is being implemented to avoid the operation of rescheduling when it is unnecessary.
- ii) Δt is larger than the rescheduling threshold, but smaller than the abort threshold. The FIFO_MSC then perform the SSP by adding the intentional delay Δt to the presentation time of any object of any media type that should be presented after the current object.

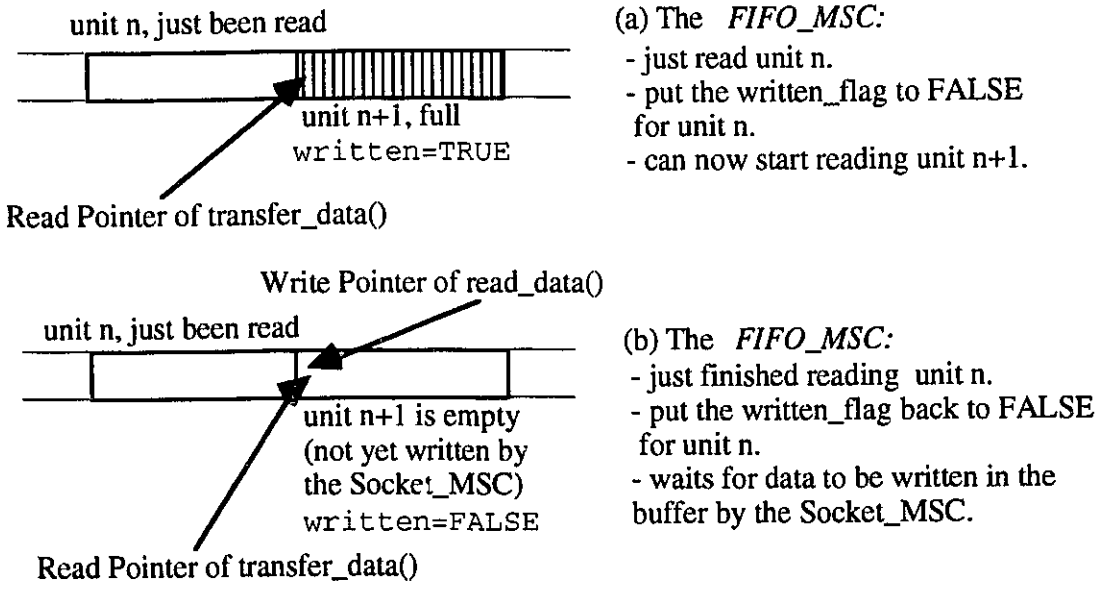

Figure 3.6.2.c Various situations where the FIFO_MSC is reading from the buffer

- iii) Δt is larger than the abort threshold. The FIFO_MSC process has to signal the initial process so that the presentation is aborted. Subsequently, the QoS parameters of the data connection(s) that is (are) causing the problem should be re-negotiated. New data connections should be re-opened, and the presentation could start all over again or resume, depending on how far it previously went.

- When this is done, and the presentation was not aborted, the FIFO_MSC calls the routine `transfer_data(5)` which reads the media object unit by unit. This routine is reading from the buffer and writing to the FIFO in a loop that terminates when the number of bytes transferred is equal to the length of the media object. If a gap occurs within the transmission of a media object, no data will be available to be read in the buffer. The FIFO_MSC will detect a gap since its 'read pointer' will catch up with the 'write pointer' used by the Socket_MSC. The FIFO_MSC will wait until data arrive, but no longer than a time period equal to the abort threshold.

- When a whole object has been read, the function returns the unit of the buffer pointed to by the 'read pointer'.

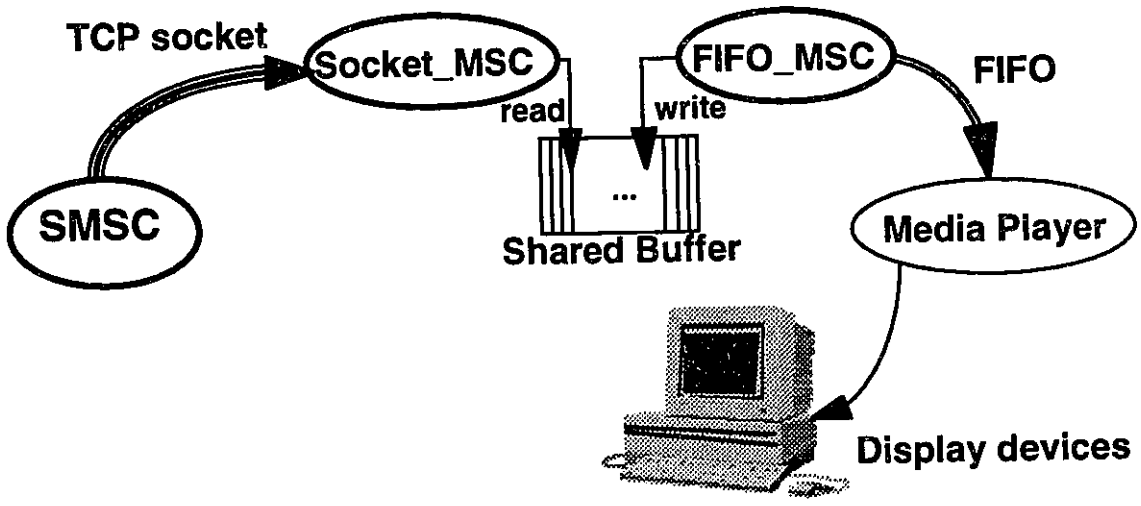

Figure 3.6.2.d Data flow from the media server to the media player

In addition to the main routine, the FIFO_MSC also has signal handlers for PAUSE and STOP (but not for PLAY since it uses a semaphore instead). When the user selects PAUSE, the FIFO_MSC receives a signal from the Socket_MSC and executes the routine contained in the corresponding signal handler:

- 1) If the presentation was going on, it records the time t_{pause} when the signal was received, stops writing data to the FIFO, and then waits for the presentation to resume.
- 2) If the presentation was already paused, it records the time t_{resume} when the signal was received. It then tries to update the shared presentation schedule by adding $t_{resume} - t_{pause}$ to the media object that remain to be presented. Only one of the FIFO_MSC should update the presentation schedule, therefore all of them try, but one the first to access the shared schedule indeed performs the operation. When the user selects STOP, the FIFO_MSC ensures that the display process terminates properly before releasing its own resources and terminating itself.

3.6.3 The Display Processes and Media Players

The Display Processes

In the implementation, we have dealt with three types of media: text, audio and compressed video (coded in the MPEG1 format). For each of them, a media player has been implemented as an independent program: `text_play`, `audio_play`, and `mpeg_play`. For example, a public domain MPEG Decoder has been used for the video. Each of these media players is 'called' by the corresponding display process.

Each display process is forked by its corresponding FIFO_MSC, and then performs the following steps:

- It creates the FIFO that is used by the FIFO_MSC to send the media objects, then opens it for reading, and signals its parent process (the FIFO_MSC) that the FIFO is open.

- Right after, it starts the media player by calling the `execlp` system call, which executes

a new program in the calling process. The necessary parameters are passed when the call is done: name of the file to execute (the media player executable), file name descriptor for the FIFO. In the case of the video display process, we also need to pass the rate of the video (in frames per 100 seconds).

When this is done the display process has a new process-image and becomes the media player. The media player then displays the data that arrive on the FIFO until there is no more data in the FIFO or until the presentation is stopped by the user.

The MPEG Decoder/Player

We have used the MPEG Video Software Decoder version 2.0 developed in the Computer Science Division of the University of California at Berkeley. We have brought the following changes to the source files to add few functionalities to the player.

The most important functionality added consists of a routine that has been implemented to control the frame rate of the video. The frame rate F, in frames per seconds, must be passed as an argument when the player is started. The rate control is implemented by having the decoder spend exactly 1/F seconds decoding a frame. Obviously, if F is chosen too large, the decoder will try and decode the video stream as fast as possible. In the opposite case, i.e., when F is small enough, the decoder will decode a frame in a time t_{decode} smaller than 1/F second. In this case, the decoder is forced to stay idle 1/F - t_{decode}, before decoding the next frame.

The other changes that have been done concern the handling of the user interactions. As soon as the MPEG player is started, it is expecting data to its input in order to decode them and display them on the screen. Therefore, we do not need to handle the starting of the playback. However, the ability to pause the decoding process needs to be added, and interrupting the decoding and display of the data slightly modified.

The video data

Since the MPEG decoder is a software decoder, we are more limited in the frame rate that the

decoder can guarantee at its output, compared to a hardware decoder. In order to maintain a frame rate of 15 frames per second, we needed to reduce the size of video frames to 160x112 pixels. And even with this frame size, we were not able to get the expected 15 frames per second with any type of coding pattern for the MPEG stream. Our video clips is specifically coded using the pattern IBBPBBPBBP, where I-frames are *intra*-frame pictures coded independently, P-frames are *predictive* pictures coded with reference to the past I-frame, and B-frames are *bi-directionally predicted* pictures with references to past and future I-frames or P-frames.

The Text Player

The text player is a simple program that reads, upon reception of a signal, data from a text file and displays the text in a scrolling text window. Each time a text object is passed from the FIFO_MSC to the text player, the FIFO_MSC signals the text player which reads the data on the FIFO (which is assimilated to a data file), and the new text object (i.e., the new lines of text) is appended to the text lines that are already in the window.

Here also, the three user interactions PLAY, PAUSE, and STOP, are implemented using signal handlers that are triggered upon reception of the appropriate signal.

The text data

During the presentation, the text objects are appended to the text lines that are already in the scrolling text window. The text in such a presentation represents subtitles of the voices that are heard in the audio stream. The subtitles, unlike the audio or video stream, do not represent a continuous medium. They have been synthetically synchronized with the voice when the multimedia document's scenario has been created, in such a way that the first word of each text object would be displayed right at the same time when this word would be spoken in the audio stream. In order for the user not to be annoyed, we have limited the visibility of the subtitles to two lines, and the text is being scrolled in such a way that only one text object can be seen at a time, i.e., the text object that is synchronized with the voice in the audio stream. To achieve

this, the number of text lines per text object has been limited to two, as well as the number of visible lines in the scrolling text window.

The Audio Player

It has been build using the audio player given as an example module of the AIX Ultimedia Services/6000, version 1.1.0. We have modified the program accordingly: signal handlers for STOP and PAUSE have been added. There was no need for a PLAY signal handler since the audio player is ready handle the data and send them to the audio output of the workstation soon after it has been started.

The audio player is implemented with a loop that sends the audio data second per second to the audio device. The audio device is guaranteeing that the audio data are used a the proper rate, depending on the format of the audio data. Thus, we did not need to take care of the audio data rate control.

The audio data

The audio objects that we use have the following characteristics : 1) the file is in a *raw* format; 2) the data is in *mu-law* format; 3) *sample rate* is 22050 bytes per second; 4) the *bits/sample ratio* is 8; 5) the type of the audio is *mono*.

According to the above information, a second of audio data will be 22050 bytes long.

3.7 The Graphical User Interface at the Multimedia News Client

Most of the windows that are popped up on the client workstation's screen by the multimedia news application have been generated using the AIXwindow Interface Composer version 1.2 (AIC). AIC is a software that is used to build graphical user interfaces (GUI). It helps the

software developer by allowing him to create, generate and test code for any GUI in a very user friendly manner. The AIC-built windows are all generated by the initial process, or Multimedia News Client, in the set-up phase of a session, before the retrieval of a news article begins. By opposition, the windows generated by the media players have directly been implemented, using X Window for the MPEG decoder, and Motif programming for the text player. Our GUI is therefore made of three different X-applications, i.e., three 'graphical interfaces' that have different application contexts, which required to give a special attention to the positioning of each of them in the display. This problem is addressed at the end of sub-section 3.7.2.

3.7.1 Functionality of the Graphical User Interface

The GUI provides the user with four main functions:

(1) *Selecting an article.* The user requests from the Multimedia Database server (Database server) the list of articles that can be viewed. This list is retrieved from the Database server and displayed on the screen, in the Main window (see Figure 3.7.2.b). The user can then effectively select an article.

(2) *Getting more information on a specific document.* Once an article has been selected, and before starting a possible presentation, the user can get additional information about the article (duration, author, keywords...)

(3) *Changing playback options for the document to be viewed.* The user can select both the network type and transport protocol that will be used for the data retrieval. For testing purposes, we allow the user to enable or disable the stream synchronization protocol (SSP), and when the SSP is enabled, to set various re-scheduling parameters, in order to visualize their influence on the quality of the synchronization.

(4) *Presenting a document.* When the user has chosen the document and the options, the retrieval

of the article can start. The GUI provides the user with the common functions of a VCR, such as PLAY, PAUSE, STOP (see Figure 3.7.2.d). However, scanning the presentation forward or backward is not yet possible at this stage of the implementation.

3.7.2 Different Windows and User Interactions

The MultiMedia News Window (or Main Window)

The *Main* window shown in Figure 3.7.2.b appears as soon as the Multimedia News Client application is started at the user's site. The function of this window is to allow the user to select an article. The user connects to the Multimedia News On-Demand service by selecting the button `SELECT ARTICLE`, which triggers the following three-steps callback routine:

i) The RPC server at the client's site connects to the RPC server at the Database server's site by invoking the `MMDB_Open()` RPC routine (see Figure 3.7.2.a). The server accepts the new client if the user is registered in the database.

ii) A second RPC routine, `MMDB_SelectAbstracts()`, is invoked that requests the list of documents available in the database. The Database server returns a linked list of abstracts for each article, where each abstract is a set containing the article id and the title of the article.

iii) As soon as the list is returned, the titles are displayed in the *Main* window in the increasing order of their article id.

The user can then select an article by clicking on its title with the mouse pointer. The first article of the list is the choice by default. Once an article is selected, the user can:

(1) change the options of the service,

(2) get additional information on the selected article,

(3) or initiate the presentation of the selected article.

(1) To change the options, the user needs only to click on the 'pull down' menu `OPTIONS` and select `PROTOCOLS` to pop up the *Protocols* window, or `THRESHOLDS` to pop up the *Thresholds* window. Further details about these two windows are given later in this section.

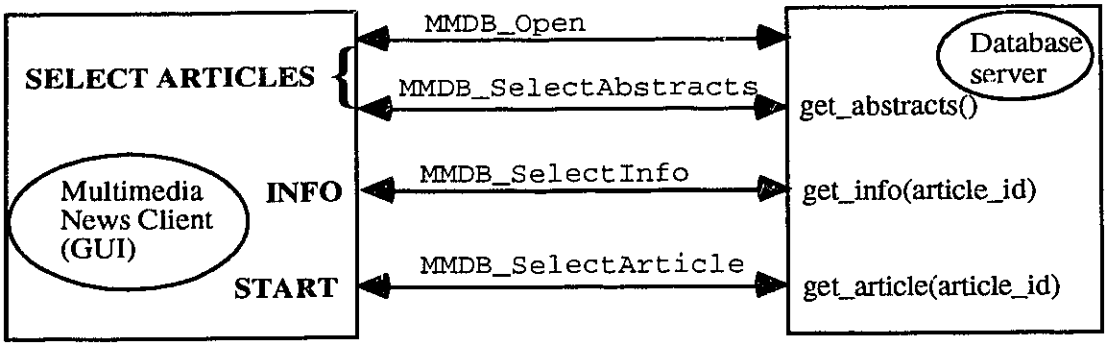

Figure 3.7.2.a RPC routines between the client workstation and the Database server

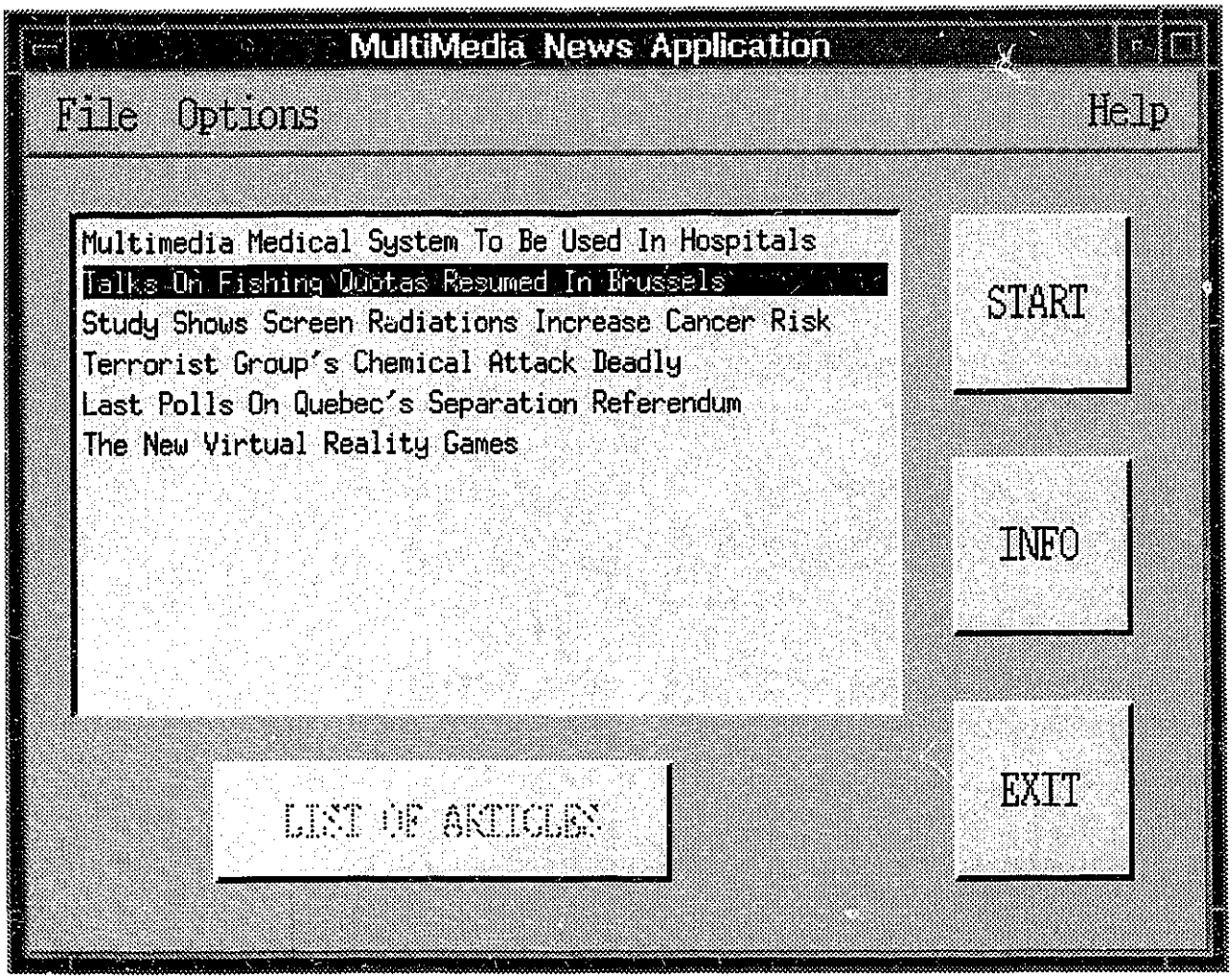

Figure 3.7.2.b The Multimedia News Application window

(2) To get more information on the document, the user clicks on the button `INFO`, thus triggering a callback function that issues the `MMDB_SelectInfo()` RPC routine to the Database server, with the article id as input parameter and a linked list of abstracts for the output, and then pops up the *Information* window containing all the information on the article that was selected (see Figure 3.7.2.c).

(3) To start the actual presentation of the article, the user clicks on `START`.

- The Multimedia News Client process pops up the *Player* window.

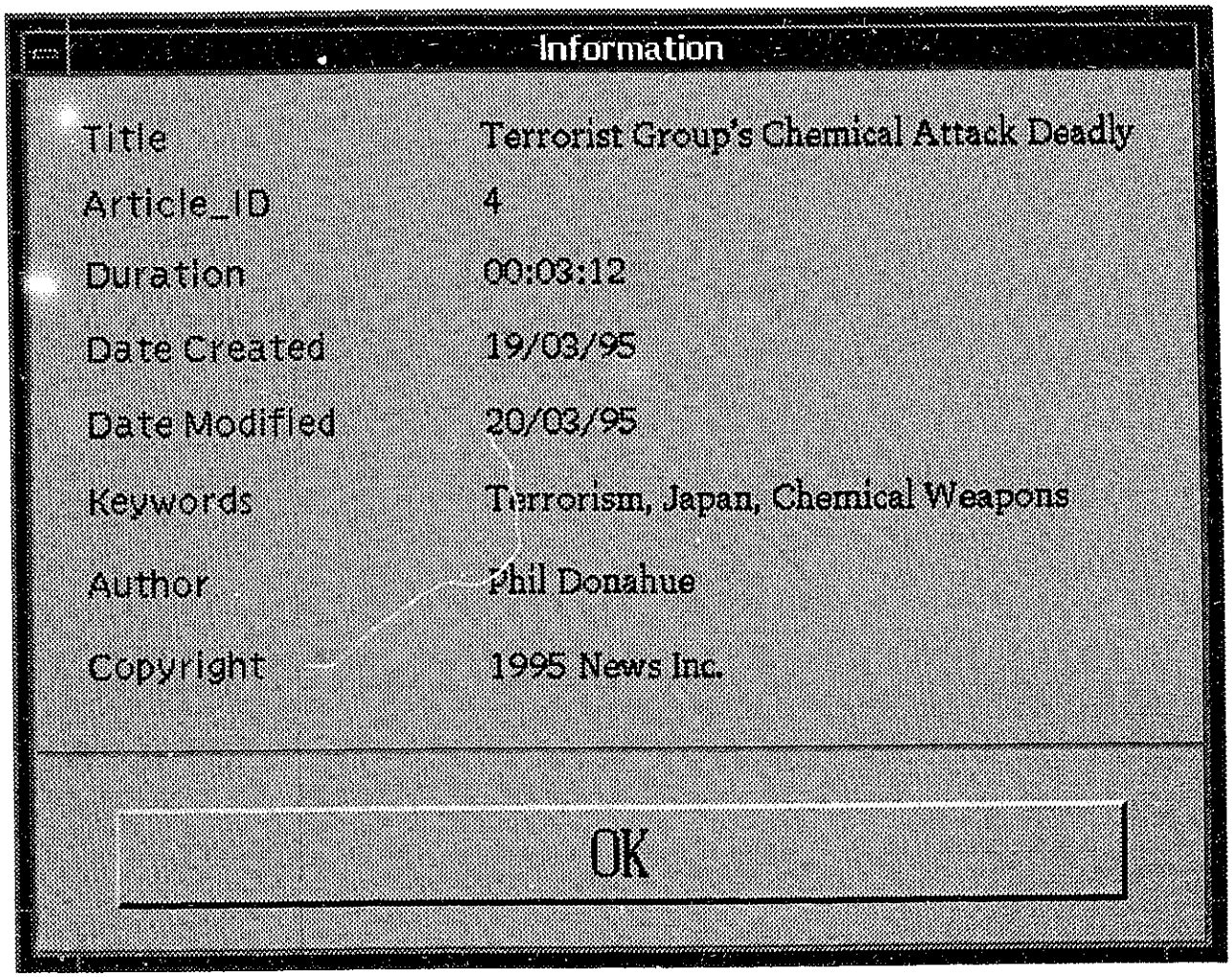

Figure 3.7.2.c The *Information* window

- It issues the `MMDB_SelectArticle()` RPC routine to the Database Server, which receives the article id as input of the RPC, and returns a linked list that only contains the QoS parameters for the selected article, its temporal scenario, and the length in bytes of the various media objects.

- It finally forks the scheduler, which automatically inherits the above information.

The Player Window

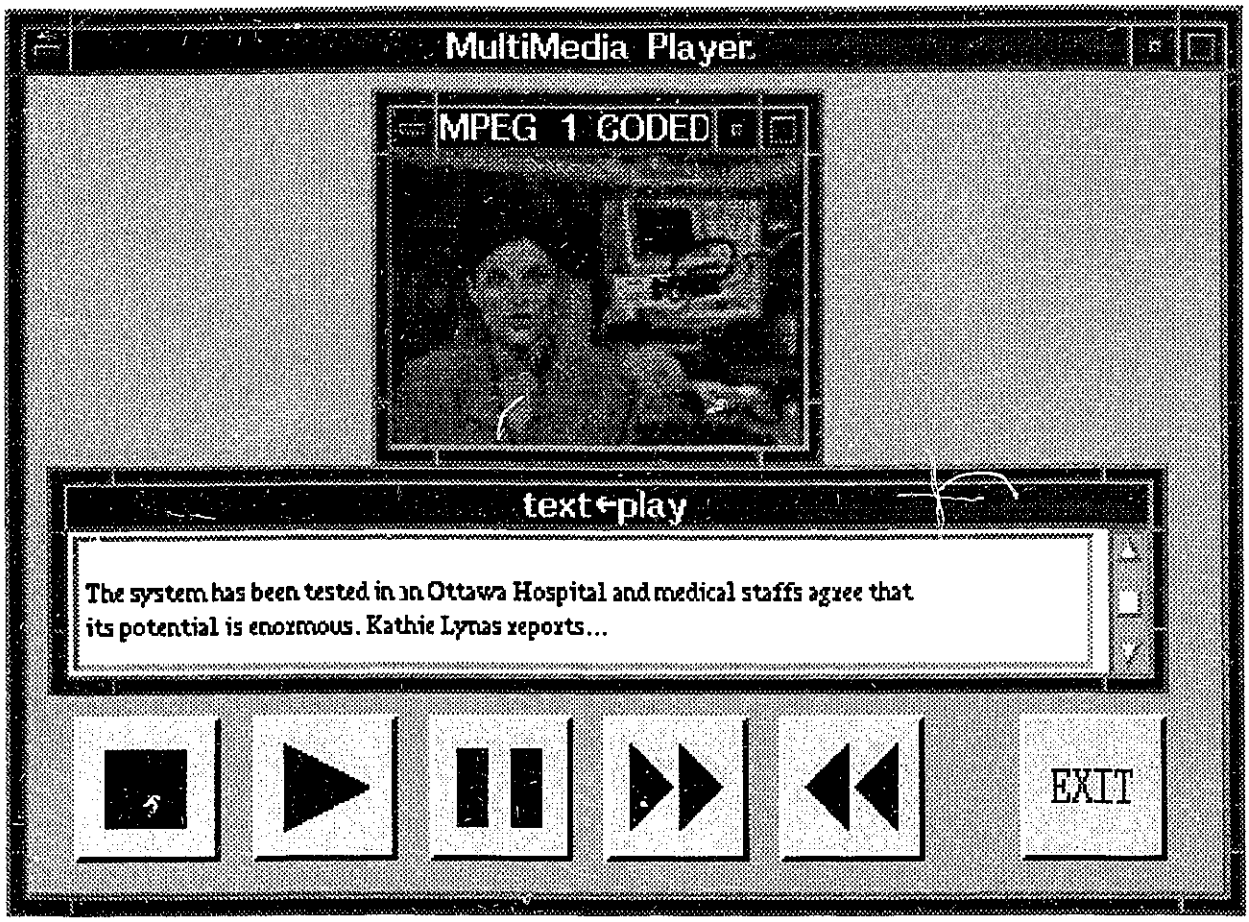

Figure 3.7.2.d The *Player* window with the windows of the text and MPEG1 players

As we just explained, the *Player* window appears right after the `START` button has been pressed. The creation of the scheduler process is the first step into setting up the whole synchronization

control software and the media players at the user's workstation, as explained in section 3.6. Once all the necessary processes have been created and gone through their initialization and resource allocation phases, the *Player* window is rendered active. At this point, all the windows used for the display of visual media —text and video, if they are involved in the document— have already appeared on top of the *Player* window (see Figure 3.7.2.d). When the *Player* window becomes active, the buttons and associated actions of this window are being enabled. Out of the six buttons, only `PLAY`, `PAUSE`, `STOP` and `EXIT` are implemented. `FAST FORWARD` and `FAST BACKWARD` are not functional, since no callback function is associated with them.

When either `PLAY`, `PAUSE`, or `STOP` is selected, the corresponding callback function, respectively `PlayArticle()`, `PauseArticle()`, or `StopArticle()` is executed. Indeed, the appropriate signal is send to the scheduler, which takes care of passing the information to the CMSCs, using signals, and to SMSCs, using RPCs (see Figure 3.6.1.b). When `STOP` is pressed the callback function additionally brings down the *Player* window. The user remains with the *Main* window and can select another article to view or exit the Multimedia News service.

The Protocols Window

The *Protocols* window allows the user to choose the type of network that interconnects the client workstation and the server workstation, and the type of protocol that we use for the Transport layer. Two columns of 'toggle buttons' allow the user to make a selection, one for the network type, the second for the Transport layer type:

- Three network types are available and can be chosen in the `NETWORK` column: two LANs, Ethernet and Token Ring, and one B-ISDN ATM network. Ethernet is the selection by default.

- The choices offered in the `PROTOCOL` column are: TCP/IP, UDP/IP, XTP (the eXpress Transport Protocol) and ST-II. TCP/IP is the default selection and it is the only transport protocol that is functional at this point of the development. Further work is going on to enable

the use of XTP and ST-II.

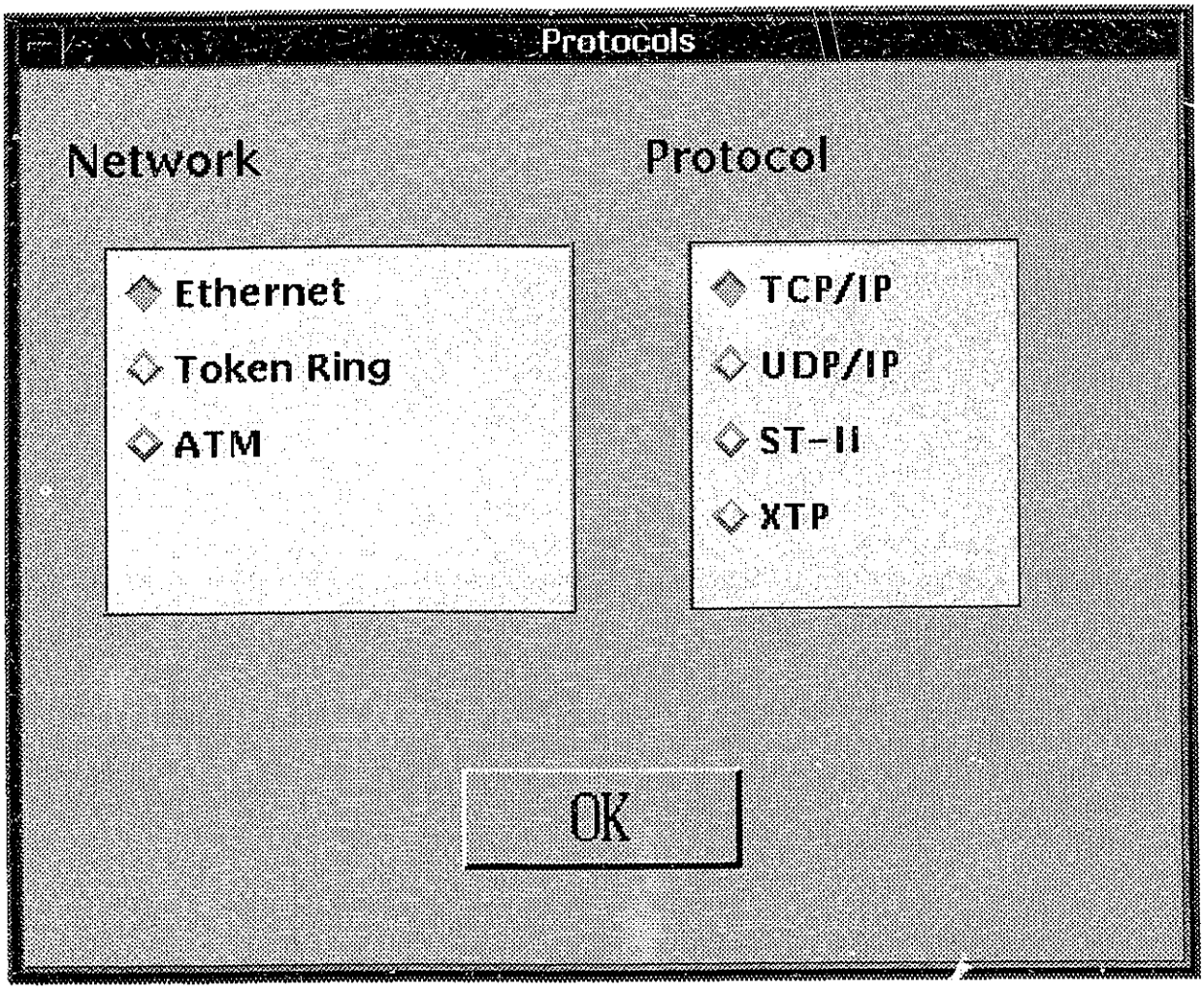

Figure 3.7.2.e The *Protocols* window

The Threshold Window

Through the use of 'toggle' buttons, the GUI allows the user to enable or disable the recovery process implemented in the CMSCs (more exactly in the FIFO_MSC). The user may also decide to add a forced delay for a certain medium and customize it using a scaling bar.

Additional scaling bars are provided to vary the threshold values for the SSP: re-scheduling threshold and skew parameters can be adjusted by the user. The skew parameters are

used to determine the abort threshold discussed in section 3.6. The skew parameters concern both lip-synchronization and voice-to-subtitle synchronization. The default values for the skew parameters are these given by Steinmetz in [Ste93]. None of these options are functional yet.

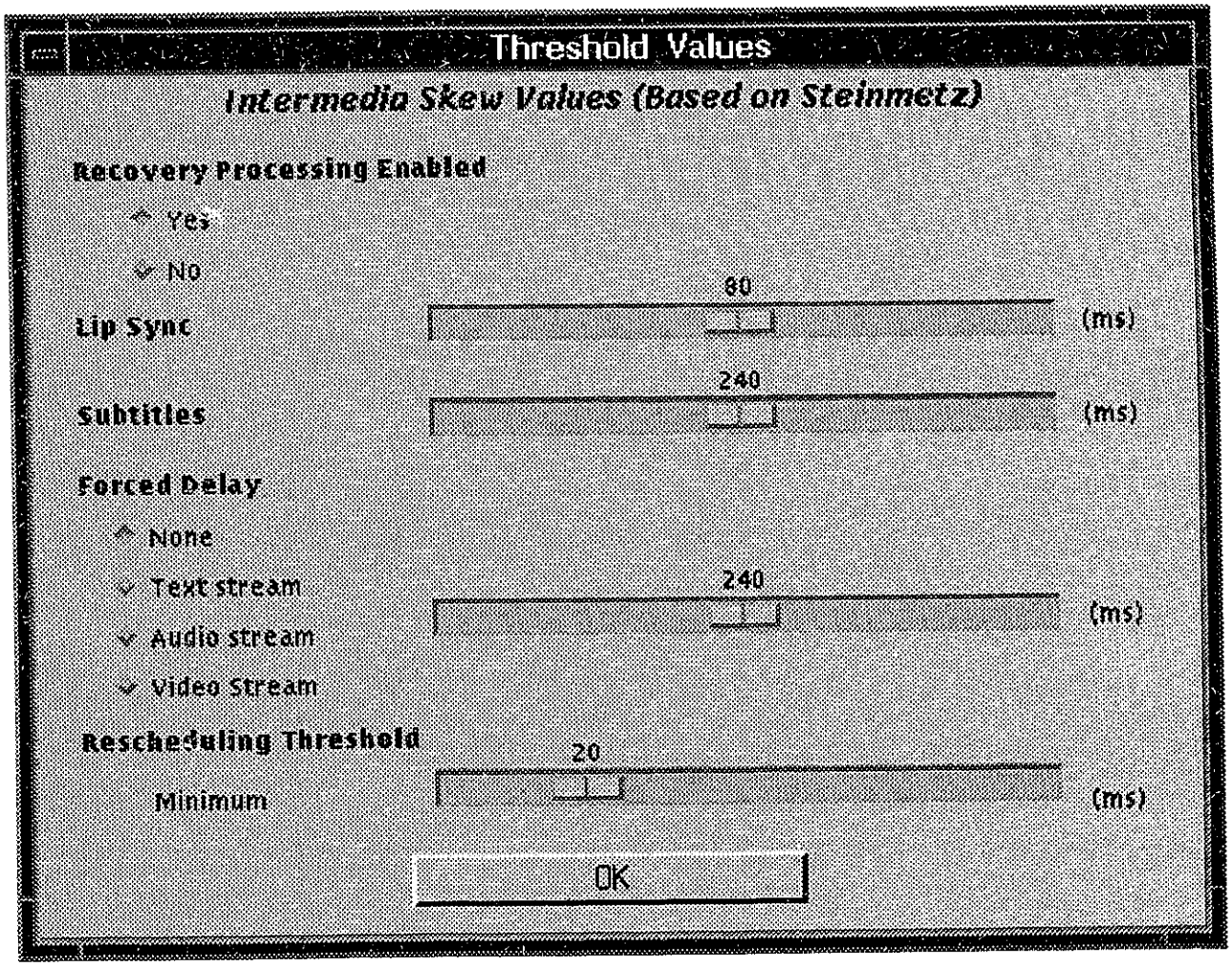

Figure 3.7.2.f The *Thresholds* window

The Information Window

The *Information* window simply displays additional information on the selected article. This information is: the article's title, its id, its duration, its date of creation, the date of the last changes to the document, keywords, its author(s) and the copyright. It is possible to get rid of

certain information field while adding new ones, such as the types of media involved, a summary of the content...

The Video and the Text Windows

The video and text windows are popped up just after the *Player* window, when the user has pressed START in the *Main* window, if the corresponding media is part of the document to be displayed. Figure 3.7.2.d shows the two media windows placed on top of the *Player* window, during the retrieval of a document.

The text and video players have been implemented as independent programs. Therefore, each of these display windows is managed independently from the other, and independently from any other window generated by the Multimedia News Client, such as the *Player* window. The two windows have their own application context, i.e., for the X-Server running on the user workstation these window belong to two different X-applications. For these reasons, whenever the user clicks on a button of the *Player* window, the latter will logically come on top of any window that is not in the same application context, which is the case for the video and the text windows. This problem has been overcome in a simple way by manipulating the general default settings of the user's Window Manager: for a window to come on the top of the window stack, i.e., in the first plan of the screen, it is necessary to click on the title bar of the window. Pressing PLAY or PAUSE then does not affect the disposition of the windows. We also ensured that the display windows would initially appear only after the *Player* window did.

3.7.3 Exiting the User Interface

If the user wants to exit the Multimedia News On-Demand service, he can do it at any time by click on EXIT which is available on the *Player* window as well as on the *Main* window. Exiting the application will cause the presentation to end up immediately. All the connections to the media servers and the Database server are closed and all the windows on the screen disappear.

When the user exits the application, the Multimedia News Client signals the processes participating to the synchronization control and the presentation through the scheduler and waits for all of them to terminate properly before terminating itself.

3.8 Summary and Observations

In this chapter, the implementation of a Multimedia News on Demand prototype was presented. The synchronization control architecture on which the prototype is based was shortly explained. The IPC facilities used within the prototype were described. The operations of each of the software components of the system's prototype (database server, media servers, different modules of the user application, GUI at the client's workstation) were detailed, outlining the various inter-process communications and possible user interactions affecting them. Finally, six "snapshots" of the various GUI's windows were presented, showing the different interactions that the user can have with the system.

The prototype has many limitations, mainly three, that arise from technological deficiencies. The first one, visible to the user, is the size of the video display. It is limited, as previously stated, to 116x160 pixels because of the use of a software decoder to decode the MPEG-1 encoded video clip, which is very computationally expensive. The use of a MPEG-1 hardware decoder should allow us to display larger video images and increase the frame rate —now at 15 frames per seconds— to 30 frames per second. The second limitation is the fact that no transport protocols implemented right now can provide us with guaranteed QoS parameters, such as the delays and delays variations experienced on the network. These are therefore hard-coded at this point. Considering the fact that the synchronization control is based on these guaranteed QoS parameters, this is a limiting factor when trying to estimate the quality of the synchronization control schemes from the prototype. The third shortcoming comes from

the use of an operating system that does not provide us with real-time characteristics such as dynamic priority scheduling, bounded job's completion time, and light-weight signaling which prevent the synchronization system from achieving a precise synchronization.

Finally, the ATM-based network that connects the database and media servers to the client workstation is, at this point, a local connection through the Newbridge VIVID switch installed in the MCRLab. Later experiments using an ATM network are expected to be done on OCRInet (Ottawa Carleton Research Institute network). A non-local connection will be simulated thanks to a loop-back established within a Newbridge 36150 switch localized at CRC (Communication Research Center) through a mapping of virtual channel identifiers (VCIs) as shown on Figure 3.8.1. A computer installed at the level of the loop-back could then introduce monitored errors, jitters, etc., within the data transmissions, which would then allow us to better appreciate the behavior and robustness of the synchronization control schemes.

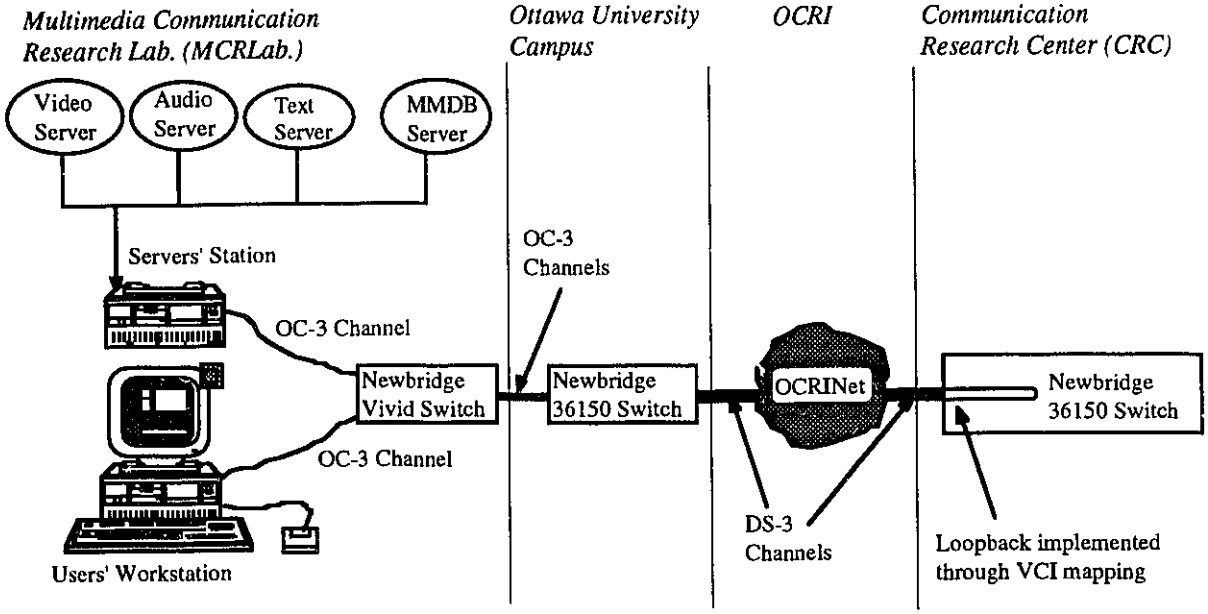

Figure 3.8.1 Set up of an ATM connection for future testing purpose

Chapter 4

Performance Modeling and Evaluation of the Stream Synchronization Protocol

The performance modeling and analysis of the synchronization architecture is indeed focusing exclusively on the SSP which is performed at the user's workstation during the presentation of a multimedia article. The stream scheduling protocol is not included in our study. The model of the SSP is restrained to the case of lip-synchronization between an audio stream and a video stream. DSPNs are used to schematize the model, and simulations using the UltraSAN [Cou91, San93] tool allow us to get performance measures from the model. The necessity of a buffering delay at the client's workstation to maintain the probability of aborting the presentation of a document under reasonable limits is assessed. Results show the effect of the SSP in preventing the asynchrony between the different media streams from been rampant, as well as quantitative measures on the required buffering delay. Indicative results are given, such as the number of synchronization errors during presentations, or the influence of the media objects' duration, and the required buffer sizes.

4.1 Petri Nets

Standard Petri Nets

Petri Nets (PNs) are a graphical and modeling tool developed by C.A. Petri. They are a very useful and effective means of describing and analyzing the dynamics of systems in which concurrent and synchronized actions take place between different entities [Mar84, Mar86].

A PN consists of a set of *places*, a set *transitions*, and a set of *directed arcs* connecting places to transitions and transitions to places. Places are graphically represented as circles, transitions as bars, while lines with an arrowhead depict the arcs. When an arc is connecting a place to a transition, it is an *input* to the transition, and it represents a condition to this event/transition. A place is, on the contrary, an output of a transition if an arc connects the transition to the place; the place symbolizes the result of the event.

Places may contain tokens, thus depicting the holding of the condition represented by the place. The graphical representation of a token is a black dot. The number of tokens in each place defines the state of the PN, also called a *marking*. A transition is enabled when all of its input places contain at least one token. When enabled, a transition can fire (i.e., takes place), thus removing a token from each input place, while putting one token in each output place, causing the PN to reach a new marking. By removing tokens from one place and depositing them in others, the firing of a transition may disable *conflicting* transitions or enable new ones. Basic extensions to PNs are *weighted* arcs, equivalent to many arcs connecting the same place to the same transition, and *inhibitor* arcs, only connecting a place to a transition, which disables the transition if the place contains a token. The inhibitor arc is depicted as a line with a terminating circle.

Stochastic Petri Nets and Further Extensions to Deterministic and Stochastic Petri Nets [Mar84, Mar86, Mur89]

Standard Petri Nets do not include any time concept. Since time plays a fundamental role in the description (delays, throughput...) of the dynamic behavior of a system, it was later introduced in

PNs by associating a time delay to the transition, which delay takes place between the enabling and the firing of the transition. When each transition is associated an exponentially distributed firing time, we are in the specific case of Stochastic PNs (SPNs) [Mar84, Mar86, Mur89].

SPNs are very interesting because they can be translated into a Markov Chain (MC), useful for obtaining the probability distribution of the different states of the MC, needed to calculate various performance measures. However, in order to cope with the state-space explosion of the equivalent MC when the SPN becomes complex, SPNs have been extended to a class of Generalized Stochastic Petri Nets (GSPNs) [Mar84], where transitions are either timed (and drawn as a box) or immediate (and drawn as a thin bar). A timed transition has a random, exponentially distributed, enabling time. On the other hand, immediate transitions fire in zero time once they are enabled [Mar84], and they are used to represent a logical control or an activity whose delay is negligible compared to these described by timed transitions. The introduction of immediate transition in SPNs has been shown to reduce the state-space.

Further extensions to Extended SPNs have been done where the timed transition may have an arbitrary distribution [Dug85]. A more restrictive extension gave birth to Deterministic SPNs (DSPNs) [Mar87], where the time associated with each transition is either an exponentially distributed random variable, or a constant (deterministic value); an immediate transition is a deterministic transition with zero enabling time. As shown in [Mar87], in order to keep the underlying stochastic model reasonably simple and thus be able to calculate analytical performance measures, a sufficient condition should be satisfied: in no reachable marking should more than one deterministic transition be enabled at a time. We will see in section 4.3 how this rule influences our performance analysis.

4.2 The DSPN Model of the Stream Synchronization Protocol

In the modeling of the Stream Synchronization Protocol (SSP), we make the following restrictive assumptions compared to the prototype:

- We consider that the SSP is performed on a multimedia document involving only an audio stream and a video stream.
- The model assumes a very strict type of presentational scenario where the audio stream and the video stream are starting at the same time. They are segmented into media objects of constant duration, independent of the media type, so that each audio object is displayed in parallel with a video object, both starting and ending at the same time (see Figure 4.2.1).

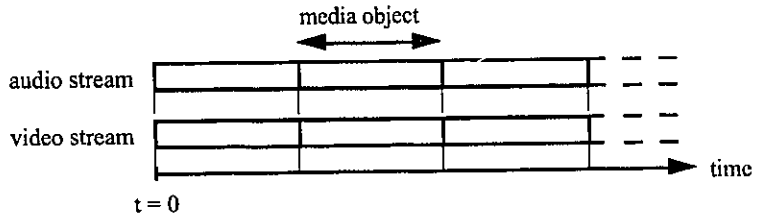

Figure 4.2.1 Segmentation of the audio and video streams.

- The streams are supposed to have an infinite duration in the case of steady state measures, and to last long enough when transient measures are done.
- We assume that each object (represented by a token) is handled as a whole by the SMSCs and the CMSCs. If we refer to our prototype, a video object of 10 seconds is 660 kbytes on average, i.e., slightly more than 5 Mbits. In the case when the transmission is done on the ATM test-bed, with a transmission rate of 45 Mbits, the video object can be sent in roughly 100 ms, not taking into account the packetization of the data into ATM cells and considering that the various protocol layers are able to sustain such a rate. If we compare the 100 ms of transmission to the actual duration of the objects, the handling of each object as a whole for transmission and

reception appears to be acceptable.

- Considering each media object as a whole also implies that gaps can not happen within the presentation of an object, but in between objects only. Any delay that would in reality be introduced during the transmission on an object thus appears to be introduced at the beginning of this object. Any random delay introduced by the network is considered to delay the whole object.

- We consider the various processing times at the user's workstation to be negligible.

- No intra-media synchronization is considered in the model.

Figure 4.2.2 gives a graphical representation of the DSPN model of the SSP and table 4.2.3 provides a short description of each place and transition and their role in the model. Place *obj_gen* together with transition *obj_dur* reproduce the delivery schedule that both servers, *audio_server* and *video_server* should follow as part of the stream scheduling protocol. Place *start_up* and transition *buffering_delay* model the initial buffering delay. Places *timer1-2* and *ABORT*, and transitions *up_thres1-2* and *t_out* participate in detecting when the asynchrony between the two data streams has exceeded the tolerated skew (presence of a token in place *ABORT*). Places *E1-2* and *ERROR* together with transitions *low_thres1-2* and *no_error* participate counting the number of synchronization errors between the two data streams. The rescheduling part of the SSP includes places *start_display1-2*, *end_display1-2*, *A1-2* and *control*, and transitions *display1-2*, *t1-2*, *no_error* and *remove_control*. Places *end_display1-2* are used to provide feedback to transitions *t1-2*, and thus force one medium to wait for the end of the presentation of the other medium's n^{th} object before starting the presentation of its own $n+1^{st}$ object, unless no synchronization error occurred. In the later case, the asynchrony between the n^{th} objects of the two media is less than the delay of transitions *low_thres1-2*, allowing transition *no_error* to fire and put a token in place *control*. The place *control* acts on transitions *display1-2* by modifying the weight of their output arcs: if *control* contains no token, arcs marked (1) have a weight of 1 and arcs marked (2) a weight of 0; in the opposite case, the weights are interchanged. This allows, together with transition *no_error*, to prevent any rescheduling operation when the

asynchrony is smaller than *low_thres1-2*.

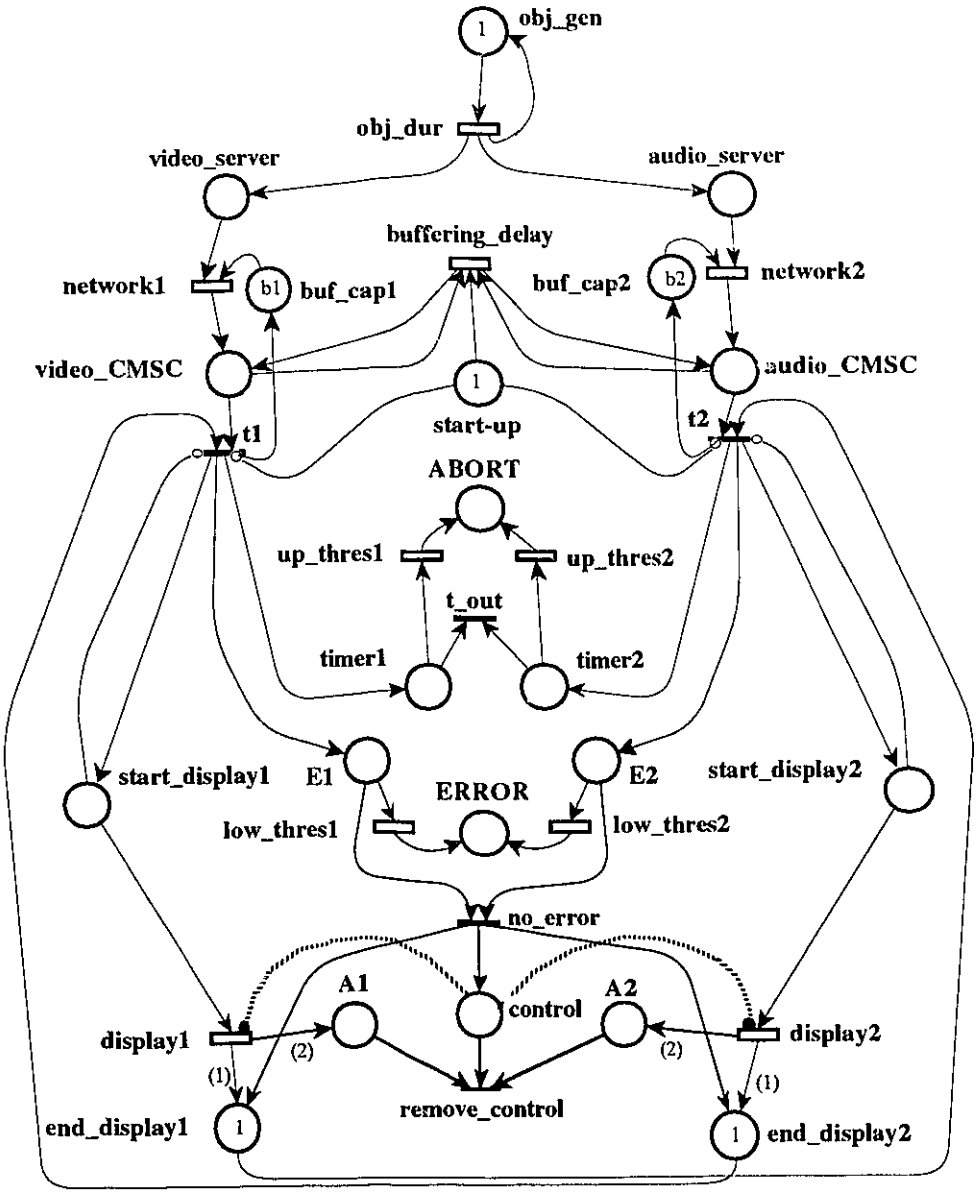

Figure 4.2.2 The DSPN model of the Stream Synchronization Protocol

Name	Type	Description
obj_gen	P	Enables transition *obj_dur* with which it forms the object generator
obj_dur	T	Represents the duration of the objects
video or *audio_server*	P	Represents the video (resp. audio) server
network1,2	T	Simulates the random delays on the video/audio data connections
buf_cap1,2	P	Maximum capacity b1 (b2) of each of the CMSC's buffer
video or *audio_CMSC*	P	Represents the buffer of the video (audio) CMSC
t1, t2	T	Symbolizes the beginning of an object's display
buffering_delay	T	Represents the buffering delay introduced at the user's workstation when receiving the first video and audio objects
start-up	P	Contains one token initially. It enables the *buffering_delay* transition to fire only once, and prevents the first audio and video object from being displayed before the start-up delay is over
start_display1,2	P	Represents the display of a video (audio) object
end_display1,2	P	Symbolizes the end of the display of an object
display1,2	T	It has the same duration as *obj_dur*, representing the time during which an object is displayed
timer1,2	P	Symbolizes the activation of a timer to check the synchronization between the arrival of two parallel objects
t_out	T	Allows *timer1* and *timer2* to be reset
up_thres1,2	T	Represents the asynchrony tolerated before the application is aborted
ABORT	P	A token in this place means the application was aborted (any subsequent event in the model thus can not be taken into consideration)
E1, E2	P	A token in one place E_i when the other place E_j is empty indicates an asynchrony between the video and the audio streams
low_thres1,2	T	This is the minimum threshold over which we consider a synchronization error exists that requires to perform the SSP
ERROR	P	Counts the number of synchronization errors during a presentation
no_error	T	When the asynchrony is smaller than *low_thres1,2*, we consider that no rescheduling operation is necessary
control	P	This place is used to control the production of token when transitions *display1,2* fire
remove_control	T	When the control has been done on *display1,2* the control can be removed
A1, A2	P	Indicate the control has been done on *display1* (resp. *display2*)

Table 4.2.3 Legend for the model of the SSP: places and transitions

In this model the timed transitions are deterministic, except for transitions *network1-2* which represent the random delays experienced by the media data during the transmission on the network. Transitions *lower_thres1-2* and *up_thres1-2* are deterministic since they represent fixed threshold values, respectively, the minimum asynchrony between the different media streams before any rescheduling operation is performed, and the maximum asynchrony between the media streams that is tolerated before the presentation is aborted. The transition *obj_dur* need not be deterministic by itself (we can easily imagine successive media objects with a different duration), however in this model we need to ensure for each media object that its duration, i.e., the time associated with transition *obj_dur*, and its display, i.e., the time associated with transition *display1* or *display2*, are exactly equal, which can be ensured only when all these transitions have the same deterministic delays.

4.3 Performance Evaluation Tool and Limitations of the Model

Once the model is build, we can use it to determine certain behaviors of the system. Petri Nets allow us to get results from a model both by simulations and analytical resolutions. However, as discussed previously in section 4.1, certain conditions have to be met for a model with deterministic transitions to lead to an analytical resolution. Existing Petri Nets computer tools provide analytical results only when the model does not generate any marking where two deterministic transitions are enabled at a time (a sufficient condition enounced by Marsan in [Mar87]).

In order to get results from the model, we used the UltraSAN tool [Cou91, San93]. UltraSAN is a software package for model-based evaluation of systems represented as stochastic activity networks (SANs) [Mey85], which are a stochastic extension to Petri Nets. UltraSAN offers both analytical and simulation solution modules for transient and steady-state

performance, dependability, and performability measures. However, the use of the analytical solver is submitted to the sufficient condition stated above.

This limitation of the analytical solver can be avoided by modifying some deterministic delays into exponentially distributed delays; the model thus obtained is an approximation of the original one. In our case, to satisfy the sufficient condition we had to assign exponentially distributed delays to transitions *display1-2*, and *obj_dur*. Unfortunately, this modification brought so much change in the behavior of the model that the measures were not relevant anymore. The analytical solvers —as well as the simulation solvers— gave results that were diverging, in that when we increased the buffer capacities (*buf_cap1-2*) of the video and audio CMSCs, the number of media objects in these buffers were increasing, without limit, and synchronization errors were happening for each single media object been displayed. Subsequently, we decided to achieve the performance measures using only the terminating (transient) and steady-state simulation solvers.

4.4 Buffering Delay

When the media data are sent by packets over the network, each packet will experience an end-to-end delay $D_{e\text{-}e}$. On average, however, the delay will have a mean μ and a variance σ^2, parameters which are part of the negotiated QoS and are therefore known by the scheduler. In order to prevent data units from missing their presentation schedule, we must ensure that the delay is such that

$$D_{e\text{-}e} - \mu \leq 0.$$

Since this can not be guaranteed, L. Li introduces the notion of buffering delay in [Lam96.1], which compensates for the predictable network delay variations: each media object must be put on the corresponding data connection $D_{buffering}$ time units in advance such that

$$P[(D_{e\text{-}e} - \mu - D_{buffering}) > 0] < P(\text{Fail}) \qquad \text{(condition 1)},$$

where P(Fail) is the maximum probability of having a late arrival that the application can tolerate (it is an application defined variable that depends on the quality of the presentation that the user of the application wants to get).

When data packets of a media object fail to arrive on time and the delays have exceeded the skew tolerance parameters, the application is aborted only when the skew between related streams is larger than a pre-defined upper threshold value. If this upper threshold has not been reached, but the late arrivals have exceeded the tolerated skew, action may have to be taken in addition to the rescheduling in order to compensate for a —now— visible gap. This is referred to as *restricted blocking* [Ste90.4]: in the example of a video stream, late frames may have to be discarded to recover from the synchronization errors. The difference in the value of the upper threshold and that of the skew tolerance parameter is an indication of the quality required for the presentation: the smaller the difference, the better the quality of the presentation.

However once the quality of the presentation is chosen, the upper threshold value is imposed to the whole system. At this point the probability that the application aborts only depends on the delay and delay variation of the data connections. In order to keep this probability under acceptable values, we need to compensate for the delay variations, and thus extend the conditions imposed on the *buffering delay*. Therefore, we now need to choose D*buffering* such that,

$$P[(D_{e\text{-}e} - \mu - D_{buffering}) > \text{upper\_threshold}] < P(\text{Abort}) \quad \text{(condition 2)}$$

where P(Abort) is the maximum tolerated probability that the application would abort. This last condition on D*buffering* is more stringent than condition 1 since:

$$\text{upper\_threshold} > 0 \text{ and } P(\text{Abort}) < P(\text{Fail}).$$

Simulation results showing the influence of D*buffering* are given in following section 4.5.

4.5 Simulation Results

All the simulations have been done on a RS6000-IBM workstation. Simulation measures have been computed with a 98% minimum confidence level and a confidence interval of 1%.

4.5.1 Qualitative Effect of the SSP

The first set of simulations demonstrates the qualitative effect of the rescheduling operation that is performed within the SSP. Figure 4.5.1.a shows the mean buffer occupancy at steady state, for the audio and the video CMSCs, when no rescheduling is done. In this case, places *end_display1-2*, *A1-2* and *control* and transition *remove_control*, as well as their connections to any transition or place in the rest of the model are removed from the model before any measure is done. By opposition, Figure 4.5.1.b shows the same results when the SSP is performed. In both cases we have set the buffering delay to 0. Various simulation sets have been done each with different objects' duration. The audio and video network connections (end-to-end delay between the media server and the CMSCs) have an associated random delay which has a normal distribution. Both have the same mean delay μ and a different variance σ^2: $\sigma_{audio} = 0.2\mu$ and $\sigma_{video} = 0.5\mu$. The mean varies from 100 ms to 1s, which we assume to be reasonable values for end-to-end delays in a broadband network.

First we can observe that in all cases the mean buffer occupancy, when the steady state is reached, appears as a linear function of the network delay. This result can also be verified with an exponentially distributed random network delay.

In absence of the SSP, the mean buffer occupancy differs for the audio CMSC and the video CMSC (see Figure 4.5.1.a). In this case, each buffer's occupancy only depends on the delay variations introduced by its own network connection: an object will not be displayed before the display of the previous object is completed. Since the connection for the video data has the largest variance, the video CMSC gets the largest buffer occupancy. The effects of the

SSP appear clearly by comparing Figure 4.5.1.a to Figure 4.5.1.b. In Figure 4.5.1.b, the two buffers end up containing the same average number of objects. It only shows two series of plots instead of the four series expected, as on Figure 4.5.1.a. Indeed the difference between the results for the audio buffer and for the video buffer is always smaller than the confidence interval of our simulations; thus, we considered the differences negligible, which leads to a superposition of the plots for the audio and these for the video. This superposition of the two series of plots means that the buffer occupancy at steady state, when the SSP is performed, is independent of the medium, i.e., independent of the variance of the network connection delay. Moreover it appears to equal the mean buffer occupancy of the CMSC whose data connection has the largest variance,

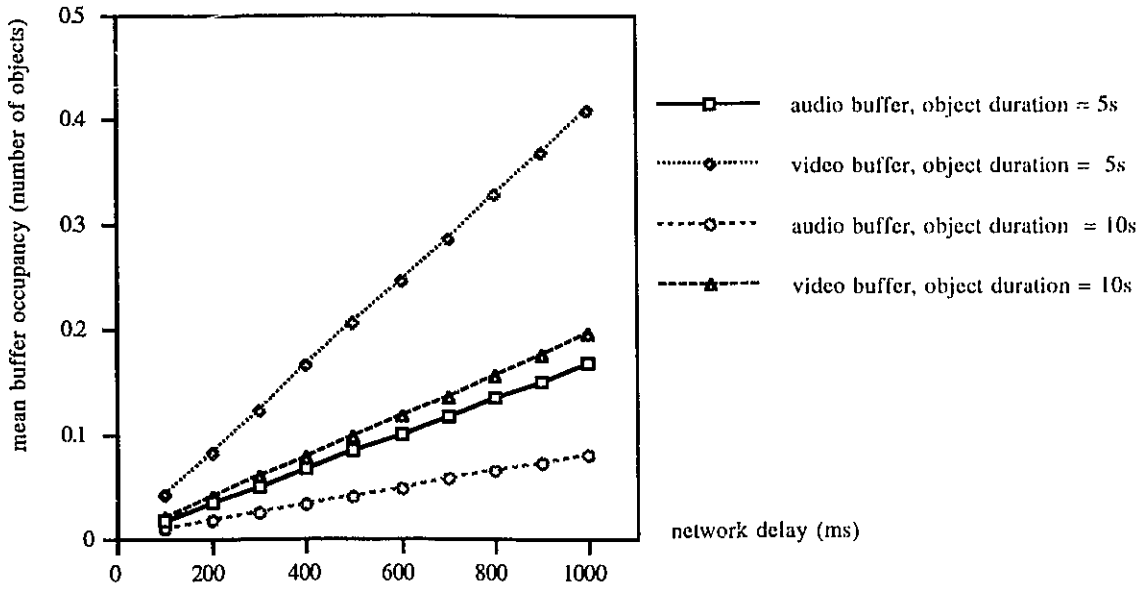

Figure 4.5.1.a Buffer occupancy in absence of rescheduling

when no SSP is performed (i.e., the video CMSC). The mean number of objects in the buffer can be interpreted in terms of the rescheduling delay that a media stream experienced during the presentation. Thus if the buffer occupancy differs for two media, the accumulated delay experienced by one of the stream is different from the one experienced by the second stream. The difference represents the overall asynchrony between the two streams.

Such an asynchrony is observed in Figure 4.5.1.a where no SSP is executed, but it is not observed in Figure 4.5.1.b, when the SSP is executed. We can therefore conclude that the SSP performed at the client site during the presentation prevents the asynchronies from being rampant.

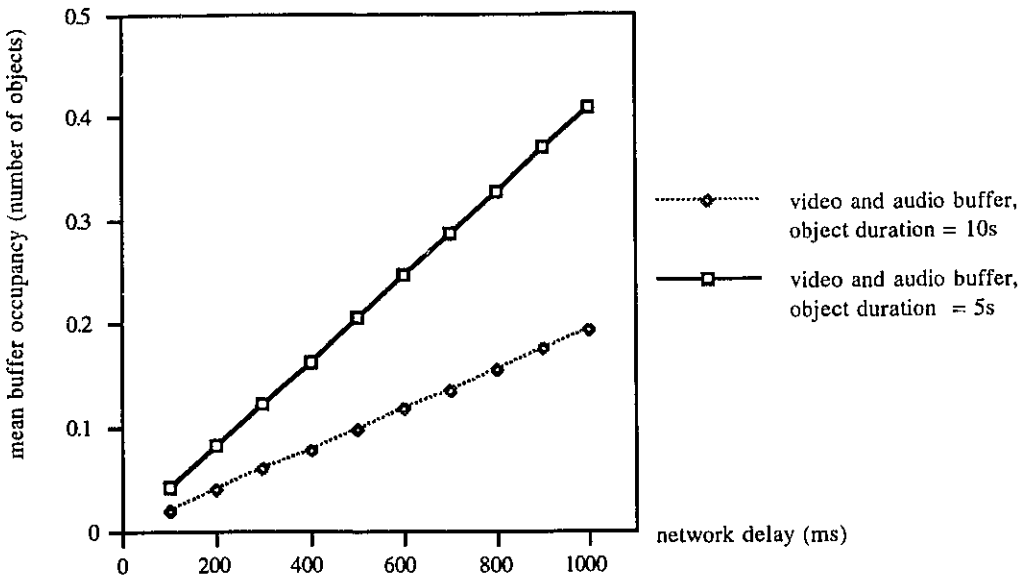

Figure 4.5.1.b Buffer occupancy when the SSP is applied

4.5.2 Influence of the Buffering Delay on the Probability of Aborting the Application

Since we want to guaranty a high quality presentation to the user, we have to ensure that the user will not be bothered by visible asynchronies. If we consider an article involving video with its related audio, we need to synchronize the two streams with "lip-synch" quality. We know from [Ste93] that any asynchrony larger than 80 ms will be detected by the user. In our model, we consider that the application will abort at any moment if the asynchrony reaches the upper threshold of 100 ms. However, because of the random variation of the network delay, in order to be able to run the application without aborting systematically, we introduced a buffering delay at

the very beginning of the presentation to reduce the probability of aborting. Given various values of buffering delay, Figures 4.5.2.a and 4.5.2.b show how the probability of aborting the application depends on the length of the presentation. For all these results, the duration of each object is 10s, b1 and b2, respectively the buffer capacity of the video and audio CMSC's buffer, are equal to 10 objects (so that they do not introduce any side effect). The delay on the data connections is exponentially distributed with a mean equal to 100 ms.

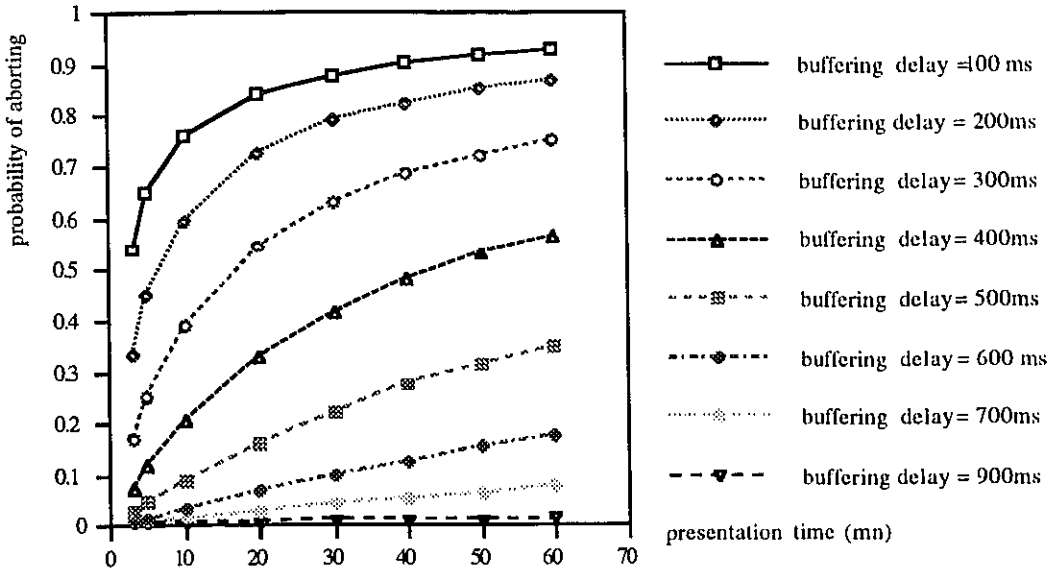

Figure 4.5.2.a Probabilities of aborting the application for various buffering delays

We obtain results that were expected:

- The longer the presentation, the higher the probability that the application will abort.
- When we increase the buffering delay, the probability of aborting the application diminishes to low and acceptable probabilities.

We see clearly on these two figures (and maybe better on Figure 4.5.2.c) that a certain buffering delay can provide a good security (in terms of probability of aborting) for a given presentation but the same delay is not enough if we want to have a longer presentation.

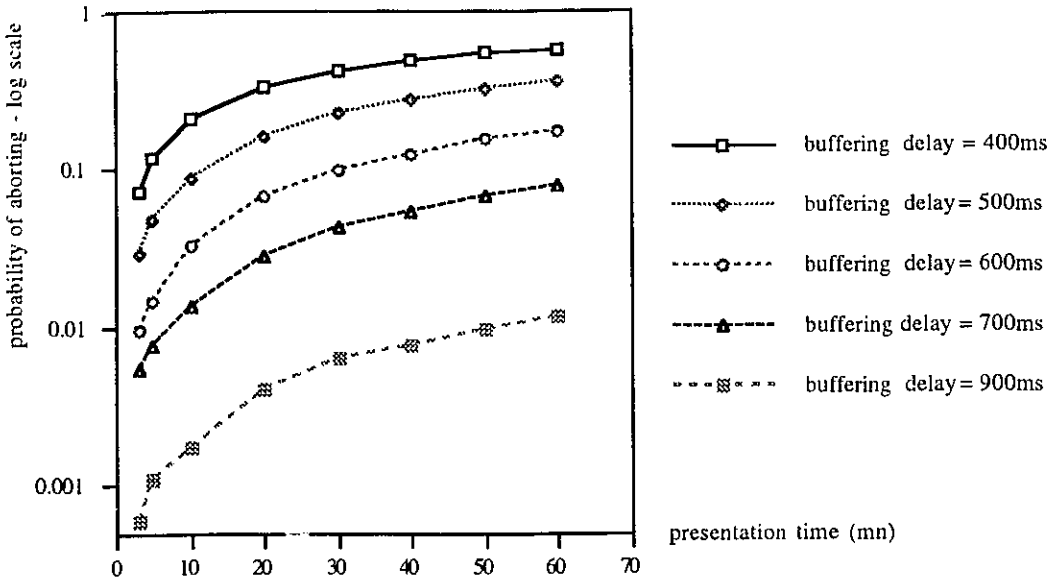

Figure 4.5.2.b Probabilities of aborting the application (with more details on certain results)

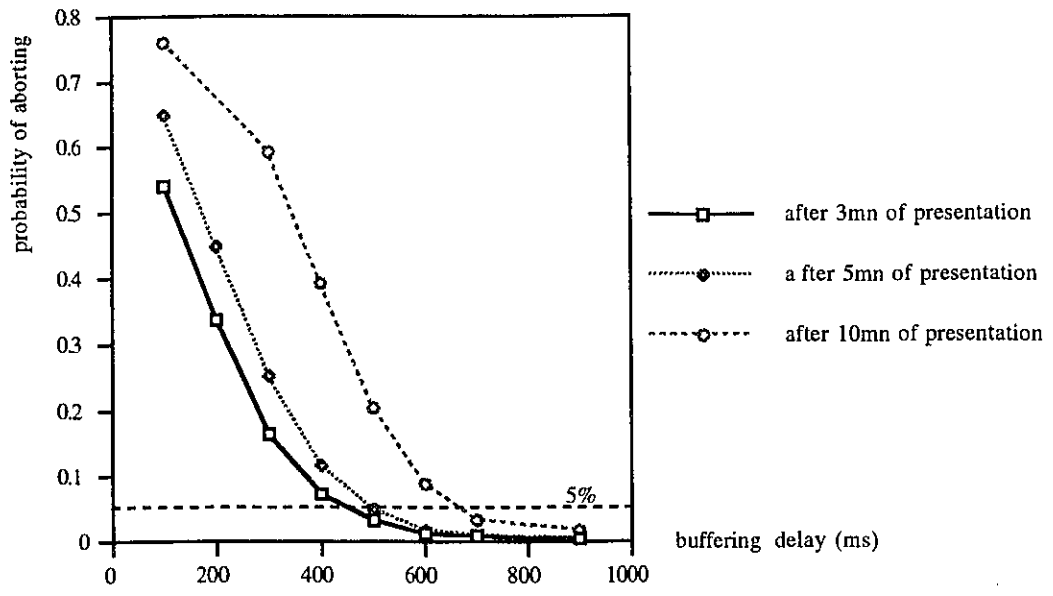

Figure 4.5.2.c The probability of aborting decreases with the buffering delay

If we choose to accept a risk a 5% that our application will be aborted, then, assuming the connection are exponentially distributed with a mean of 100 ms and object of duration 10s, we should choose the buffering delay to be around 450 ms for a 3 mn presentation, 500 ms for a 5 mn presentation, and 670 ms for a 10 mn presentation (see Figures 4.5.2.b and 4.5.2.c). For an hour of presentation, a 900 ms delay lets only 1% of chance that the application aborts.

Of course, if the connection introduces larger delays, the buffering delays should be increased too. However these results show that the required buffering delays are reasonable, considering they are necessary only at the start of a presentation.

4.5.3 Mean Buffer Occupancy

The next results to be presented (see Figure 4.5.3) deal with the mean overall buffer occupancy. In the model we assume that the data of an object arrive all at once (as does a token). Therefore the granularity of the CMSCs' buffer, for our model, is the size of an object. In the model, objects will wait in the buffer of their CMSC until the previous object has finished been displayed. If no random delays were introduced by the data connection, an object would not wait in its CMSC's buffer. However the data of this object would still have to be stored until they are completely displayed, which is represented in the model by the presence of a token in the places *start_display1* or *start_display2*. Hence, the minimum size of the buffer at the CMSC is the size of one object. The buffering of objects that are not yet being displayed has two origins: the rescheduling that takes place when objects are late, and the buffering delay. It corresponds in the model to the presence of tokens in the place *audio_CMSC* or *video_CMSC*. The following parameters have been taken for the simulations used in Figure 4.5.3: the duration of an object is 10 s; the buffer sizes (b1 and b2) are equal to 10 objects; the buffering delay is chosen so that for the given data connection (and thus the given mean network delay, which as an exponential distribution) the probability of aborting is 5% after 10 minutes of presentation.

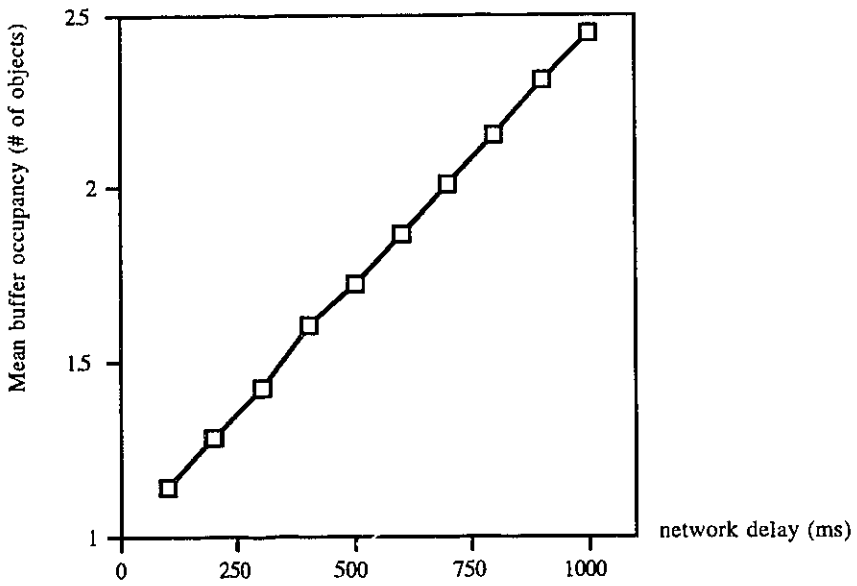

Figure 4.5.3 Mean overall buffer occupancy after 10 mn of presentation

This curve can provide additional information by noting that the object duration is 10s and that 1s of audio data is 22050 bytes, and 1s of MPEG-1 coded video data is roughly 68 kbytes (if we consider the audio and video streams to be as described at the beginning of this section). The values given in Figure 4.5.3 are only indications since they are not peak values.

4.5.4 Synchronization Errors

The last result to be presented (see Figures 4.5.4.a and 4.5.4.b) shows the number of synchronization errors that will occur on average during an hour presentation. It gives us some indications about the rescheduling that will occur at the user's workstation. Figure 4.5.4.a shows the number of synchronization errors varying with the buffering delay. The object duration is 10s, the buffer sizes are still 10 objects and the mean network delay is considered to be 100ms. We observe that the number of errors strongly diminishes when we increase the buffering delay.

This expected result is similar to the decrease of the probability of aborting shown in Figure 4.5.2.c. We observe in all cases that the number of errors is largely acceptable and that the SSP does not introduce any heavy additional load on the client's workstation.

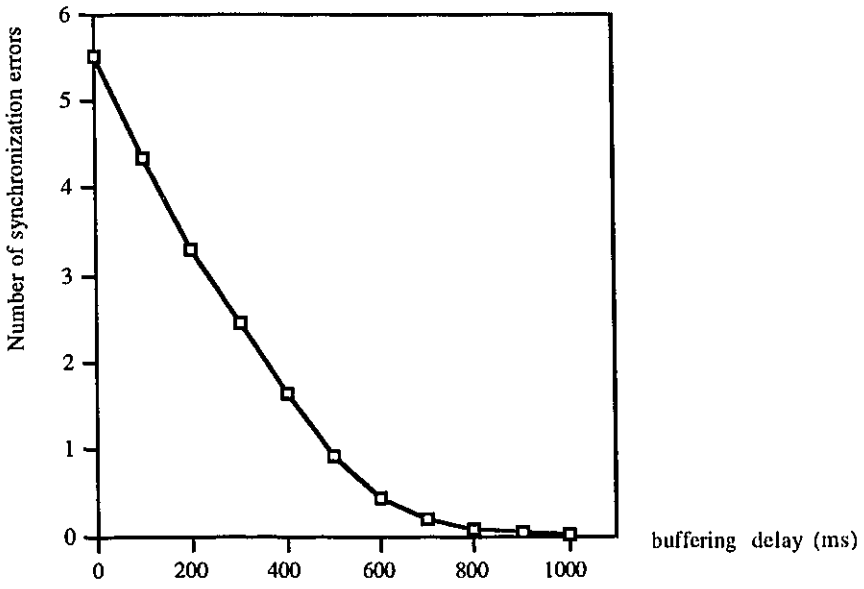

Figure 4.5.4.a Increasing the buffering delay reduces enormously the number of synchronization errors

Figure 15 shows, for some values of the buffering delay, how the number of synchronization errors varies with the duration of the presentation. For these results, the object duration is 10s, the connection has an exponentially distributed network delay with mean 100ms. The number of errors logically increases with the presentation time. We can see that for an buffering delay of 400ms (which is close to the minimum value of the buffering delay that guarantees at most 5% of probability of aborting, see Figure 4.5.2.c) we will have 1 or 2 rescheduling operations to perform during a presentation of up to one hour.

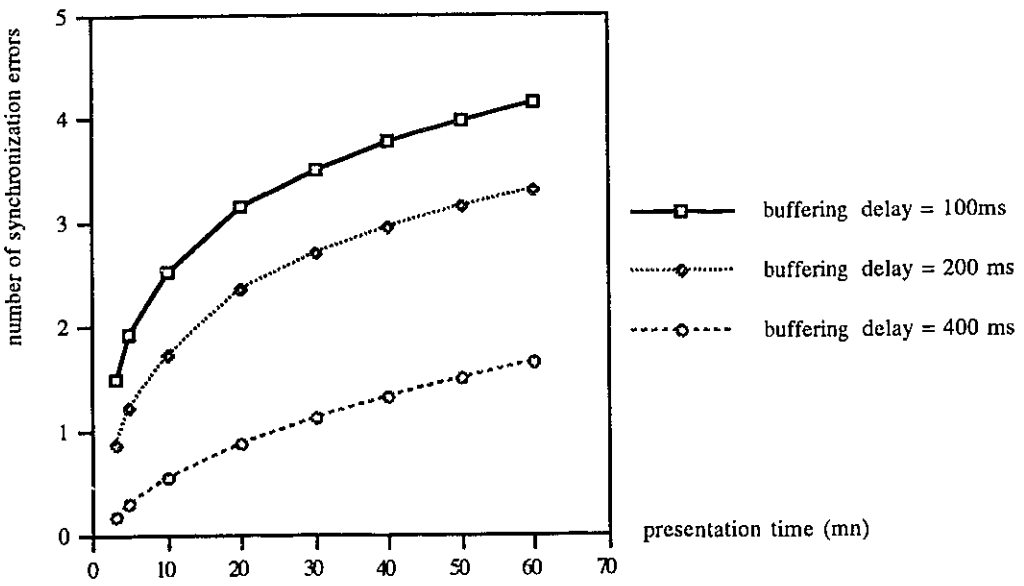

Figure 4.5.4.b Synchronization errors versus the duration of the presentation

4.6 Summary

In this chapter, we presented a DSPN model of the SSP. The model itself does not describe the system in details, in that it considers a media object as the smallest entity manipulated by each components of the system. However it provides us with rough estimates of certain system parameters which comfirms the expected behavior of the system when the SSP is performed.

The simulation results have shown that using the SSP to achieve the synchronization between different media prevents the asynchrony between the different media from being rampant. Moreover the load added to the user's workstation is not significant since the number of re-synchronization operations corresponds to the number of synchronization errors which is kept very small (on average), even for lengthy presentation. The introduction of a 'initial' buffering delay has also shown to be necessary to avoid a flagrant asynchrony, and a subsequent

interruption of the presentation. In addition, the required buffering delay is kept within reasonable values (few hundred milliseconds) and is not considered disturbing for the user since it is introduced only at the very beginning of the presentation. Finally the buffer size required at the user's workstation is kept under reasonable limits.

Chapter 5

Conclusions

5.1 Summary

In this thesis, we first presented the implementation of a prototype for a complete software control system, namely a multimedia synchronizer, that synchronizes multiple data stream generated from the distributed servers of a multimedia database. The multimedia synchronizer is meant to be used within any type of presentational application, where the document to be presented, as well as its temporal structure, are stored in the database. For this prototype, the target application was a Multimedia News On-Demand service.

The implementation was successfully done on IBM-RS6000 workstations connected by a LAN. The types of LAN supported are Ethernet, Token Ring, and a ATM-based LAN. Most of the software components have been designed from scratch and coded in C. They are the Database server, the media servers, the scheduler, the FIFO_MSCs and Socket_MSCs, and the text player. The audio player was already included in the IBM-Ultimedia software library, while the MPEG decoder is a public domain software developed at Berkeley University, CA. Both of

them have been modified to suit the needs of this prototype. The GUI was also designed using an IBM GUI builder, namely the AIXwindows Interface Composer.

The description of the implementation in chapter 3 gives an overview of the functionality of each of the modules previously enumerated. For each module, the important steps of its operation were detailed. In particular, an emphasis was put on the necessary inter-process communications and the means used to achieve them. As a part of the implementation, the GUI and its various windows were presented, and each user interaction and its consequences on the many modules of the prototype were thoroughly explained.

Secondly, we conducted an analysis of the Stream Synchronization Protocol, one of the two synchronization schemes that is used as part of the synchronization control architecture. A high-level DSPN model of the SSP was proposed. A simulation tool was used to get measures from the model, providing estimates on the behavior of parts of the system while the SSP is performed.

The simulation results indicate that the SSP, as expected, is able to prevent the asynchrony between two different media streams from being rampant, i.e., from increasing with the presentation time. The results also demonstrate the need for a buffering delay to be used at the user's site, at the very beginning of the presentation. Indeed, the use of a buffering delay reduces the amplitude of asynchronies thus preventing them to be larger than what the user tolerates. For example, more than 80 ms of asynchrony between a video stream and the related audio stream is visible to the user, thus annoying. Therefore, when the first media objects are received, we purposedly buffer them for a certain duration —the buffering delay— in order to compensate for large delay variation that could appear on the network and avoid later flagrant asynchronies that would lead to the interruption of the presentation. The measures showed that the required buffering delay, in order to have a 5% maximum probability of been larger than the threshold, was of the order of a few 100 ms, and thus acceptable to the user.

The simulation results measures also show that, in the condition of a sufficient buffering

delay, the number of synchronization errors was kept small, and equivalently the number of re-scheduling operations, thus letting the SSP be the fine tuning operation it is supposed to be. Finally, the cummulated intentional delay increasing at each re-scheduling operation does not affect dramatically the buffering need. The buffer space is mainly required to accomodate the high rate at which objects are downloaded and the buffering delay. However, the buffer space required is kept much smaller with the use of the "just-in-time delivery" policy implemented in the Stream Delivery Scheduling, than with a store and forward policy.

5.2 Suggestions for Further Research

As far as the implementation of the synchronization system prototype is concerned, further work should concentrate on two aspects, the communication protocols, and the problem of intra-media synchronization.

- Some intra-media synchronization schemes should be studied and implemented to observe how they complement the SSP.
- Concerning the communication protocols, the research should further investigate a transport protocol that would function on top of an ATM backbone, and that would support QoS negotiation when virtual connections are set up, as well as guarantee the negotiated QoS parameters throughout the use of the connection. Possibly it should allow dynamic QoS re-negotiation to respond to critical situation and avoid applications to abort.

The performance analysis that was done in this thesis is based on the theoretical algorithm and it is also a high-level model. The performance evaluation needs to be pushed much further:

1) A model should be designed that is going into much more details. Transport data units should be considered instead of media objects; the characteristics of the underlying network, of the

servers, of various modules on the client's site should be taken into account; the number of media streams supported should be variable. This model should be close enough to a real-life implementation so that it becomes a useful tool for predicting the behavior of the synchronization control system under the chosen conditions.

2) Performance evaluation should include the qualitative, and possibly quantitative, evaluations of an implementation of the control system that would support the QoS guarantees which are essential for the control system to perform properly. Measures obtained from these observations should then be analyzed and compared to those obtained from the model discussed in 1), to estimate the quality of the model, thus the quality of the predictions that would be derived from it.

Bibliography

[AIC93] IBM Inc., *AIXwindows Interface Composer Developer's Guide.* IBM, 1993.

[And91] D. Anderson and G.Homsy, "A Continuous Media I/O Server and its Synchronization Mechanism," *IEEE Computer*, Oct. 1991.

[Bla91] G. Blair, D. Hutchinson and D. Shepherd, "Tutorial 4: Multi-Media Systems," in *Proceedings of the 3rd IFIP International Conference on High Speed Networking, Berlin, Germany, 18-22 Mars*, 1991.

[Bla96] G. Blakowski, and R. Steinmetz, "A Multimedia Synchronization Survey: Specification, Reference Model and Case Studies," *IEEE JSAC*, Jan. 1996 (to appear).

[Bou92] J. Le Boudec, "The Asynchronous Transfer Mode: a Tutorial," *Computer Networks and ISDN Systems 24*, pp.279-309, 1994.

[Cou91] J. Couvillion *et al*, "Performability Modeling with UltraSAN," *IEEE Software*, Vol. 8, No. 5, pp. 69-80, Sept. 1991.

[CITR94] *Book of Abstracts, CITR Annual Conference '94, Ottawa*, 1994.

[Dra93] Draft Research Program 1994-95, Broadband Services, Feb. 11, 1994.

[Dub94] D. Dubois, N. D. Georganas and E. Horlait, "A QOS Selector for Multimedia Applications on ATM Networks," *Proc. IEEE ICC'94*, New Orleans, May 1994.

[Dug84] J. B. Dugan, *et al.*, "Extended Stochastic Petri Nets: Applications and Analysis," in *Proc. PERFORMANCE 84, Paris, France*, Dec. 1984.

[Dup92] S. Dupuy, W. Tawbi and E. Horlait, "Protocols for High-speed Multimedia Communications Networks," *Computer Communications*, Vol.15, No.6, July/Aug. 1992.

[Fer92] D. Ferrari, A. Gupta, M. Moran and B. Wolfinger, "A Continuous Media Communication Service and its Implementation," *Proc. IEEE GlobeCom '92.*

[Fer92.7] D. Ferrari, "Delay Jitter Control Scheme for Packet-switching Internetworks," *Computer Communications*, Vol. 15, No. 6, pp. 367-373, July/Aug. 1992.

[For93] M. Fortier, "A Store-and-forward Architecture for Video-on-Demand Service," in *Proc. ICCC Multimedia Communications '93 Conf., Banff, Canada*, Apr. 1993.

[Fur94] B. Furht, "Multimedia Systems: An Overview," *IEEE Multimedia Magazine*, Vol. 1, No. 1, 1994.

[Hel92] D. Heller, *Motif Programing Manual (ed. 1992).* O'Reilly & Associates, 1992.

[Her92] R. G. Herrtwich, "The HeiProjects: Support for Distributed Multimedia Applications," *IBM European Networking Center Technical Report No. 43.9206*, 1992.

[Hoe92] P. Hoepner, "Synchronizing the Presentation of Multimedia Objects," *Computer Communications*, Vol. 15, No. 9, Nov. 1992.

[IEEE92] "Multimedia Communications," *Special Issue of the IEEE Comunnications Magazine*, Vol. 30, No. 5, May 1992.

[Jai94] R. Jain, "What Is Multimedia, Anyway?," *IEEE Multimedia*, Vol. 1, No. 3, Fall 94.

[Lam94.5] L. Lamont and N. D. Georganas, "Synchronization Architecture and Protocols for a Multimedia News Service Application," in *Proc. IEEE International Multimedia*

Computing and Systems Conference, Boston, U.S.A., May 1994.

[Lam94.6] L. Lamont, Lian Li, and N. D. Georganas, "Centralized and Distributed Architectures for Multimedia Presentational Applications," *in Proc. of the 3rd International Conf. on Broadband Islands, Hamburg, Germany, 7-9 June*, 1994.

[Lam96.1] L. Lamont, Lian Li, Renaud Brimont, and N. D. Georganas, "Synchronization of Multimedia Data for a Multimedia News on Demand Application," *IEEE JSAC*, Jan 1996 (to appear).

[Leo94] H. Leopold, K. Frimpong-Ansah, and N. Singer, "From Broadband Network Services to a Distributed Multimedia Support-Environment," *Lecture Notes in Computer Science, Multimedia Transport and Teleservice*, Vol. 882, pp. 47-68, Springer Verlag, 1994.

[LiLi92.4] Li Li, A. Karmouch, and N. D. Georganas, "Real-time Synchronization Control in Multimedia Distributed Systems," in *Proc. IEEE Multimedia '92, Monterey, U.S.A.*, April 1992.

[LiLi92.6] Li Li, A.Karmouch and N. D. Georganas, "Synchronization in Real Time Multimedia Data Delivery," *Proc. IEEE ICC '92, Chicago, U.S.A.*, June 1992. pp. 322.1

[LiLi92.7] Li Li, A. Karmouch and N. D. Georganas, "Real-Time Synchronization Control in Multimedia Distributed Systems," *ACM Computer Communication Review*, July 1992.

[LiLi93.4] Li Li, A. Karmouch and N. D. Georganas, "Performance Modelling of Distributed Data Integration in Real-time Multimedia Systems," in *Proc. ICCC Multimedia Communications '93, Banff, Canada*, April 1993.

[LiLi93.6] Li Li, L. Lamont, A. Karmouch and N. D. Georganas, "A Distributed Synchronization Control Scheme in a Group-oriented Multimedia Conferencing System," in *Proceedings of the 2nd International Conference on Broadband*

Islands, Athens, Greece, June 15-16, 1993.

[LiLi94.2-1] Li Li, A. Karmouch, and N. D. Georganas, "Multimedia Teleorchestra with Independent Sources: Part 1 - Temporal Modeling of Collaborative Multimedia Scenarios," *ACM Journal of Multimedia Systems*, Vol. 1, No. 4, Feb. 1994.

[LiLi94.2-2] Li Li, A. Karmouch, and N. D. Georganas, "Multimedia Teleorchestra with Independent Sources: Part 2 - synchronization algorithms," *ACM Journal of Multimedia Systems*, Vol. 1, No. 4, Feb. 1994.

[LiLi94.5] Li Li, A. Karmouch, and N. D. Georganas, "Multimedia Segment Delivery Scheme and its Performance for Real-time Synchronization Control," in *Proc. IEEE ICC '94, New Orleans, U.S.A.*, May 1994.

[Lit90.4] T. D. C. Little and A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, Apr. 1990.

[Lit90.11] T. D. C. Little and A. Ghafoor, "Network Considerations for Distributed Multimedia Object Composition and Communication," *IEEE Network Magazine*, Vol. 4, pp. 32-49, Nov. 1990.

[Lit91.10] T. D. C. Little and A. Ghafoor, "Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks," *IEEE Computer Magazine*, Vol. 24, pp. 42-50 , Oct. 1991.

[Lit91.12] T. D. C. Little and A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services," *IEEE JSAC*, Vol. 9, No. 9, Dec. 1991.

[Loe92] S. Loeb, "Delivering Interactive Multimedia Documents over Networks," *IEEE Communications Magazine*, May 1992.

[Mar84] M. A. Marsan, G. Conte, and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Transaction on Computer Systems,* Vol. 2, No. 2, pp. 93-122, May 1984.

[Mar86] M. A. Marsan, G. Balbo, and G. Conte, *Performance Models of Multiprocessor Systems*
The MIT Press, 1986.

[Mar87] M. A. Marsan and G. Chiola, "On Petri-Nets with Deterministic and Exponentially Distributed Firing Times," *Lecture Notes in Computer Science, Advances in Petri Nets 1987*, Vol. 266, pp. 132-145, Springer Verlag.

[Mey85] J. F. Meyer, A. Movaghar, and W. H. Sanders. "Stochastic Activity Networks: Structure, Behavior, and Application," in *Proceeding of the International Conference on Timed Petri Nets, Torino, Italy*, pp. 106-115, July 1985.

[Mil93] G. Miller, G. Baber and M. Gilliland, "News-on-Demand for Multimedia Networks," in *Proc. ACM Multimedia'93 Conf., San Diego, CA*, Aug. 1993.

[MPEG-1] MPEG, " Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s," *DRAFT International Standard ISO/IEC DIS 11172*, 1992.

[Nic90] C. Nicolaou, "An Architecture for Real-Time Multimedia Communication Systems," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, pp. 391-400, Apr. 1990.

[Ram93] S. Ramanathan and P. V. Ragan, "Adaptive Feedback Techniques for Synchronization Multimedia Retrieval over Integrated Networks," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 2, pp.246-260, April 1993.

[Ran92.4] P. Venkat Rangan, S. Ramanathan, H. M. Vin, and T. Kaeppner. "Media Synchronization in Distributed Multimedia File Systems," in *Proc. Multimedia '92 —4th IEEE ComSoc International Workshop on Multimedia Communications*, Monterey, Calif., pp 315-328, Apr. 1992.

[Ran92.7] P. Venkat Rangan, H. M. Vin, and S. Ramanathan, "Designing a Multimedia On-Demand Service," *IEEE Communications Mag*azine, Vol. 30, No. 7, pp. 56-65, July 1992.

[Rav93] K. Ravindran and V. Bansal, "Delay Compensation Protocols for Synchronization of Multimedia Data Streams," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, No. 4, Aug. 1993.

[Ric90] W. Richard Stevens, *UNIX Network Programming*. Prentice Hall, Englewood Cliffs, N.J., 1990.

[Sad90] H. Sadamoto and S. Matsushita, "Synchronization Control between Text and Voice for Multimedia Message Communication," in *Proc. ICCC 90 International Conf. on Computer Communication*, 1990.

[San93] W. H. Sanders *et al*, "UltraSAN Version 2: Architecture, Features, and Implementation," *Research Report 93S05*, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, Oct. 1993.

[Shep90] D. Shepherd and M. Salmony, "Extending OSI to Support Synchronization Required by Multimedia Applications," *Computer Communicatons*, Vol. 13, No. 7, pp. 399-406, 1990.

[Shep92] D. Shepherd, D. Hutchinson, F. Garcia and G. Coulson, "Protocol Support for Distributed Multimedia Applications," *Computer Communications*, Vol. 15, No. 6, pp. 359-366, July/ Aug. 1992.

[Ste90] R. Steinmetz, "Synchronization Properties In Multimedia Systems," *IEEE Journal on Selected Areas in Communicaitons*, Vol. 8, No. 3, pp. 401-412, Apr. 1990.

[Ste92] R. Steinmetz, "Multimedia Synchronization Techniques: Experiences Based on Different System Structures," in *Proc. IEEE Multimedia'92, Monterey, U.S.A.*, April 1992.

[Ste93] R. Steinmetz and C. Engler, "Human Perception of Media Synchronization," *Technical Report 43.9310*, IBM European Networking Center, Heidelberg, 1993.

[Sut93] S. L. Sutherland and J. Burgin, "B-ISDN Internetworking," *IEEE Communications Magazine*, Aug., 1993.

[Taw93] W. Tawbi, L. Fedaoui and E. Horlait, " Dynamic QoS issues in Distributed Multimedia Systems," in *Proceedings of the 2nd International Conference on Broadband Islands, Athens, Greece, June 15-16*, pp. 69-75, 1993.

[Vin93] H. M. Vin, and P. Venkat Rangan, "Designing a Multiuser HDTV Storage Server," *IEEE Journal on Selected Areas in Comm.*, Vol. 11, No. 1, pp. 153-164, Jan. 1992.

[Vog93] A. Vogel, (Ed.) " Project Overview, QoS Architecture and QoS Negotiation Protocol," Report, Oct. 27, 1993.

[Wit93] V. Witana and A. Seneviratne, "Operating System Support for Multimedia Applications," *ICCC Multimedia Communications'92.*