Design and Implementation of a Time-Based Multimedia Document Architecture

by

James W. Emery

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

M. A. Sc.
in
Electrical Engineering

Ottawa-Carleton Institute of Electrical Engineering

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

October, 1994

ISBN 0-612-00530-5

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

to my family.

Table of Contents

List of Figures

List of Tables

Abstract

Multimedia documents differ significantly from traditional documents that are composed of text and possibly geometric graphics. The introduction of continuous media such as audio, video, and computer generated graphics (e.g., animation) imposes new requirements on document representation. In contrast with static media (e.g., text and graphics), continuous media are presented according to a specific rate and duration. While traditional documents consists of a sequence of pages that can be read, multimedia documents are viewed as a presentation that changes continuously with time. This presentation is referred to as the playback of a document. An *author* creates a multimedia document that will be both seen and heard by a *viewer*. The multimedia document creation process includes the creation of a presentation schedule to specify when the multimedia objects contained in a document will be rendered.

In this thesis, a time-based multimedia document architecture called MEDIADOC is presented. This architecture contains rules and guidelines for the creation and representation of multimedia documents. It is a time-based model since it provides support for the creation of presentation schedules that will drive the playback of multimedia documents. Our model integrates many of the concepts of document representation including logical structures, layout structures, and presentation schedules. Special multimedia authoring tools and system support, which are required to facilitate the potentially complicated nature of multimedia document creation and manipulation, are also discussed. MEDIADOC was implemented as part of a multimedia authoring system. The discussion of the implementation includes a description of the authoring system, the object-oriented methodology that was used for implementing MEDIADOC, and the creation of a sample multimedia document.

Acknowledgments

I would like first to thank my thesis supervisor, Dr. Ahmed Karmouch, for his continuous guidance and patience. Dr. Karmouch provided a very good environment in which to study, and his directions were very helpful to me.

I am grateful to Dr. Balu Madaparthi, Dr. Omar Megzari and James Rody, for their technical help especially during the implementation stage of my work. Their friendship and kindness also helped to make my stay at the University of Ottawa very enjoyable.

The help of the University of Ottawa staff in general, and Ms. Michèle Roy and Ms. Lucette Lepage in particular, is also acknowledged.

My special thanks are due to my parents for their constant support, without which this work would not have been possible.

Acronyms

AVIs	AudioVisual Interactive scriptware
B-ISDN	Broadband Integrated Services Digital Network
BER	Bit Error Rate
CD-ROM	Compact Disk - Read Only Memory
CSCW	Computer Supported Cooperative Work
DQDB	Distributed Queue Dual Bus
DTD	Document Type Definition
DVI	Digital Video Interactive
FDDI	Fibre Distributed Data Interface
GUI	Graphical User Interface
ISO	International Standards Organization
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
MAN	Metropolitan Area Network
MAS	Multimedia Authoring System
MDE	Multimedia Document Editor
MHEG	Multimedia Hypermedia Experts Group
MIS	Multimedia Information System
MPEG	Moving Pictures Experts Group
NMIS	Networked Multimedia Information System
O-O	Object-Oriented
OCPN	Object Composition Petri Net
ODA	Office Document Architecture
SGML	Standard Generalized Markup Language
SORT	Scenario Object Rendering Time
VCR	Video Cassette Recorder
WAN	Wide Area Network

Chapter 1
Introduction

1.1 Motivation

Technological advances and applications in the area of multimedia information systems have been rapidly increasing in recent years [FOX 91]. This major step forward in the field of multimedia was made possible by the tremendous technology transformation in the key domains of information processing and computer communications. The following is a summary of some of the most important technological advances:

1. The emergence of high speed networks with powerful workstations, and new storage technology imposes a new way of processing information and provides

valuable opportunities for the design of new applications that were not possible with existing technologies.

2. Distributed system configurations permit several powerful workstations to share resources located at several sites by communicating through LANs. Other configurations may cover wider areas such as LANs interconnected through MANs (Metropolitan Area Network) and even WANs (Wide Area Network) allowing communications between geographically dispersed users.

3. Direct manipulation of new types of information such as video, voice, and image all integrated in one single entity (multimedia document) is becoming possible. Multimedia documents require faster networks for transmission, high performance processing and powerful storage systems.

4. Optical fibre technology will overcome the limitations of the current networking technology by providing high speed networks permitting the transfer of large amounts of multimedia information.

All of these evolving technologies aim at providing the ability for people to communicate and exchange information in a "natural" way by integrating multimedia information processing and communications. It is likely that multimedia technology will play an important role in the information systems of the future. This technology will support many types of information and will provide new facilities for their acquisition, transfer, manipulation, storage, and retrieval. Multimedia applications will rapidly evolve from stand-alone applications (e.g., workstations with DVI or CD-ROM database, video playback with limited editing capabilities) to fully distributed and complex applications such as remote delivery of entertainment video services to the home (e.g., video on demand), real-estate information systems, groupware, telepresence, audio-visual interactive applications, large general purpose multimedia databases, and multimedia mail.

In our multimedia information research laboratory we are exploring a vision of multimedia research which combines many aspects of telecommunications and information processing. This vision of multimedia information systems has resulted in the creation of the MEDIABASE project. The purpose of MEDIABASE is to develop an advanced high performance multimedia information and communications system with a particular focus on document architectures, database models, high-level communications and synchronization protocols, and real-time physical storage of multimedia data [KAR 93]. Objects manipulated in MEDIABASE are composed of text, graphics, still image, motion video, audio, and voice.

The first step towards the design of such a multimedia information system is to provide an integrated homogeneous way to describe, organize, and structure multimedia information objects, and to represent their temporal relationships in a single entity called multimedia document. Authors can create these documents by using an editor. However, traditional editors have been designed to support only static media such, as text and graphics. Thus, there exists a need to design multimedia document editors that can support the creation and manipulation of multimedia information. One of the applications being developed as part of MEDIABASE is MDE, a Multimedia Document Editor. A multimedia document model is required by MDE so that it can understand how documents containing multimedia information can be created and manipulated. MEDIADOC is the multimedia document model that has been designed for MDE.

1.2 Objectives

The main objective of this research was to design MEDIADOC, a multimedia document architecture. Once completed this architecture would be used as the basis for the development of MDE, the multimedia document editor. Using MDE, authors should be able to create multimedia documents that can be used in a variety of applications. Two

major problems were identified with current multimedia document architectures and authoring systems, namely limited functionality and poor authoring environments. With those two problems in mind, two goals were established for the development of MEDIADOC. The architecture had to be:

1. ***powerful*** so that it adequately describes multimedia documents to the extent that is required by authors. In particular, it should provide a useful set of synchronization specification types for creating the presentation schedule of a multimedia document. The architecture should also support the creation of a wide range of multimedia document types that can be used in many different applications.

2. ***simple*** in the sense that it should not be overly difficult for authors to create multimedia documents, and that their structures should be clear and concise so that they can be easily interpreted, understood, and modified. Furthermore, the task of creating presentation schedules for multimedia objects should not be unnecessarily complicated or cumbersome.

1.3 Thesis Outline

This thesis is organized as follows: Chapter 2 reviews the field of multimedia computing including the various data types and their characteristics, multimedia information systems, and multimedia applications of which three are described in detail. The use and importance of multimedia documents in multimedia applications is also shown. Chapter 3 describes two standards, called Standard Generalized Markup Language (SGML) and Office Document Architecture (ODA), which have been designed to enable the representation and interchange of structured electronic documents. The inability of each standard's document model in supporting multimedia information is also shown.

Chapters 4 and 5 present MEDIADOC. The document creation process, data structures, and rendering synchronization, which together represent the core of the architecture, are described in Chapter 4. Several useful extensions, namely an authoring tool and system support for verifying presentation schedules are detailed in Chapter 5. A comparison of MEDIADOC with related work is also provided at the end of the chapter. Chapter 6 presents the implementation of our multimedia document architecture. Conclusions and suggestions for further research are given in Chapter 7.

1.4 Main Contributions

The main contribution of this research is the design of a multimedia document architecture called MEDIADOC. Our architecture provides rules and guidelines for the creation and representation of multimedia documents with prescheduled presentations. It is different from other architectures in that it provides support not only for the creation of presentation schedules, but also the specification of document structures and layout information. Furthermore, it has been designed with the authoring environment in mind. Authoring tools and support are proposed to simplify the potentially complicated nature of multimedia document authoring.

Another important contribution is the implementation of MEDIADOC. This multimedia document architecture was implemented as part of MDE, our multimedia document editor. The editor allows authors to create multimedia documents and specify temporal information for document playback.

Chapter 2
Multimedia and Applications

2.1 Introduction

"Multimedia refers to the integrated generation, representation, processing, storage, and dissemination of independent machine processable information expressed in multiple time dependent and time independent media such as data, graphics, drawings, voice, audio, and video "[STE 90]. These media, which possess a wide range of characteristics, can be divided into two major classes: *dynamic* and *static* media. While the presentation of dynamic media changes continuously, the display of static media remains constant. These media can be combined in a variety of different multimedia applications.

Multimedia information systems (MIS) are required to support these diverse applications by allowing for the creation, storage, processing, retrieval, and playback of multimedia data. One of the key challenges of a MIS is to manage the heterogeneous mix of data types that multimedia computing offers. These data types have a wide range of characteristics which greatly influence the design of a MIS.

In this chapter, an introduction to the field of multimedia computing and multimedia applications is provided. The following section includes both a discussion of the major multimedia data types and their characteristics, and a description of multimedia information systems. Section 2.3 begins with an overview of multimedia applications followed by a detailed description of three specific examples: *document production*, *advertising kiosks*, and *electronic mail.* Some of the benefits and requirements of these applications are discussed, and the need for multimedia documents and authoring tools is identified.

2.2 Multimedia

2.2.1 Data Types

Clearly the defining characteristic of multimedia is the large variety of data types (Figure 2.1) that are used in multimedia applications. These data types, also referred to as media types, belong to two major classes: *static* media types and *dynamic* media types. *Text, raster graphics* (image), *geometric graphics*, and *data* are examples of static media types. They are classified as static media because their presentation does not change in time; it remains the same when it is displayed and viewed. On the other hand, dynamic media are characterized by a presentation which changes continuously. For this reason, dynamic media are also referred to as continuous media. *Audio*, *video*, *animations*, and *slide shows* are examples of dynamic media.

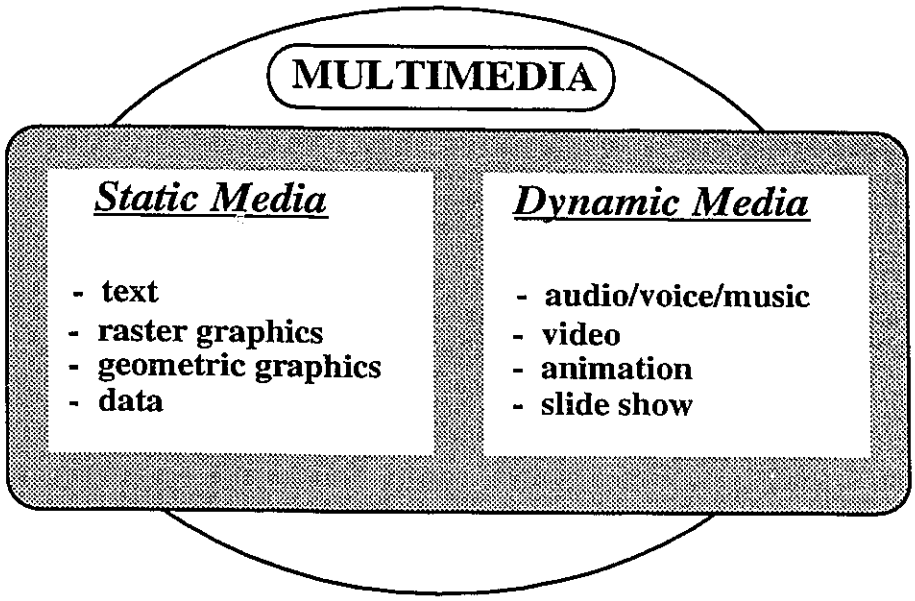


Figure 2.1. *Multimedia data types.*

Text consists of a string of characters, which can be represented in computer systems using a standard coding scheme such as ASCII. Often presentation style parameters are associated with groups of characters to describe how those characters should be displayed. For example, while all the text in a document may require a certain font type (e.g., Courier), titles may be displayed using a larger character size as compared with the characters found in paragraphs. Textual information is also often divided into logical parts such as chapters, sections, and paragraphs.

Raster graphics items (digital images) can be created by scanning pictures on a scanner, and they can subsequently be stored on a digital storage device (e.g., hard drive). A digital image is a two-dimensional array of pixels, where each pixel represents a point having a position and an illumination level. The quality of images can vary greatly depending mainly on two factors: *image resolution* and *colour resolution* [MOO 90]. Image resolution describes the number of pixels in an image's width and height (e.g., 640x480), and colour resolution refers to the number of available colours ranging from

two colours (black and white) to more than 65,000 colours for some high quality images. However, high quality digital images require more storage space than those of a lower quality. Compression techniques, such as those provided by the Joint Photographic Experts Group (JPEG) standard [WAL 91], can be used to reduce those storage requirements.

Geometric graphics items are often referred to simply as drawings. Geometric graphics diagrams are different from images in that they can be created with a graphics drawing package by combining several basic geometric shapes, such as squares and circles.

Data is another type of static media which can be used to create, for example, graphs or pie charts. It consists of a set of integers and/or real numbers and possibly character strings. A typical bar chart could contain information on the rising cost of automobiles over the last ten months. The costs and months could be represented as integers and character strings, respectively. It is interesting to note that this information can be stored in the same format regardless of which type of display (e.g., bar graph, pie chart, etc.) is required by a user for a particular application.

Dynamic media, in contrast with static media, are characterized by a presentation which changes continuously during rendering. For example, a video item is composed of a sequence of frames which are displayed in succession. The display duration of each video frame is constant throughout the presentation. Video, as well as audio, voice, music, animation, and slide shows are examples of dynamic media types.

Audio, voice, and music are the audible media types. Although voice, music, and audio are sometimes treated as independent media types, they can all be considered as audio [DAV 91, MOO 90]. Audio is the media type referring to information which can be heard after being delivered to an audio output device (i.e., speaker). Music can be

represented in two forms: recorded audio or standard music notation (i.e., notes, bars, tempo, etc.). As compared with music, voice is considered a lower quality audio [MOO 90]. Music must be recorded at higher quality levels in order to preserve musical sounds. On the other hand, voice can be recorded at a lower quality since humans can usually understand spoken words even when they are not "crystal clear". Voice is also different from music in that it contains more periods of relatively long silence when a person is not talking.

Video, animation, and slide shows are the visual continuous media types consisting of a sequence of images which are displayed consecutively. The time difference between the display of two consecutive images is normally constant, except in some slide shows where the duration of each slide is equal to the duration of an associated verbal annotation. While this time difference is usually several seconds for slide shows, video and animation sequences require faster display rates. For video, which is composed of images (called frames), a display rate of at least 25-30 frames/second is require in order to maintain smooth motion; video sequences displayed at this rate are referred to as *full-motion video* [DAV 91, MOO 90]. However, the storage requirements for full-motion video are very large (e.g., 30 Mbytes are requires for each second of video [DAV 91]). Compression algorithms, such as the one provided by the Moving Pictures Expert Group (MPEG) standard [LEG 91], can reduce these storage requirements. Compression ratios range from about 30:1 to 200:1 [FUR 94].

2.2.2 Multimedia Information Systems (MIS)

Multimedia computing offers the possibility of developing applications which include several media types. A multimedia information system (MIS) is required to support the requirements of these applications. A very important class of MIS is the

networked multimedia information system (NMIS), which is a MIS that includes a network for the transport of its multimedia information.

The combination of computer communications and multimedia computing allows possibilities for new applications, since data can be distributed across the network residing at possibly multiple locations. For example, data for each media type could be stored on different servers located at several physical locations. In other words, a NMIS could include a set of interconnected media specific servers (i.e., video server, audio server, data server, etc.) which can offer data management specialized to suit the particular characteristics of each media type. Other distributed applications such as conferencing and collaborative document editing are also possible when networking forms a part of the MIS.

Figure 2.2 shows a NMIS including its major components. Multimedia data can originate from a wide variety of sources including video cameras, microphones, file systems, and databases. Video cameras and microphones are refereed to as live sources. In applications such as video conferencing, live voice and video are transported directly from the sources to their destinations. Multicasting is required if more than two parties are participating in the conference. File systems and databases are referred to as data repositories. Persistent multimedia objects can be stored in these data repositories and later retrieved in a manner dictated by the user and the application.

Media compression and large storage devices are required for storing multimedia objects. Video is the most notable example; ten minutes of compressed (30:1) digitized video requires about 550 Mbytes of storage. Compression also provides another benefit, namely improved network utilization, since transporting compressed data over the network requires less bandwidth than uncompressed data.

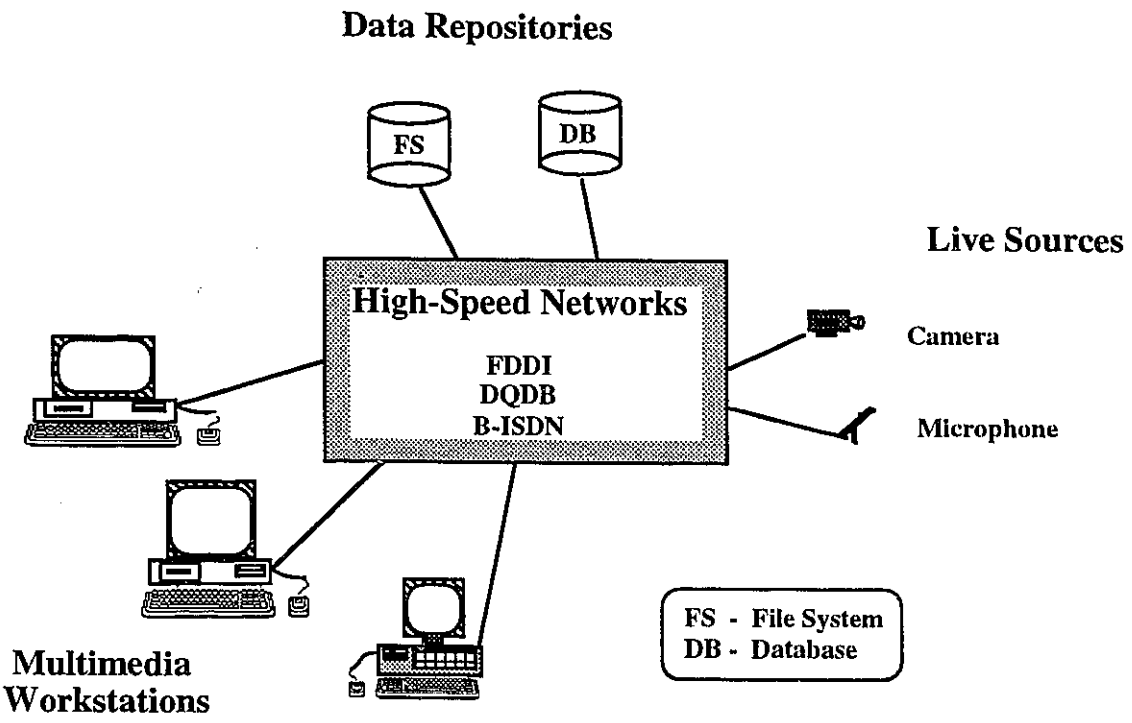


Figure 2.2. *Networked multimedia information system (NMIS).*

Another major component of a NMIS is a high-speed network such as Fibre Distributed Data Interface (FDDI), Distributed Queue Dual Bus (DQDB), and Broadband Integrated Services Digital Network (B-ISDN). These networks can not only provide large bandwidth, but also low delay and low bit error rates (BER), which are required by multimedia applications. High bandwidth is required by multimedia applications, since they contain large pieces of information (i.e., media objects) such as video and audio sequences, animation, and images. Low delay is needed by interactive and real-time applications such as video conferencing. Large delays in video conferencing reduce the quality of communications. Lastly, low bit error rates are important since they enable the transport of information without significant packet loss or corruption and without the need for retransmission [HEH 90].

After being transported across the network, these media objects can then be presented at a local workstation. This arriving data may be buffered and it may require decompression before being displayed. Alternatively, the multimedia information could be stored and edited locally with an appropriate media editor. The type of presentation and/or local manipulation of the media objects depends on the particular application. Some multimedia applications will now be investigated.

2.3 Multimedia Applications

2.3.1 Overview

Existing applications, such as document authoring, can be greatly enhanced with the use of multimedia information. When media such as audio and video can be included in a document, information can be communicated in a more natural and effective manner. Multimedia computing is not only beneficial for existing applications, but has also led to the development of many new multimedia applications that take advantage of the recent advances in information processing and communications technology. For example, video conferencing is now possible due both to the emergence of multimedia in the computing environment and to new high-speed networks.

Currently many potential application areas for multimedia exist. These general areas include office automation, service industry applications, retail applications, computer supported cooperative work (CSCW), domestic applications, science and engineering applications, and cultural activities [WIL 94]. Some specific multimedia applications include joint document editing, kiosks for advertising and banking, and medical information system applications. Table 2.1 shows how these and other specific applications relate to the general application areas. The remainder of this section includes

a detailed discussion of three multimedia applications, namely document production, advertising kiosks, and electronic mail.

Table 2.1. *Multimedia applications.*

General Area	Specific Area	Applications
Office Automation	Document production	multimedia document editor hypermedia browser
Service Industry	Educational	distance learning multimedia encyclopedia multimedia manuals
	Financial	multimedia bank kiosk stock market reports
	Health	medical information systems
Retail	Advertising	advertising kiosks multimedia buy/sell
	Publishing	multimedia news
	Travel and property	vacation package brochures real estate systems
Computer Supported Cooperative Work	Message systems	multimedia electronic mail
	Computer conferencing	message conference
	Meeting rooms	meeting room conference
	Collaborative work	joint document editing
Domestic		home shopping cooking lessons video on demand
Science and Engineering		distributed manufacturing and design geographic information systems
Cultural Activities		virtual museums and art galleries

2.3.2 Multimedia Document Production

Electronic document production refers to the creation of documents on computers. Two areas of document production applications exist, namely the *publishing* environment and the *office* environment. Documents, such as books, journals, and encyclopedias are produced by the publishing environment. On the other hand, the office environment includes business documents such as forms, letters, and reports. Standard Generalized Markup Language (SGML) [ISO 86] and Office Document Architecture (ODA) [ISO 88] are two standards designed to enable *open* (i.e., vendor independent) document processing. They are discussed in detail in the next chapter.

The first electronic documents contained only text. They were created by using a word processor or editor, which would assist the author in creating documents by handling activities such as text entry, word-wrap, spell checking, word counting, and text style changes. Later, the capabilities of these editors were extended so that they could support the inclusion of geometric graphics in documents. These documents, which contain text and possibly geometric graphics, are referred to as traditional documents. They are either displayed on monitors or printed on paper for human perception.

The emergence of multimedia has led to the development of multimedia document editors. These authoring systems are capable of supporting a wide range of media types for inclusion in documents. Several new problems and requirements have developed due to the incorporation of dynamic media such as audio and video. For example, dynamic media cannot be printed on paper. Each media item in a document is referred to as a multimedia object. Multimedia editors must be able to support the creation, management, storage, manipulation, retrieval, and display of these multimedia objects.

Storage and retrieval services can sometimes be provided by a database which has been designed to support certain queries on the information that it contains. Authors can manipulate multimedia objects by using a media editor. For example, a video editor must provide support for "cutting and pasting" a frame or sequence of frames. It should also provide VCR features such as *play*, *stop*, *rewind*, *fast-forward*, and *pause* so that the author can move through the video sequence in a variety of manners. The heterogeneous mix of dynamic and static media also causes problems for display since the rendering of some media change continuously while others remain constant. New support for the display of multimedia objects must be provided so that the viewer, who is the person "reading" the document, can extract information from the document effectively.

Several multimedia document editors have been introduced to the market, and many research prototypes have been developed [CHR 86, NAF 90, GAI 93]. Since many of these products and prototypes are limited in terms of their capabilities and the range of media that they can support, research and development in industry and at educational institutions is very active.

2.3.3 Multimedia Advertising Kiosks

Another important multimedia application is advertising kiosks that can be used to help the process of buying and selling. These kiosks can be stationed in a wide variety of locations such as shopping malls, office buildings, and restaurants. Multimedia adds a new dimension to advertising that is not present in magazine, billboard, radio, and television advertising.

A multimedia advertising kiosk can provide users with many features that are not available with other advertising methods. Firstly, since these kiosk advertisements are stored, they can be replayed if a user wants to view the information again. However, this

capability does not exist with radio and television advertising which cannot be instantly replayed, paused, or rewinded, etc. These advertising media are limited to providing only a limited amount of information that must be quickly memorized by the user.

Another advantage to kiosk advertising exists when the multimedia objects contained in the advertising are stored in a database. For example, at a buy&sell kiosk, users can perform queries to select only the items that they want. A buyer may choose to view all the VCRs from a particular manufacturing company for sale that have been listed in the last two weeks. This query capability can greatly reduce the time required to search for the types of items that a buyer wants to view.

The use of dynamic media in kiosks can greatly enhance advertisements by conveying information in a more exciting and effective way than by using magazines, etc. Buy&sell magazines can offer only a limited amount of information due both to the small layout spaces and the limitations of static media. Multimedia buy&sell kiosks provide an interesting alternative. For example, an advertisement to sell a car could include a video of the owner starting and driving the car. This video could be accompanied by exciting music and a voice recording of the owner describing the most interesting features of his/her car.

Although exciting possibilities exist for multimedia advertising in which several different media can be displayed simultaneously, the creation of these advertisements is not trivial. Firstly, the media items must be created and grouped together. The media items can be created by video cameras, microphones, text editors, and graphics packages. Video and audio information is digitized and stored on a disk along with text and graphics. These multimedia objects are assembled into a document and organized by the author of the document. A presentation schedule must also be created so that the multimedia objects are displayed at specific times. The result is a dynamic presentation in which various media items are appearing and disappearing on the monitor and speaker.

2.3.4 Multimedia Electronic Mail

Electronic mail is a very useful means for computer users to communicate information quickly. Traditionally text was the sole media type found in electronic mail, but current research and standardization efforts are attempting to provide support for other media types such as audio and video. However, the inclusion of multimedia information causes several problems that do not exist with traditional text-based electronic mail.

The size of the message is one of these problems, since the storage requirements could vary from very small (e.g., a few lines of text) to very large (e.g., one minute video) messages [WIL 94]. It is quite possible for a few large messages to exceed the available storage capacity for electronic mail at a receiving site. One solution is to leave the larger media items at the creator's location and to simply reference them at the receiving location. Viewing the message would then require real-time transmission of these multimedia objects from source to destination.

Since multimedia electronic messages may contain dynamic media, presentation schedules must also be created by the author of the message. These schedules allow messages to be played in a way that is useful for the recipient by both defining the order of multimedia objects to be displayed sequentially and specifying those multimedia objects that should be presented concurrently. Thus, a multimedia message can be viewed as a multimedia document, since it may contain a mixture of dynamic and static media that have presentation specification requirements.

2.4 Conclusion

The common feature of all multimedia applications is that they may contain several different media types including audio, video, text, and graphics. Three multimedia

applications were discussed in this chapter, namely *document production*, *advertising kiosks*, and *electronic mail*. In each of these multimedia applications, a set of multimedia objects has to be created, assembled, and organized. This grouping of information is a multimedia document whose media items must be pre-scheduled for presentation to a viewer.

Although all multimedia applications do not need their multimedia information to be packaged in a document with presentation schedules, many applications including those discussed in this chapter have such a requirement. A document model, which can represent, structure, and organize these multimedia objects, must therefore be developed in order to support many multimedia applications. In the next chapter, two standards (ODA and SGML) for structuring documents are discussed, and their ability to support the requirements of multimedia documents are analyzed.

Chapter 3
Standards for Structured Documents

3.1 Introduction

A document is some group of information that is intended for human perception. Often this information can be divided into logical parts, such as titles, appendices, chapters, and sections; the aggregate of these parts is referred to as a document's *structure*. However, most word processors are not aware of the structure of a document, and they view a document simply as a long string of characters within which presentation style specifications, such as font and character size, can be embedded. This type of structuring is *implicit*, since it is managed only by the author. On the other hand, some editors maintain an *explicit* document structure.

Two major benefits are derived when editors maintain explicit document structures. Firstly, an editor that understands the logical structure of a document can help the author by providing certain services such as chapter/section numbering, table of contents generation, and consistent layout specification (i.e., all chapter titles should have a particular character size and font) [BRO 89]. The second benefit is open document processing, which allows the transfer of documents between dissimilar document processing systems. However, a common document model, which must be understood by each system participating in the document transfer, is required.

Document models are provided in the Office Document Architecture (ODA) [ISO 89] and Standard Generalized Markup Language (SGML) [ISO 86] standards. These two international standards are designed to enable the representation and interchange of structured electronic documents.

Electronic documents are used to communicate information. The information contained in a document is conveyed after being *laid out* on a plane surface (i.e., a terminal display screen or paper). Documents are laid out according to rules varying with culture and language. For instance, Western languages expect pages to be laid out from left to right and top to bottom. This process is termed layout control; text is placed in lines of a particular length, lines are assembled into paragraphs, paragraphs are placed on pages without isolating the first line of a paragraph at the bottom of a page, etc. The layout process precisely defines where information is to appear on a surface when it is to be displayed or printed.

ODA has been created specifically for the office environment, and SGML was defined for the publishing environment. Since these two standards have been designed to meet the needs of their respective intended environment, they contain document models, which though similar are not identical. The purpose of document models is to provide a

method for creating the structured description of the electronic representation of these documents.

3.2 Standard Generalized Markup Language (SGML)

The Standard Generalized Markup Language (SGML) standard was designed to enable the representation of documents, such as books, journals, and encyclopedias, that are produced in the publishing environment. These documents are created by either one or more authors and submitted to a publishing group, which produces the final form of the documents so that they can be distributed to customers [BOR 91]. The authors do not take care of document layout, and their main task is to create the content of the document that has been organized into chapters, section, paragraphs, etc. Layout controls are determined and added by the publishing house. Often several layout forms are produced for different market segments.

3.2.1 SGML Markup

Since SGML is a generic markup language, SGML documents contain not only the text of the document, but also markup commands identifying the start and end of each logical part. These logical items are referred to as *elements*. Each element has a *start-tag* and an *end-tag*, and its contents is found between these two tags. Hierarchical structures are formed by including the tags of other elements ("children") in the content of a "parent" element. Thus, a tree-like structure is used to represent the logical composition of each SGML document. Figure 3.1 shows an example of SGML markup.

```
<Chapter>
<Title>The Rules of Squash</Title>
<Par>Squash is played with either two or four players. .....
.....
<Figure file = "squash_rule1">
.....</Par>
<Par>.....</Par>
<Par>.....</Par>
</Chapter>
```

Figure 3.1. *Example of SGML markup.*

The markup tag for each element begins and ends respectively with '<' and '>' symbols. The end-tag also includes a '/' symbol to differentiate it from the start-tag. Each markup tag has a *generic identifier*, which is the name of the element type (e.g., Chapter, Title, etc.) to which the element belongs. Optional *attributes* describing an element are placed after the generic identifier in the start tag. In the example, the chapter is composed of a title and three paragraphs (Par). One of the paragraphs contains a figure with a filename attribute to identify the file (on the local disk) that contains the figure.

3.2.2 Document Type Definition (DTD)

In SGML, each document refers to a Document Type Definition (DTD), which contains the rules for creating documents of that document type. In this way, documents are created from document classes. Thus SGML provides the object-oriented capability of creating objects from classes.

The following is a brief review of this concept. In object-oriented terminology, a *class* defines the properties of all the objects that belong to that class. Specific objects,

called *instances*, are created from a class definition. For example, a class called "Human Being" could be defined with attributes such as age, weight, and height, to represent people. Thus all human beings should have an age, weight, and height. However, since every person does not have the same values for these attributes, values are assigned to the instances of the class, such as Jane Smith, Felix Potvin, Peter Gabriel, and all other persons. The process of creating instances from a class is called *class instantiation*.

Each SGML document is created from a document class whose definition is found in a Document Type Definition (DTD). A DTD is a list of element types (ELEMENT) and attribute list (ATTLIST) definitions. Figure 3.2 provides the DTD for the previous SGML document example. Each ELEMENT definition is composed of a generic element identifier as well as the allowable content type (shown in brackets in the figure) for that particular element type. Thus several elements of a particular type can be created from the ELEMENT definition.

```
<!ELEMENT  Chapter     (Title, Par +)>
<!ELEMENT  Title       (#PCDATA)>
<!ELEMENT  Par         (#PCDATA) +(Figure)>
<!ELEMENT  Figure      EMPTY>
<!ATTLIST  Figure      file      ENTITY      #REQUIRED>
```

Figure 3.2. *SGML document type definition (DTD) example.*

In our SGML document example, the document contains three elements of type "Par" (paragraph). An ELEMENT definition in the DTD (Figure 3.2) shows that elements of type "Par" contain only characters (#PCDATA is SGML notation for character content) and figures. Furthermore, each chapter consists of a title and one or more paragraphs (Par).

"Figures" contain neither character nor SGML element types, and thus their content type is defined as EMPTY. However, each figure element has a "file" attribute so that its content can be found.

Thus, an SGML document is created by following the definitions contained in its Document Type Definition (DTD). In other words, a DTD acts as a guide for driving the creation of documents. A completed SGML document is compiled by an SGML parser in order to verify the validity of the document structure and the markup. Parsing also enables the replacement of markup with processing instructions that will enable a formatter to lay out the document. However, SGML does not include any semantics for representing the physical view of a document (i.e., where the content is to be placed on each page). On the other hand, ODA includes standardized directives required in the formatting of a document. ODA is discussed in the following section.

3.3 Office Document Architecture (ODA)

The Office Document Architecture (ODA) standard was created to enable the representation of documents, such as reports, letters, forms, invoices, and memoranda, that are frequently used in an office environment. It is called an architecture because it provides a comprehensive set of rules for office document representation. These rules can be used by an editor for the purpose of structured document creation.

Within an office environment, documents are interchanged *blindly* [BOR 91]. In other words, documents must be transferred between arbitrary originators and recipients. These documents must often be in a processable form so that they can be edited by the recipient. Furthermore, document layout is often controlled by the originator who may, for example, want to maintain a company style. Thus, layout information associated with these documents must also accompany the interchanged document.

3.3.1 ODA Document Model

The ODA standard provides a document model that is the basis for the representation of electronic documents. Figure 3.3 shows the main components of this model. Each document is composed of a document *profile* and its *content*. The document profile describes the document as a whole by means of a list of attributes such as title, version number, name of the author, and date of creation. The document's content is the set of all the pieces of information, such as text and figures, that we normally associate with documents. ODA supports three content types: text, raster graphics (images), and geometric graphics.

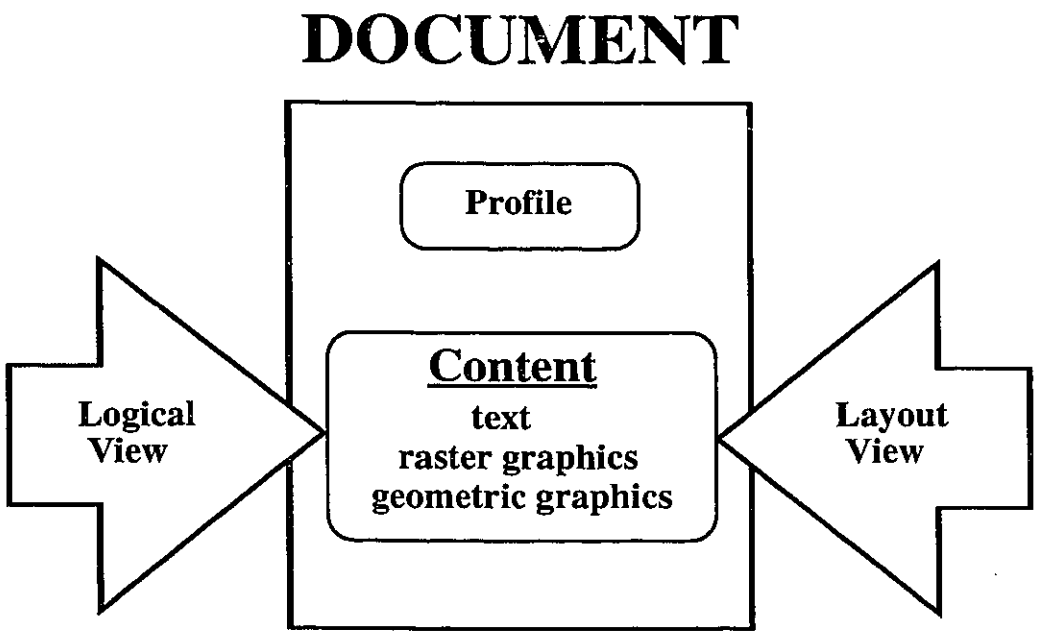


Figure 3.3. *Main components of the ODA document model.*

A key concept in the ODA document model is that two views are required to represent the content of a document [SCH 88]. Figure 3.3 shows that these two views are

the logical view and the layout view. The logical view describes the content of a document in terms of components that are useful for organizing the document. The names of typical logical components are title, paragraph, figure, caption, etc. These logical components are analogous to SGML's *elements*. Furthermore, ODA also represents the logical organization of a document by means of a hierarchical structure (called the *logical structure* in ODA).

However, SGML does not have an equivalent to ODA's layout view, which describes the presentation properties of documents. Layout information is required for describing how the document content will be physically laid out on paper or the display of a monitor. The *layout structure* contains this presentation information. The separation of logical and layout information is an important concept, since it permits the creation of multiple layout structures for a document without duplication of the logical structure and content [MOU 91]. These two types of structures will now be discussed.

3.3.2 Logical and Layout Structures

The principal concepts used to express the structures of documents are the notions of abstract objects, attributes for describing the objects, hierarchical links between these objects, and object ordering. All of these concepts are represented in a tree structure allowing a document to be fully described in a graphical form. The logical and layout structures are used to describe the content of a document.

The hierarchical logical structure for ODA documents is shown in Figure 3.4. The top level in this structure is the *document logical root*. This root object can be divided into a number of *composite logical objects*, which can themselves be divided into composite logical objects. Thus several layers of composite objects can be created. The nodes at the lowest level are referred to as *basic logical objects*, and they act as containers for the content portions of a document. These containers can hold only one type of content.

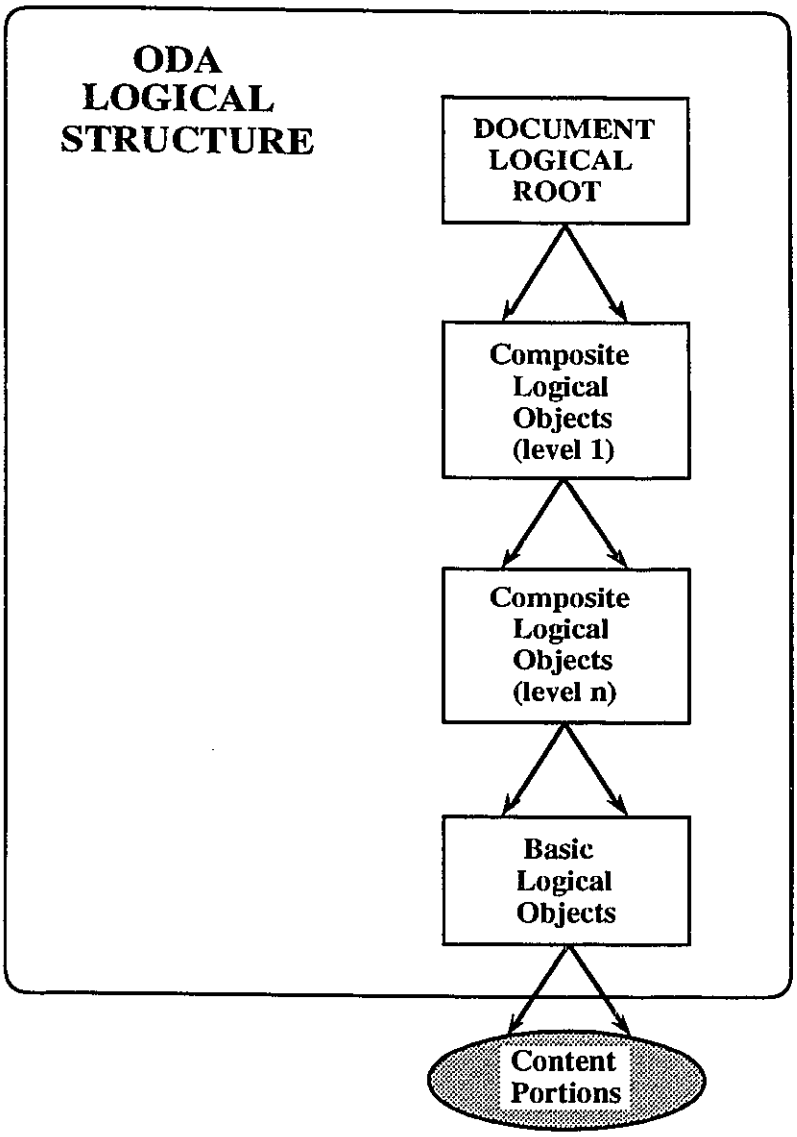
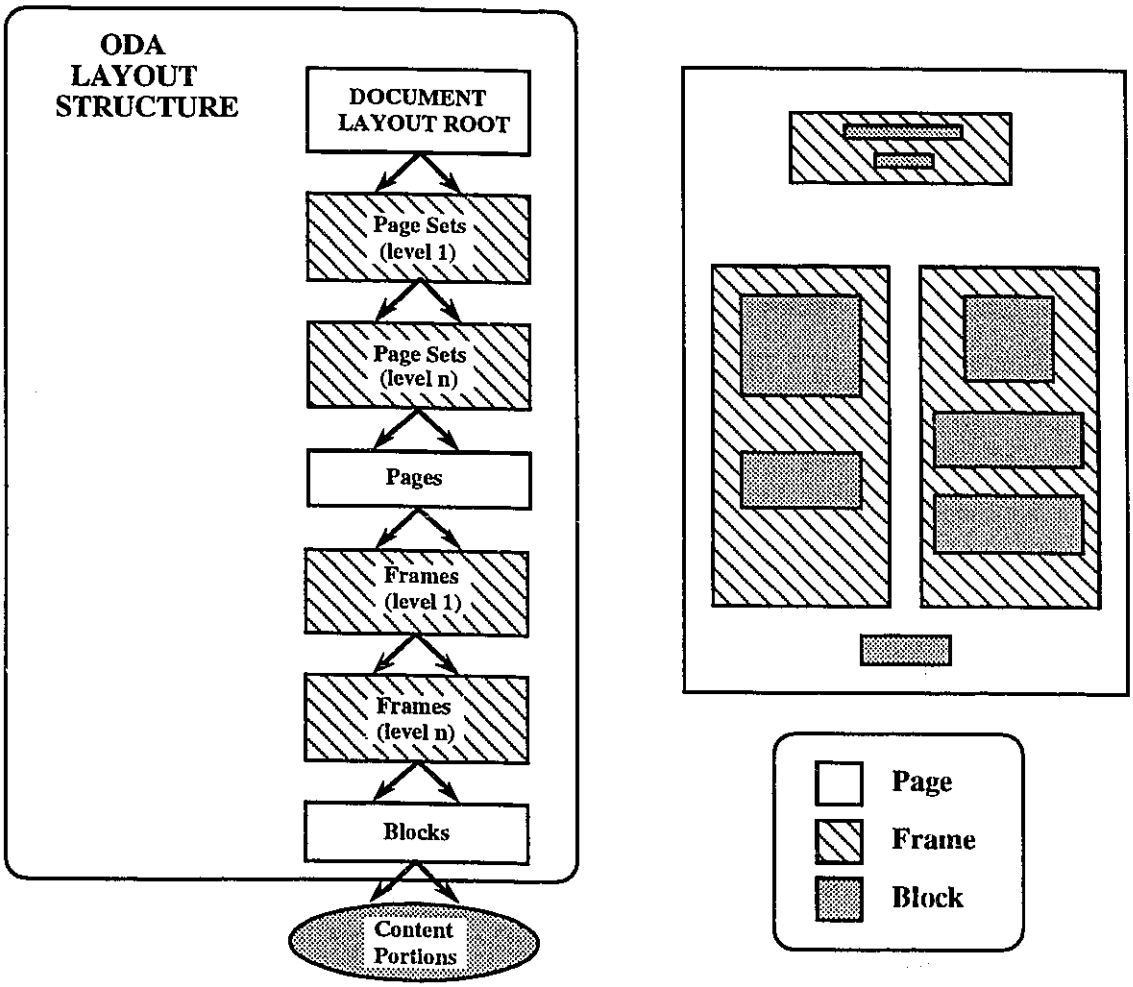


Figure 3.4. *ODA hierarchical logical structure.*

The hierarchical layout structure for ODA documents is shown in Figure 3.5(a). Temporarily ignoring the optional object types (shown with a hatched background), a layout structure is composed of a *document layout root*, which is divided into *pages*, which are further subdivided into *blocks*. *Page sets* and *frames*, the optional object types, serve for grouping pages and blocks, respectively. Blocks are the basic layout objects. These objects describe the size and location of *content portions*, which are individual pieces of information from the document contents. Thus, by laying out blocks on a page, we are effectively specifying the format of the document so that it can be either printed on paper or viewed on a monitor.

An example of page layout is provided in Figure 3.5(b). Each page layout object represents one page of paper. However, it is the application that decides how to use the page layout objects to suit the needs of the user. For example, several pages can be reduced in size and displayed simultaneously on a monitor. This resizing facility can be used by an author when he/she wants to view several pages simultaneously on an editor that supports a "print preview" function. In editors that support this feature, an author edits a document without seeing the effects of presentation style attributes, such as font and character size. By executing the print preview function, the author can view one or more pages of the formatted document before it is printed on paper.

As can be seen from the figure, blocks and frames are rectangular regions on a page. Since a block may either be part of a frame or an independent region of a page, its purpose is to position document content within frames and pages. Blocks, which for example represent paragraphs or pictures, can be assembled into frames. These frames can be used to divide a page into one or more columns of content. The position and extent of these frames can be specified as fixed or variable. While a frame with a fixed layout specification is assigned a specific region on the page object, a variable specification allows the position and dimensions of the frame to be defined relative to other frames.



(a) Hierarchical layout structure.

(b) A page layout example.

Figure 3.5. *ODA layout model.*

3.3.3 Generic and Specific Structures

As in SGML, ODA also provides the object-oriented capability of creating documents from document classes. For this purpose, ODA permits the definition of document classes and document instances for both its logical and layout structures.

In ODA, logical and layout structure classes are referred to respectively as *generic logical structures* and *generic layout structures*. The individual instances of these generic structures are termed *specific logical structure* and *specific layout structure*, respectively. Thus, using the object-oriented paradigm, several specific logical structures (i.e., documents) can be created from a single generic logical structure (i.e., document class). In other words, a generic structure acts as a template for driving the creation of specific structures. Examples of ODA document classes are "Monthly Financial Report" and "Company Newsletter".

A generic logical structure provides definitions for *logical object classes* to describe the characteristics of the objects in the associated specific logical structures. These logical object class definitions include a list of attribute names and their associated values types. An example of an attribute name is *header* and its value type is "character string". Another important attribute *constructor for subordinates* is used to define allowable subordinate (child) object classes and their modifiers. For example, the definition of a "section" may specify that it must be composed of a sequence of "paragraphs" and "figures", which are considered the subordinates (or children) of "section". In this way, the generic logical structure defines how specific logical structures (i.e., hierarchies) can be created for a particular class of documents.

Three modifiers, namely sequence (SEQ), aggregate (AGG), and choice (CHO), are available for specifying the relationship among subordinates of a logical object class. While *sequence* implies an ordering of objects, *aggregate* subordinates have no ordering. The *choice* modifier is used when one class from a group of logical object classes is to be selected as the subordinate type for a logical object in the specific logical structure.

In generic logical structures, allowable subordinates for each object class are marked as either optional (OPT), required (REQ), repetitive (REP), or optional and repetitive (OPT REP). This identifies the allowable number of subordinate objects for each

object that is created in a document's specific logical structure. *Optional* implies either one or no occurrences, *required* implies exactly one occurrence, *repetitive* implies more than one occurrence, and *optional and repetitive* implies any number (including zero) of occurrences of a subordinate object are possible.

An example of a generic logical structure is shown in Figure 3.6. Documents created from this document class are composed of a sequence of "Parts" each of which is a sequence of "Sections". Sections have two possible formats. The first format consists of a raster graphics, followed by an author and a title. The other format includes a paragraph, a raster graphics item, and an optional geometric graphics item, which are sequentially related. This generic logical structure will act as a guide to drive the creation of document instances. The document creation process must follow the creation rules found in a document's generic logical structure.

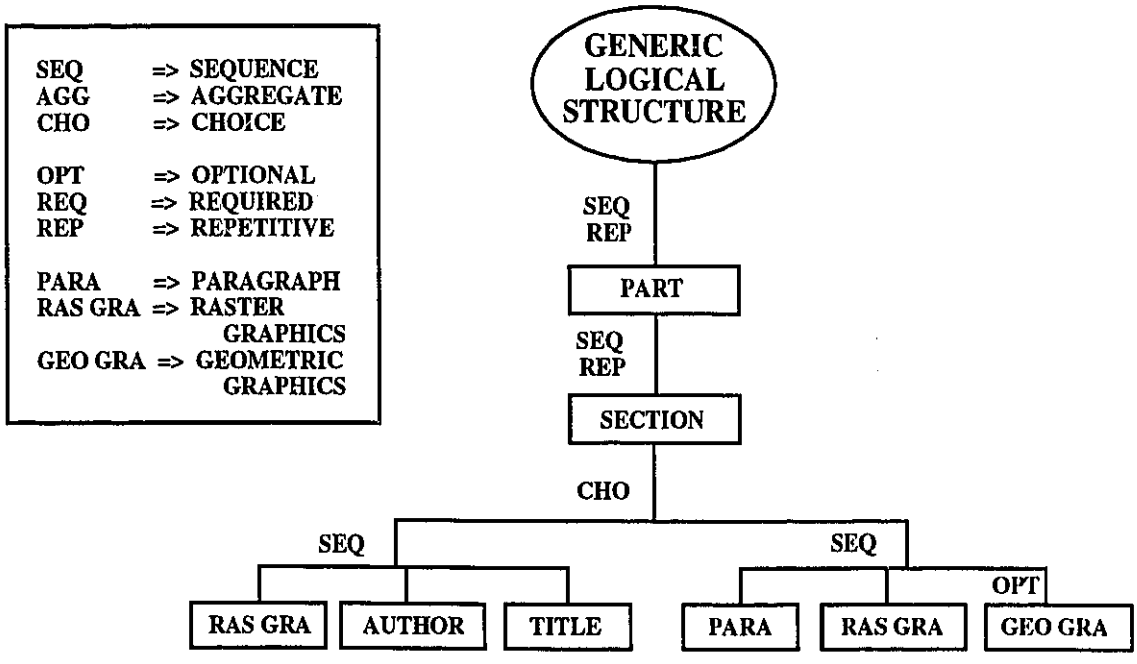


Figure 3.6. *Generic logical structure example.*

Generic layout structures are used to define layout specifications for a particular class of documents. In one sense, the creation of these structures is less complicated than that for generic logical structures, since the layout object types and hierarchical ordering are defined by ODA. It was described in the previous section that layout structures are composed of pages that can be divided into blocks, and they may also contain page sets and frames for respectively grouping pages and blocks. Layout information of a document class can be specified by describing the locations of blocks and frames for cover pages, pages for the main body of a document, company memos, etc. The purpose of a generic layout structure is to define a particular style. For example, a block at the top left-hand corner of each page may need to be reserved for a company's logo [CHA 87].

From these two generic structures, specific logical and layout structures can be created. The generic logical structure of a document is a template for the creation of specific logical structures. The information contained in a document's generic layout structure, specific logical structure, and content are used to create a specific layout structure in which content portions are assigned to specific regions on abstract pages. If a content portion is too large to be placed at the bottom of a page, it is divided into two portions of which the second portion is placed at the top of the following page. This process is called the *layout process*, and it is normally performed by a formatter [JOL 89].

An example of the two specific structures of an ODA document is shown in Figure 3.7. The *specific logical structure* of this document, which was created from the previous generic logical structure example, is found in the top half of the figure. This document contains three parts (only one is shown entirely), which are divided into sections that contain paragraphs (text) and graphics. Its *specific layout structure* (shown in the bottom half of the figure) provides a method for organizing the content of a document into blocks (areas within a page), frames (groups of blocks), and pages. These two specific structures,

logical and layout, are joined through the associated content portions. Thus, relationships have been created between basic logical objects and basic layout objects (blocks).

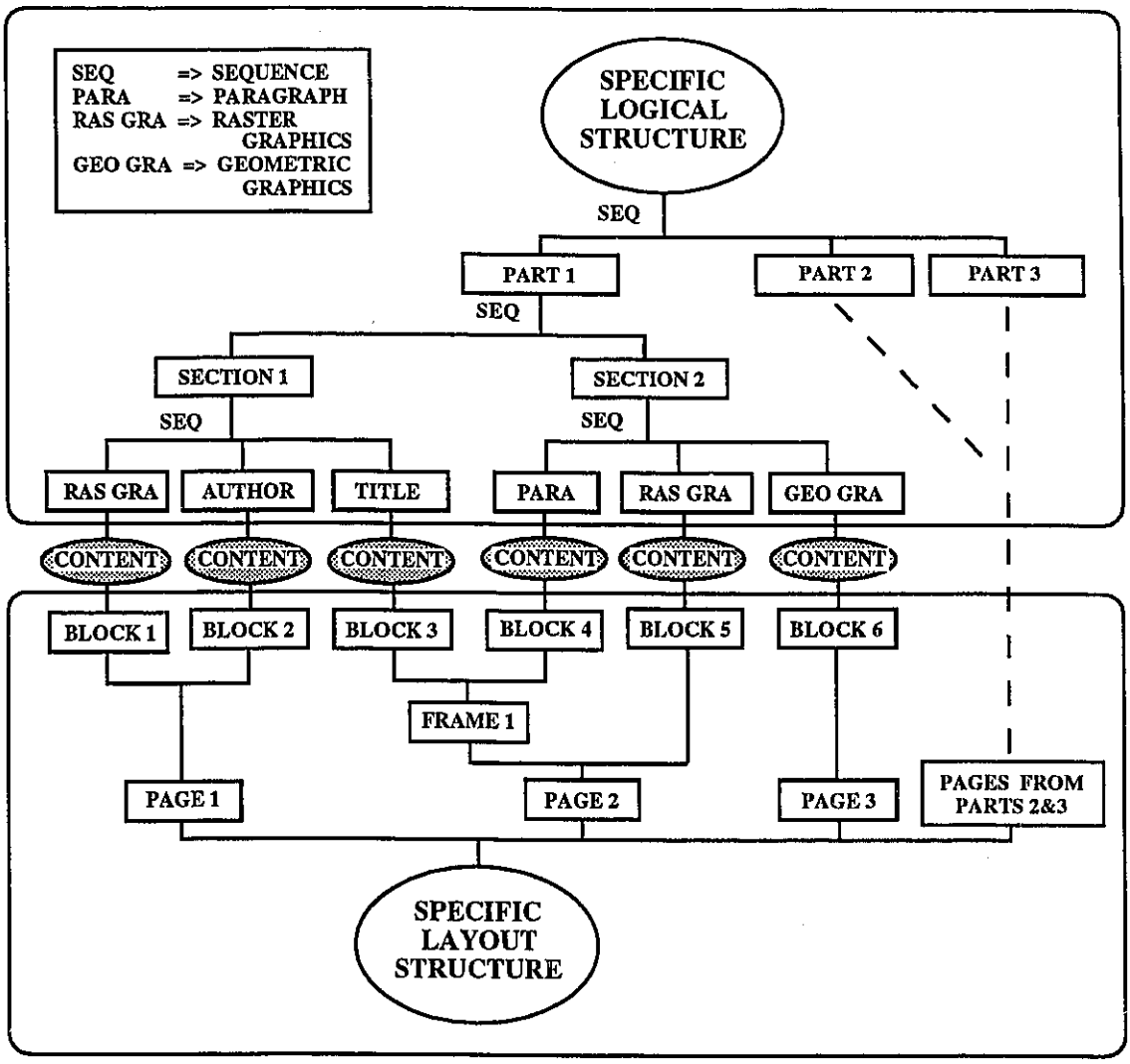


Figure 3.7. *ODA specific document structures example.*

3.3.4 Document Processing Model

ODA's document processing model (Figure 3.8) permits three different document forms, namely *processable*, *formatted*, and *processable formatted*, for the purposes of document representation. Documents can be interchanged in any one of these forms, and eventually they can be either printed out on a printer or displayed on a monitor. Conversion from one document form to another is possible by means of either the *editing* process or the *layout* process. However, any conversion must follow the rules found in the document processing model. For example, a processable document can be converted into a formatted document after completing the layout process, but the opposite conversion is not possible.

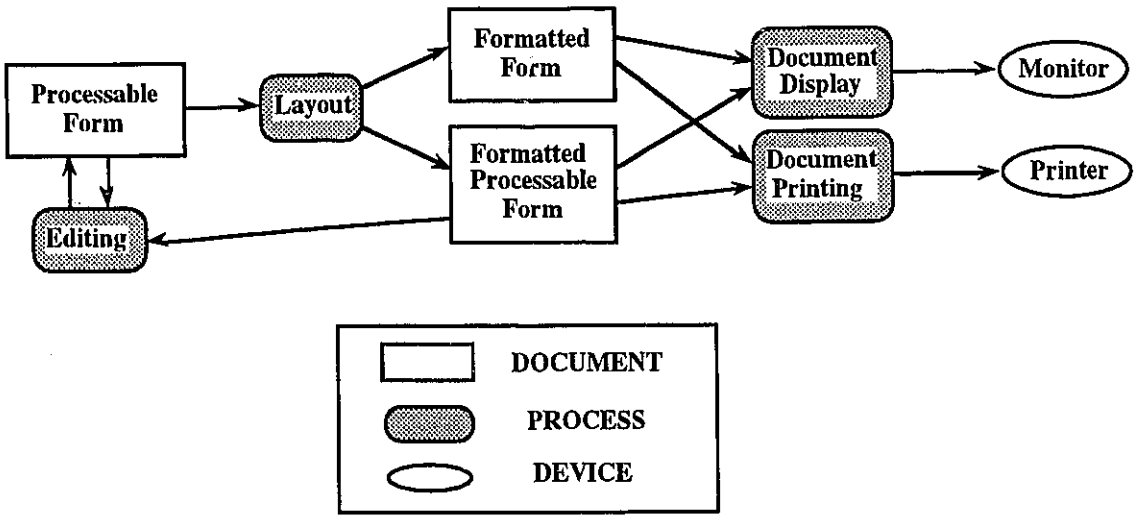


Figure 3.8. *ODA's document processing model.*

Processable documents:

An author begins by first creating and subsequently editing a document that is in a processable form. This document is represented by an editor in terms of its logical components, which can be modified by an author during the editing process. Layout and presentation styles can be added to the document's logical structure in order to describe the display characteristics of logical objects and contents, respectively. A layout style may, for example, specify that each chapter should begin on a new page. Examples of presentation style attributes for character contents are line spacing, alignment, and font. The recipient of a processable document can modify its logical structure, content, layout styles, and presentation styles.

Formatted documents:

The next step is to create the formatted version of the document, or in other words a document that can be displayed in its final form on a monitor or printer. Once this form is created, it cannot be edited (see Figure 3.8). Translation of a processable document into a formatted document is enabled by the layout process, which converts a specific logical structure into a specific layout structure. However, this process requires information contained not only in the specific logical structure, but also that found in the generic layout structure, layout styles, presentation styles, and content of the document.

The result is a document that is represented in terms of its layout components and contains all the information necessary for a printer or monitor to display it. This final conversion from a formatted document to a monitor's display or printer paper is a result of either the *document display* process or the *document printing* process (sometimes collectively called the *document imaging* process).

Formatted processable documents:

The drawback of a formatted document is that it cannot be edited by a recipient after interchange. ODA's formatted processable form permits the interchange of documents that can either be edited and reformatted or displayed by the recipient as it was intended by the originator. Thus, a document of this form must be represented in terms of both its logical and layout structures.

3.4 Extensions Required for Multimedia Documents

SGML and ODA are sometimes referred to as standards for multimedia document representation [HUN 89], since both standards support several media types such as text and graphics. However, no support has been provided for dynamic media such as audio, video, and animation. These media are useful or essential for multimedia applications such as multimedia advertising, medical information systems, and multimedia news. Furthermore, since it was shown in the previous chapter that many multimedia applications require a multimedia document as a means of assembling, organizing, and specifying presentation schedules, therefore multimedia document models should provide support for dynamic media.

The current version of the ODA standard supports only static media types: text, raster graphics, and geometric graphics. For each of these media a *content architecture* has been defined by the standard. A content architecture contains the rules for defining the representation of a particular type of content. Each architecture is described in terms of content elements, attributes, control functions, and presentation control guidelines. The common feature of the three content architectures is that they have been designed for static media types. Furthermore, ODA does not provide support for creating presentation

schedules. Indeed the document model defined for ODA applies only to documents created and manipulated using an editor package for the purpose of their layout on paper.

The document model provided by the SGML standard is more simple than that of ODA, since it does not include the definition of any content architectures. Furthermore, it does not possess an equivalent of ODA's layout structure, which describes a document in terms of layout objects such as pages, frames and blocks. SGML also provides no support for the creation of presentation schedules. Another inherent disadvantage of SGML results from the use of markup as a means of specifying the structure of a document. Since SGML markup commands are embedded into the document contents, therefore independence between document structure and contents is not achieved. This lack of independence results in several problems including poor flexibility and limited specification power [RAY 92a, RAY 92b].

Although these standards are, in their current form, not suitable for multimedia document representation, they contain may useful concepts that can be used in the design of a multimedia document architecture. Thus, ODA and SGML provide static multimedia document models that can serve as a base from which a more complete multimedia document model can be developed. MEDIADOC, the multimedia document architecture presented in this thesis, uses the ODA standard for this purpose.

Several reasons exist for choosing ODA instead of SGML; from an analysis of the two standards, it can be seen that ODA provides a more powerful model, since it includes a means of representing a document in terms of its layout components (i.e., a layout structure), it has defined content architectures for text, geometric graphics, and raster graphics, and it supports the independent representation of structure and contents. For these reasons, the ODA standard was chosen to serve as a base for the design of MEDIADOC.

Although ODA provides a good starting point from which to begin the design of MEDIADOC, it has a major deficiency in that it does not support continuous media such as audio and video. These media differ from static media in that they execute in time and thus have temporal properties. Currently ODA does not address the temporal relationships (i.e., synchronization) between objects within a document. As we will see in the following chapter, temporal relations are an integral part of the specification of presentation schedules. In order to include continuous media and to support temporal representation, ODA had to be modified and extended.

Chapter 4
Multimedia Document Architecture (MEDIADOC)

4.1 Introduction

A multimedia document architecture, called MEDIADOC, has been designed to enable the creation and representation of multimedia documents. The description of this architecture is divided into two parts. This chapter provides a description of the multimedia document creation process, the structures for representing multimedia documents, and the rendering synchronization scheme. In the next chapter, two other parts of the architecture, namely the timeline authoring tool, and system support for rendering synchronization specifications, are presented. A comparison of MEDIADOC with related work is also provided in the next chapter.

MEDIADOC is referred to as an architecture because it provides rules and guidelines for the creation and representation of multimedia documents. The first part of the description of the architecture covers the document creation process, which outlines how multimedia documents can be created with an editor that supports MEDIADOC. Next, the Global Logical and Layout Structures are presented. Although these two structures are similar to those found in ODA, several modifications and extensions were made to support continuous media. Finally, the last part of this chapter describes the purpose of and means for rendering synchronization specifications.

Current research in the field of multimedia document representation can be divided into two areas. While one area focuses on *passive multimedia documents*, the other is concentrating on *active multimedia documents*. In passive multimedia documents [CHR 86, NAF 90, GAI 93], a simple approach has been taken to solve the problem of continuous media integration; continuous media are represented in a *static visual form* and later activated by a viewer. For example, the static visual form of a video sequence is a frame, and the static visual form of audio is an icon. These visual forms are embedded into traditional documents (i.e., those containing only static media types), and they are played only after being activated by a *viewer*. A viewer is a person who is both viewing and listening to a multimedia document, which has been created by an *author*.

On the other hand, a different approach is required for integrating continuous media in active multimedia documents [BUC 92a, BUC 92b, HOE 92, LIT 90a]. In this approach, each media item is treated as an object that will be presented in time, and thus each object is rendered (i.e., presented) for a certain duration. Some objects have an implied duration. For example, audio and video sequences are normally rendered at the same rate as they were captured. On the other hand, static media do not have implied durations. However, if a duration can be assigned by an author to every object in a multimedia document, then a presentation schedule can be created to describe when each

object in the document should be presented. In MEDIADOC, these presentations schedules are referred to as *scenarios*.

MEDIADOC has been developed specifically for the representation of active multimedia documents that include presentation schedules. In fact, only very few extensions to ODA are required in order to represent passive multimedia documents, since this type of document does not include synchronization specifications. Since MEDIADOC supports the creation of active multimedia documents, the following description of the multimedia document creation process under MEDIADOC includes the specification of presentation schedules.

4.2 Multimedia Document Creation Process

The following is a description of how authors can create multimedia documents using an editor that supports the MEDIADOC architecture. Several different approaches can be employed by an author when creating multimedia documents. For example, if the content portions of a document already exist in an information repository, typically a file system or database, then document creation simply involves searching for the required multimedia objects and organizing them in a manner that reflects the type and intended use of the document. Special navigation, browsing, and querying support should be provided to minimize the time needed to search through large databases or file systems.

In our multimedia document editor project, multimedia objects are stored in a database to allow for the efficient manipulation, query, and playback of documents. Once the required multimedia objects have been located, they can be assembled in a document and structured in a manner that complies with MEDIADOC's Global Logical Structure. This structuring process may also be accompanied by an editing process, in which the various pieces of information are modified. On the other hand, the required multimedia objects may

not already exist. In this case, the author creates video, audio, text, graphics, and image content portions at a production server (Figure 4.1), which should include equipment such as video cameras, VCRs, microphones, and scanners.

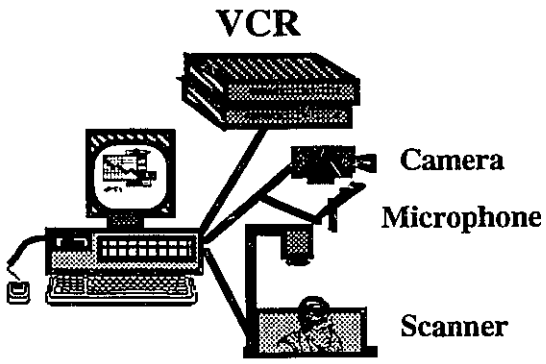


Figure 4.1. *Multimedia production server.*

At some point in time the document contents will have to be laid out. The layout process involves assigning each piece of information to a region on a display device. The resulting document layout structure must comply with MEDIADOC's Global Layout Structure. Typically the layout process will not be completed until after the document's rendering schedule, called a *scenario* in MEDIADOC, has been defined.

Scenarios provide the *rendering conductor*, the module of the multimedia document editor that is responsible for executing the document, with the information required to *playback* multimedia documents. Playback is a term used to describe the presentation of an active multimedia document, where the presentation is guided by the temporal specifications found in the document's scenario. A scenario contains information that describes when and for how long each multimedia object in a document will be rendered. Scenarios are automatically converted into Scenario Object Rendering Time (SORT) graphs by the editor so that authors can both verify scenarios and complete the layout process.

After the scenario has been verified and the multimedia objects have been laid out, the document is ready for playback.

4.3 Global Structures

4.3.1 Global Logical Structure

MEDIADOC's Global Logical Structure is the part of its architecture that provides rules for creating the specific logical structure of a document using a tree-like shape. In our multimedia document model, specific logical structures can be created without a generic logical structure. A model that incorporates document classes, which can be used as templates for the creation of multimedia document instances, is presented in [WOE 86]. The concept of document classes and instances is analogous to ODA's generic and specific logical structures, and no major changes are required for supporting multimedia document classes. Since document classes are not required for the creation of documents, they are not included in the description of MEDIADOC.

The use of *abstract objects* to represent the document in an *aggregation hierarchy* serves several purposes:

1. facilitates integration of diverse media types in a document.
2. organizes documents so that they are easy to understand and to edit.
3. provides a means of manipulating groups of objects.
4. allows the creation of attributes which can be associated with any object.
5. permits advanced queries based on attribute values and object types.
6. enables the creation of standard functions (e.g., slow motion and frame-skip for video) which can be developed independently of content architectures.

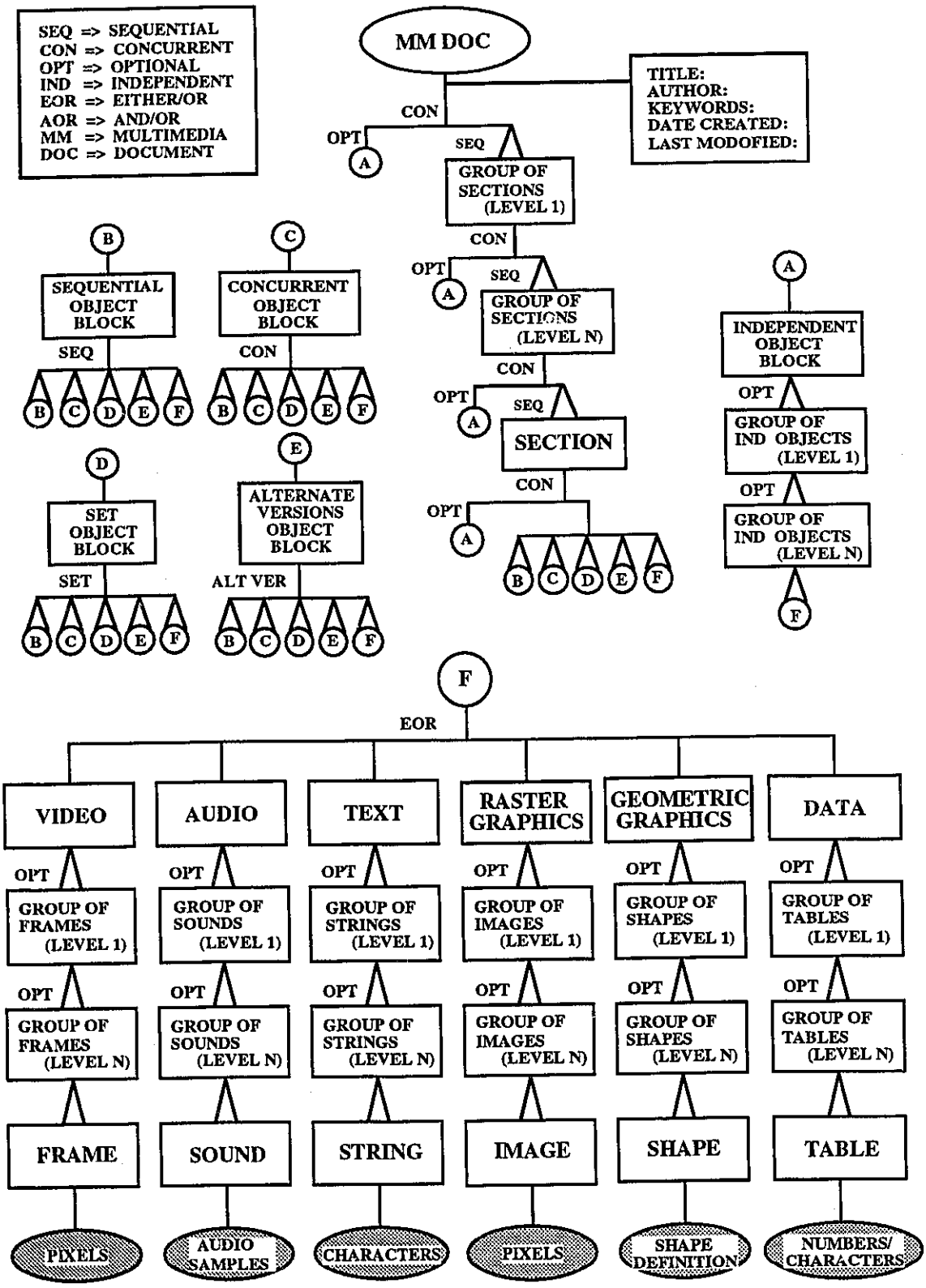


Figure 4.2. *MEDIADOC's Global Logical Structure.*

In MEDIADOC, each multimedia document has a logical structure which describes and organizes the various pieces of information that it contains. Figure 4.2 shows all the different logical object types (i.e., classes) that can be used to create a logical structure. It also provides information describing how logical hierarchies can be formed by showing the allowable children object classes for each logical object class. The Global Logical Structure contains seven main object classes: *independent object blocks* (A), *sequential object blocks* (B), *concurrent object blocks* (C), *set object blocks* (D), *alternate versions object blocks* (E), *media objects* (F), and *section-related objects*.

Media object classes (lower layers):

Media object classes (F) consist of several layers: *media type* classes (audio, video, etc.), *grouping layer* classes, *basic media object* classes (frame, sounds, etc.), and the *content architectures* for each media type class consisting of pixels, audio samples, etc. These media object classes serve several purposes:

1. provide the author with a set of pre-defined classes that, when instantiated, can be used to identify the media type of a particular piece of information in a document.

2. describe media items by means of attributes such as *number_characters*, *audio_quality_level*, and *number_frames_per_second*, which are useful not only for the author but also for the multimedia information system.

3. enable the development of functions independently from the content architectures.

Each type of content portion (ovals at the bottom of Figure 4.2) is identified by one of six basic media object classes: *frame*, *sound*, *string*, *image*, *shape*, and *table*. The frame was chosen as the basic media object class for the *video* media type class so that many functions related to video sequences could be developed independently of the video content architecture. These video functions include slow-motion, fast-motion, inter-frame

synchronization, and inter-media (e.g., voice and video) synchronization. Similarly, basic media object classes were chosen for the other media type classes: *audio, text, raster graphics, geometric graphics,* and *data.*

Optional grouping layers (maximum of N layers) are available to organize or index basic media objects. For example, the algorithms in [NAG 92] can be used for automatic indexing and searching for object appearances in video segments. The automatic video indexing function scans a video segment and marks scene changes by evaluating the difference in colour levels between successive frames, and the full-video search function detects and records the frames that contain an occurrence of a specified object. MEDIADOC's grouping layers are very useful for marking scene changes, since a group of frames could represent a scene. These grouping layers can be used not only for organizing the basic logical objects of video sequences, but also for any of the other pre-defined media types classes.

Although some multimedia data types, namely voice, music, animation, and slide shows, are not shown explicitly in the figure, they can also be included in MEDIADOC documents. As mentioned in Chapter 2, voice and music are two types of audio. Voice and music items can be represented with the audio media type class, which includes attributes for specifying the type and quality of an audio item. Similarly, animations and slide shows are very similar to video, and thus they can be represented with the video media type class.

Mid-layer object classes:

At mid-layers *sequential, concurrent, alternate versions,* and *set* object classes describe how their children are related. Sequential objects blocks define an ordering of information with objects such as chapters, sections, and paragraphs. Concurrent object blocks are useful in describing, for example, a video object and its soundtrack. When a document contains different versions of an object, an alternate versions object block is used to identify them. Finally, set object blocks provide a means for grouping information

without having to specify a specific relationship (sequential, concurrent, or alternate versions) for them.

Upper layer object classes:

Independent object blocks, which allow authors to include pieces of information without having to declare how they should be incorporated into the document, and section-related objects form the upper layers of a document's logical structure. Each section-related object includes a *type* attribute to identify its type (e.g., preface, part, chapter). In addition to a logical hierarchy, each document is accompanied with a *document profile*, which is a list of information such as the author's name, the title of the document, and keywords.

Example:

Figure 4.3 illustrates an example of the specific logical structure of a document entitled "Travel Guide". This multimedia document was created to provide a travel agency's customers with information on two of its travel packages. Finland and Italy (destinations) are shown both as keywords in the document profile and as objects joined together by the set relationship in Section 2 ("Countries") of the document. MEDIADOC permits the children of section objects to be organized with any of the available modifiers (sequential, concurrent, etc.). Traditional music, the national flag, a short video of popular sites, and fare information has been included for each destination. Section 4 ("Ending") contains some travel tips (text), and an audio (voice) "Summary". A document profile has also been created for the structure.

The recent improvements in display devices and user interfaces makes it possible for a document creator to see this graphical structure on a monitor and to manipulate the document's abstract objects by selecting graphical objects with a mouse and issuing appropriate menu commands. When documents contain a very large number of objects, special viewing techniques should be implemented to facilitate document creation and editing. Document structures should be viewed by section or subsection instead of in their

entirety, thereby decreasing the number of objects that are displayed at one time. The multimedia document editor's graphical user interface should also provide important features, such as scrolling and zooming.

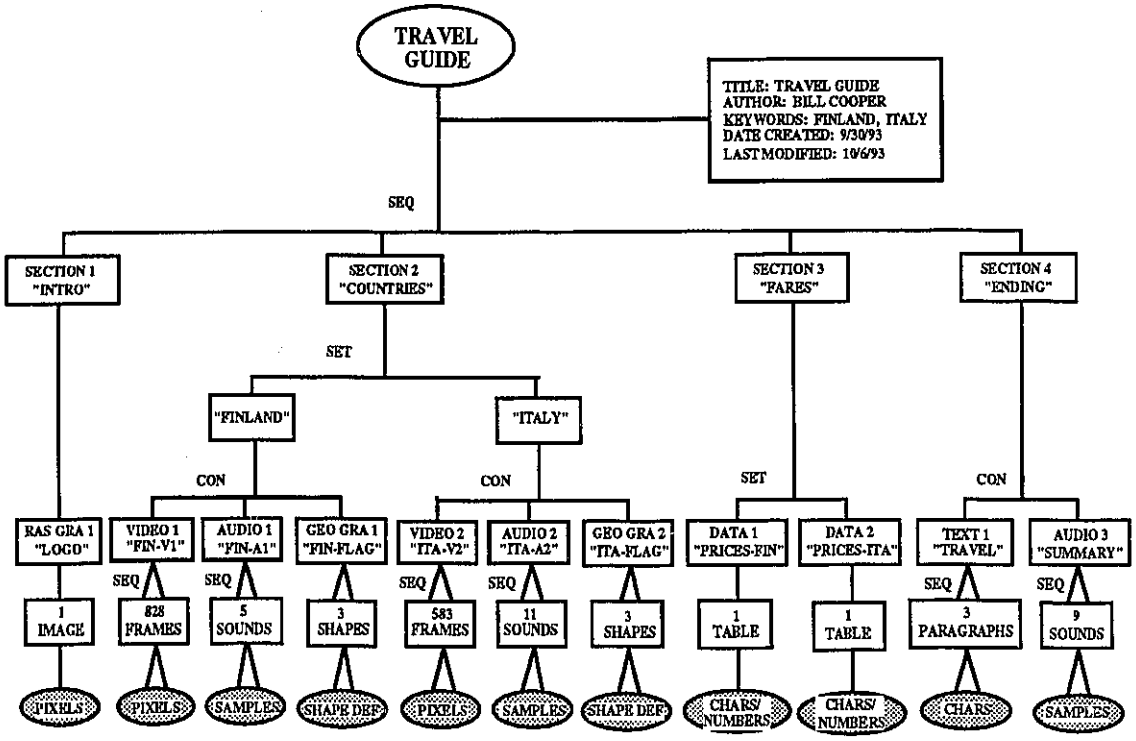


Figure 4.3. *Logical structure for Travel Guide example.*

4.3.2 Global Layout Structure

A document's layout structure describes the spatial properties of the document's media objects that will be presented during playback. By assigning a document's content portions to blocks on a display device, the author defines links between basic media objects and basic layout objects. The rendering conductor will send the pieces of information to the physical locations specified by those links. MEDIADOC's Global Layout Structure is shown in Figure 4.4. This global structure provides the rules for creating a document's layout structure.

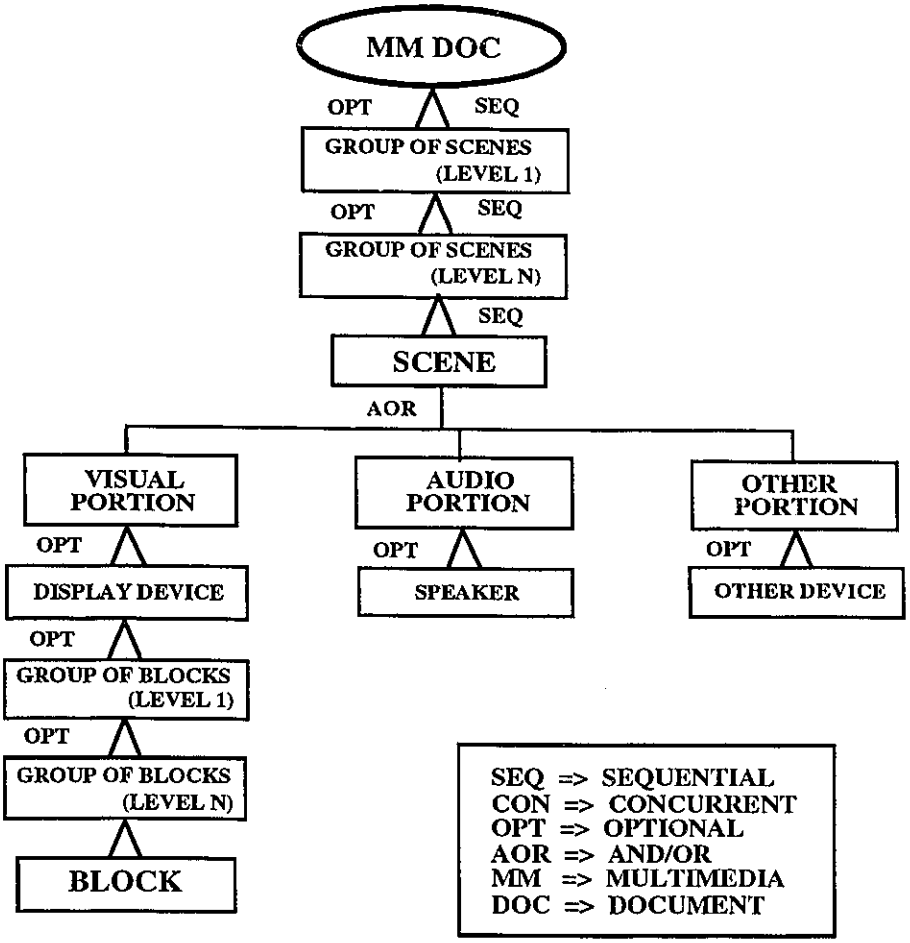


Figure 4.4. *MEDIADOC's Global Layout Structure.*

Media group types:

Visual objects are separated from audio objects since their content portions must be physically laid out for presentation on a display device such as a printer or a monitor. Basic visual media objects are assigned to blocks whose position and extent are specified by the author. This layout procedure is not required for audio information since it will be rendered by a speaker which is a linear device.

Multiple devices layer:

An optional multiple devices layer allows an author to specify which media objects should be sent to which devices when several devices are available for a particular media

group (i.e., visual or audio). For example, in a medical application, a physician might send three x-ray images, representing three views of a patient's chest, to three different monitors for simultaneous viewing.

Scene layers:

Finally, at the highest layers of the layout structure, a document is divided into scenes or groups of scenes.

In contrast with ODA, page layout objects do not exist in MEDIADOC since they are not meaningful in active multimedia documents. ODA documents require page layout objects for the imaging process. Pages are displayed or printed one at a time by using the layout information contained in page layout objects and their subordinates including the links to content portions. In contrast, the playback of active multimedia documents is characterized by a presentation which changes continuously in time. Thus temporal information, as opposed to layout information, dominates the rendering process in active multimedia documents.

For this reason, hierarchical generic and specific layout structures are not required for rendering active multimedia documents. Instead, layout information can be added to the presentation instances of media objects rather than in a hierarchical structure. We will see how this assignment can be realized in the next section. A generic layout structure is also not required for active multimedia documents since these documents include neither the notion of pages (and page sets) nor a layout process for generating the assignment of content portions to specific blocks on a sequence of pages. However, a specific layout structure can be maintained in order to apply layout information at the scene level.

Page layout objects could be added to our multimedia document architecture for printing documents. In order to print a multimedia document, text, graphics, images, and a few video frames (treated as images) are selected and laid out on abstract pages to produce

a printable form of the document. Page layout objects could also be used for supporting the creation of passive multimedia documents. KWrite [GAI 93] is an example of a multimedia document publication system that divides its passive multimedia documents into pages in the same way as conventional word processors. Multimedia material, which has been embedded into these text-based documents, is rendered only when activated by a viewer. However, the orchestrated presentation of multimedia objects is not possible for these passive multimedia documents, since the document's content has been assigned to pages instead of instants in time.

4.4 Playback

In order to include continuous (time-based) media in active multimedia documents, a schedule that drives the playback of the document must be created. Playback consists of presenting the media objects of a document according to this schedule. Synchronization mechanisms are required so that these objects can be rendered at the appropriate time instants. In this section, the rendering synchronization scheme that has been developed for MEDIADOC is described. Although synchronization can be applied to many different levels, we will only investigate *synchronization at the presentation level.* This type of synchronization allows authors to create a presentation schedule by specifying the temporal information of the media objects in a document.

This schedule, referred to as a *scenario* in MEDIADOC, describes when and for how long each media object should be rendered. A media object that appears in a scenario is called a *scenario object.* Scenarios are essential for the playback of active multimedia documents, since they provide a means of integrating static (i.e., text and graphics) and continuous (i.e., audio and video) media. An author creates a scenario by providing the document editor with a list of temporal specifications that adequately describes the desired rendering schedule for a multimedia document.

4.4.1 Temporal Specifications

Each temporal specification must include one of the three available temporal characteristics of scenario objects: *start*, *end*, and *duration*. The start of an object represents the time at which the rendering conductor should begin to present the object. Similarly, the end of an object describes when the object should no longer be rendered. The start and end of each object are referred to as its *endpoints*. An object's duration is simply the amount of time that it will be rendered. These three temporal characteristics can be represented in a graphical form as shown in Figure 4.5.

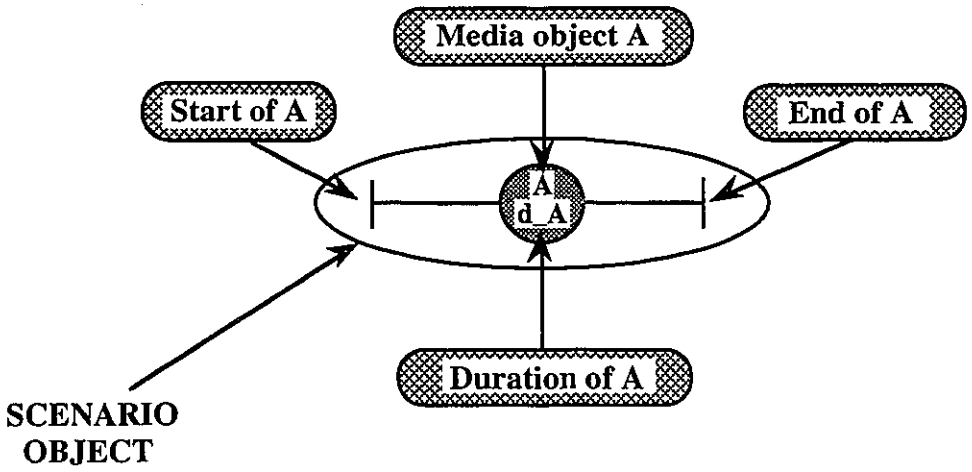


Figure 4.5. *Graphical representation of a scenario object.*

This representation is based on that which is used in Object Composition Petri Nets (OCPN) [LIT 90a]. Each graphical scenario object consists of two vertical bars, a circle, and a horizontal line which joins the circle with the two vertical bars. The circle contains a media object (e.g., A), and possibly a duration (e.g., d_A) if one has been assigned to the scenario object by means of a *duration specification*. The vertical bar on the left-hand side of the scenario object represents the start of the object, and the vertical bar on the scenario object's right-hand side is the end of the object.

Two methods exist for specifying the start and end times of scenario objects. The first method uses timelines (Figure 4.6) for specifying start and end times explicitly. In timeline-based systems, media objects for each scene are placed by an author along a single timeline [OGA 90]. This binding of media objects to a timeline leads to inefficient scenario editing. For example, if an author wishes to modify a scene by deleting a sequential object, he/she would have to "manually" decrease the start and end times of each object which follows the deleted object to remove the gap in the presentation schedule.

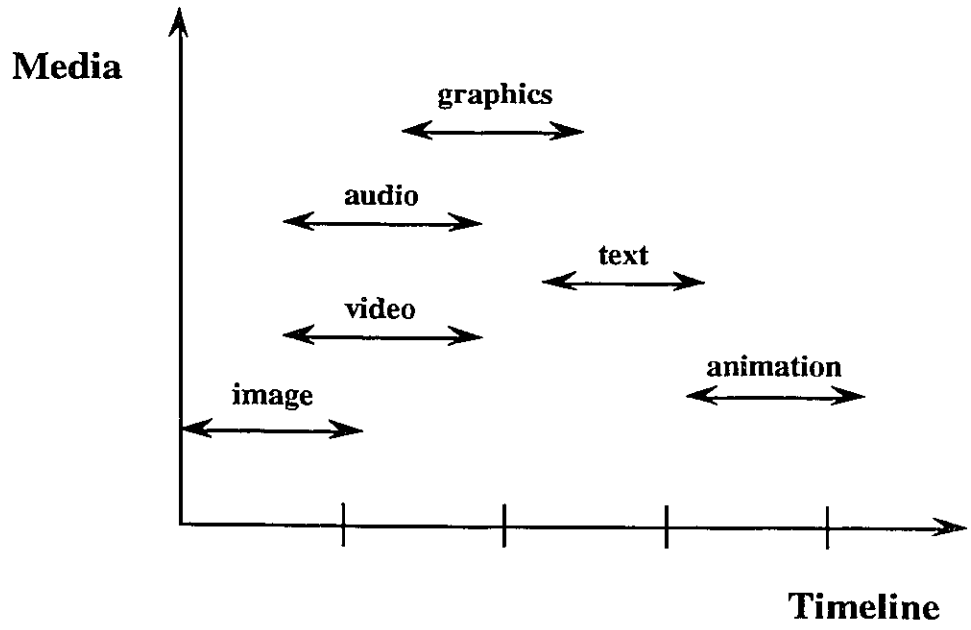


Figure 4.6. *Timeline presentation schedule.*

Furthermore, timeline-based methods are not well suited for handling unpredictable events, such as user interaction, since start and end times are fixed to points on the timeline. Specifically, it is difficult for authors to include unpredictable events in their timeline presentation schedules. For example, an author might want to specify that the end of an object should occur when a user clicks a mouse. This is particularly useful for static media, since it is normally not known for how much time an author will want to either read a segment of text or view an image.

In the second method, temporal relations are used to specify how the endpoints of each scenario object are temporally related to those of one or more other scenario objects. These relations, which help to eliminate the problems of timeline based methods by means of automatic synchronization adjustment and independence from explicit start and end times, serve as a powerful synchronization mechanism. For these reasons, temporal relations are also an integral part of many other synchronization models [LIT 90a, BUC 92a, BUC 92b, HOE 92].

Our model provides three temporal relation types: *before* (<), *equals* (=), and *after* (>). When a scenario object endpoint appears in a temporal specification, it is referred to as a *synchronization specification*; the following is an example: **Start of A is 5 seconds > End of B.** "Start of A" is called the *resulting event*, "End of B" is referred to as the *synchronizing event*, and "5 seconds" is the *offset delay*. This synchronization example has scheduled the rendering of object A to begin five seconds after object B is no longer being rendered.

MEDIADOC's graphical synchronization model also includes the explicit representation of synchronization specifications, which are placed inside rectangles appearing directly on the left-hand and/or right-hand side(s) of scenario objects. A rectangle on the left-hand side of a scenario object applies to the object's start, while the rectangle on the right-hand side involves the end of the object. A directed arc joins the two endpoints of a synchronization specification; the arc points to the endpoint that occurs later than the other. Clearly an arc is not required for the equals relation, since the two endpoints will occur simultaneously during rendering. Figure 4.7 shows the set of all possible combinations for synchronizing an endpoint of one object (A) to an endpoint of another object (B). The graphical representation of the previous synchronization specification example is shown in Figure 4.8.

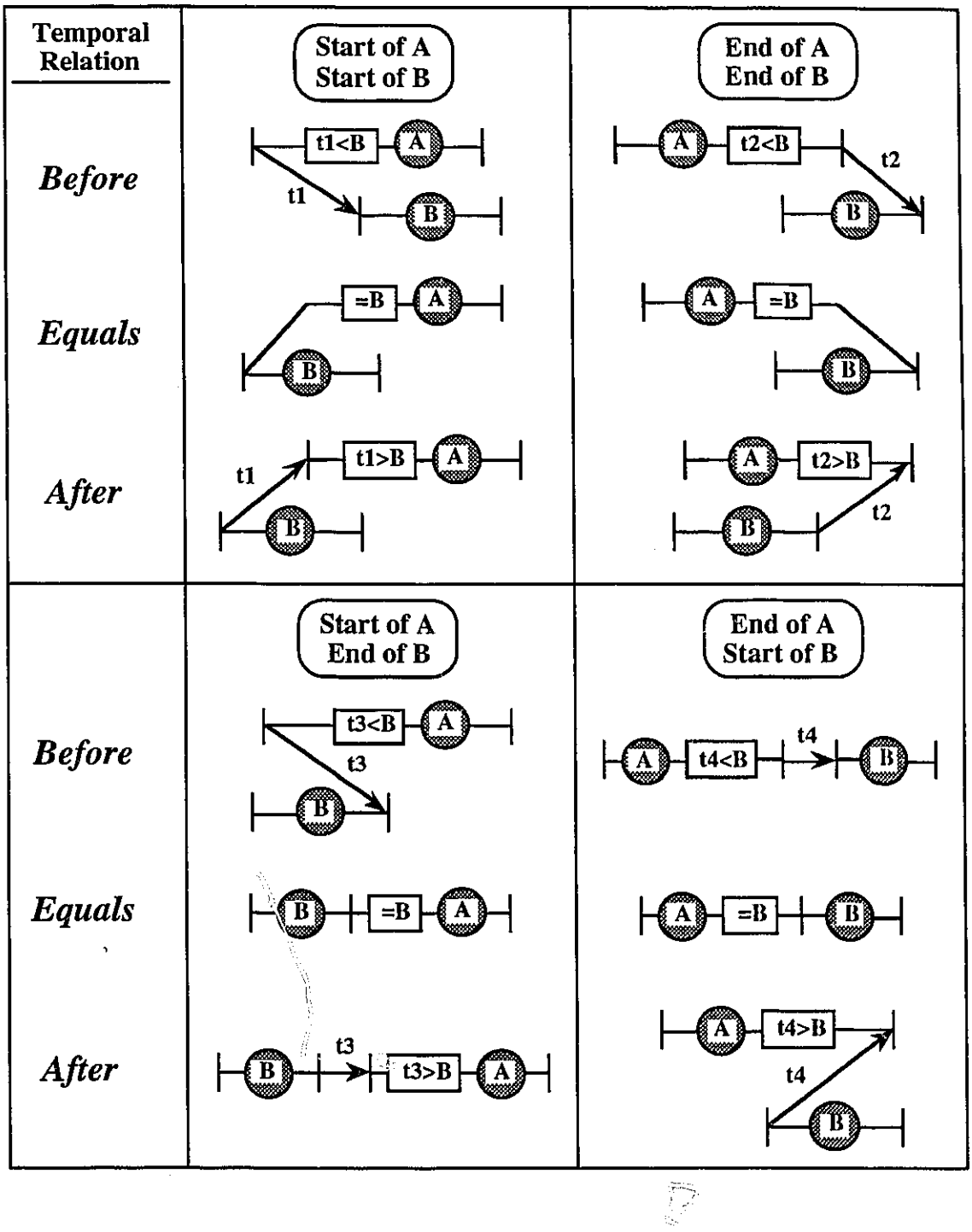


Figure 4.7. *Set of temporal relation combinations for two scenario objects.*

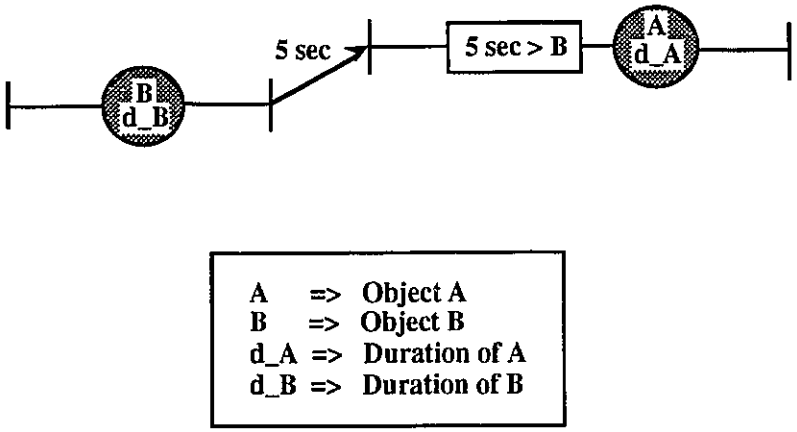


Figure 4.8. *Graphical representation for a synchronization specification.*

Our rendering synchronization scheme also permits the inclusion of several synchronizing events in a temporal relation by means of a multiple synchronizing event definition. An author chooses a number N representing the Nth event from a set of M synchronizing events. An example of such a definition in a synchronization specification is: "**End of A is = 2 (Start of B, End of C, Start of D)**" where N = 2 and M = 3. In this example, object A will no longer be rendered exactly when the second (in time) of the three synchronizing events occurs. Although it is expected that the two most common usages of this definition will be N = 1 and N = M, N is permitted to be any number in the range: $1 \leq N \leq M$.

An important time dependency for the presentation of media objects is known as *continuous* synchronization [STE 90]. The most common form of continuous synchronization is called the "lip-sync" condition, which requires that the voice emanating from a person's mouth be played simultaneously with the movements of the person's lips [NIC 90]. Clearly, a synchronization mechanism is required to ensure that each video frame is displayed simultaneously with the associated voice samples in order to avoid a degradation in the quality of the presentation. The unique characteristic of this type of

synchronization is that it be performed continuously during the entire display of the voice and video segments. Although our model permits the specification of continuous synchronization, the mechanism through which it is achieved is the responsibility of the rendering conductor and it is therefore not described in this thesis.

MEDIADOC recognizes five types of synchronizing events: *time of day*, *scene endpoint* (start or end), *scenario object endpoint* (start or end), *viewer input*, and *other events* (occurrences that are not included in the other synchronizing event types). The viewer input synchronizing event is particularly important since it allows user interaction during the playback of multimedia documents. Clicking a mouse button is an example of user interaction. In Summary, an author can create temporal specifications by means of temporal relations, synchronization specifications, and duration specifications.

4.4.2 Rendering Scenarios

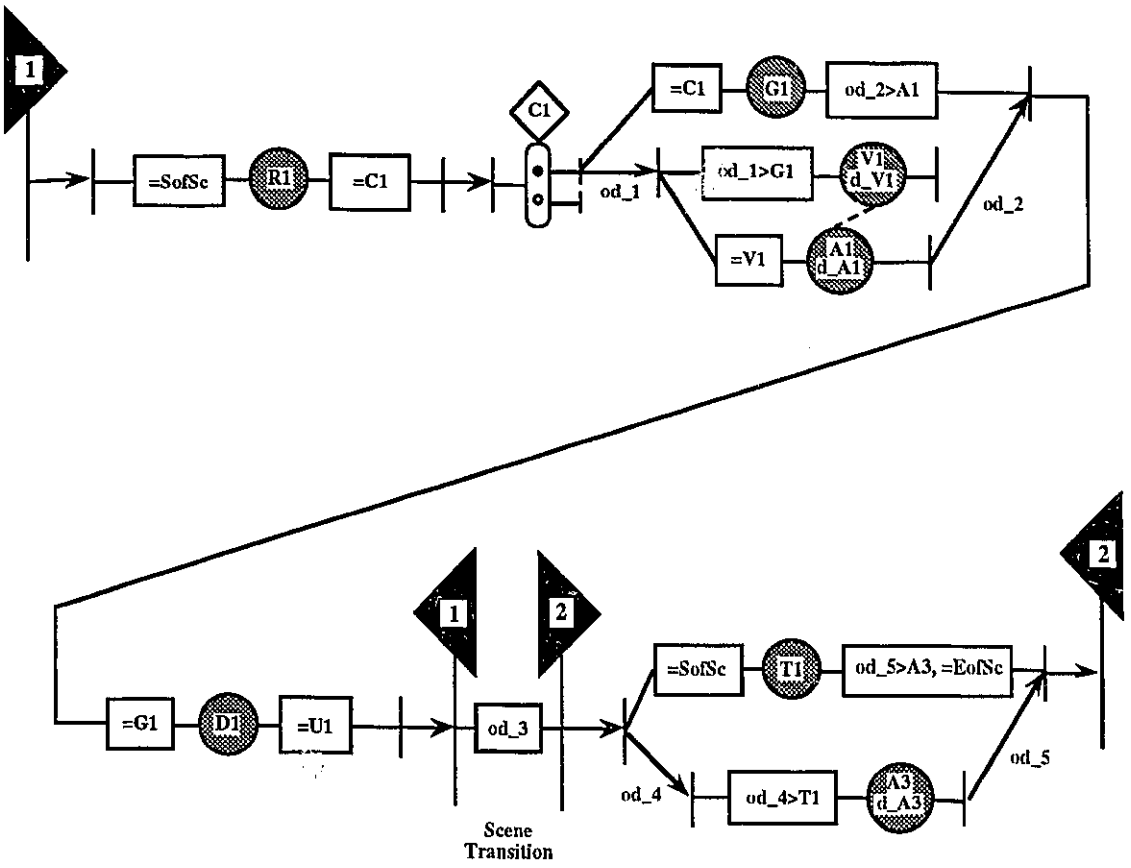
The collection of all the duration and synchronization specifications is the scenario of a document. An author can either modify an existing scenario or create a new one that meets his/her desired rendering schedule. Rendering scenarios are formed by selecting media objects from a document's logical structure and by adding temporal information to those objects. Scenarios are divided into several scenes, and media objects within each scene are temporally related.

A media object that is selected for a scenario is referred to as a *scenario object*. This distinction is important since it emphasizes the notion of the object's role. Media objects, which are found in the logical structure, represent pieces of information contained in the document. On the other hand, scenario objects, which are the presentation versions of their associated media objects, contain temporal information for rendering. Since they are presentation instances, layout information can also be added to scenario objects in order to describe their position and extent.

Each multimedia document can have several scenarios representing different ways that it can be rendered. Rendering scenarios can also be converted from a list of temporal specifications into a *graphical format*. The graphical format of a scenario for the Travel Guide document (presented earlier) is shown in Figure 4.9. A condensed version of the document's logical structure has been included in the figure as a list of sections. The name of each section's media objects as well as their associated scenario object abbreviations (e.g., A1, V1, etc.) have also been included in the list.

Playback for the Travel Guide begins with the travel agency "Logo". This image will remain on the display device until the viewer has chosen (C1) between the travel information on "Finland" and "Italy". The black circle in the choice object (C1) indicates that the scenario in the figure shows the *use-case* for the first option (Finland). The Finnish flag, traditional music, and a video segment will then be rendered. Continuous synchronization between the music (audio) and video objects is graphically represented with a dotted line joining the two media objects. Next, the fare information (table) will be displayed until the viewer clicks the mouse button (an event represented by U1). Playback concludes with an audio "Summary" and the textual list of travel tips entitled "Travel".

Graphical scenarios, like the one shown in Figure 4.9, are not only a means of visualizing scenarios from the point of view of a synchronization model, but they can also be displayed to authors using a graphical user interface. A multimedia document editor should provide a graphical scenario authoring tool so that authors can create and modify scenarios. Alternatively, the document editor could display the textual list of temporal specifications for a scenario creator. However, the graphical format is a better representation compared to the list format, since the effect of adding/deleting a temporal specification is more obvious in graphical scenarios, which both organize the temporal information by scenario object and simultaneously show all the synchronization relationships in which each scenario object participates.



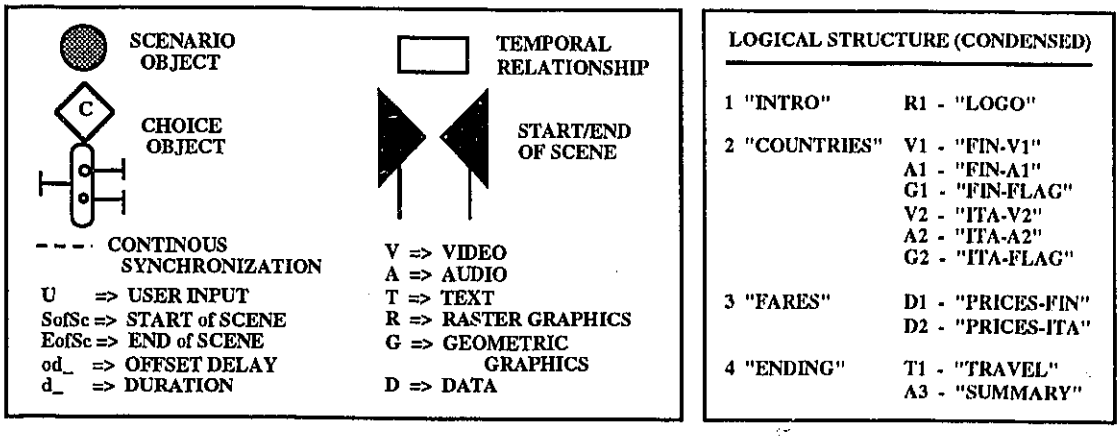


Figure 4.9. *Scenario for Travel Guide example.*

4.5 Summary

MEDIADOC contains rules and guidelines for the creation and representation of multimedia documents. The Global Logical and Layout Structures serve several purposes. Firstly, they help authors to organize documents in terms of logical components and layout components, respectively. It was shown that the layout structure need not be represented as a hierarchy, but instead layout information can be associated with presentation versions of media objects. On the other hand, the logical structure is maintained as a hierarchy and it assists authors in organizing their documents and integrating static and dynamic media.

In the design of this multimedia document architecture, particular attention was paid to providing support for the creation and representation of presentation schedules. It was shown that temporal relations can help authors to create these presentation schedules, which we call scenarios. Temporal relations are more useful than timelines for synchronizing the presentation of multimedia objects, since they are able to capture timing relationships among presentation objects. These temporal relations also provide the opportunity for describing unpredictable events such as user interaction.

However, the use of temporal relations adds more complexity for authoring documents. Although MEDIADOC's graphical scenarios makes it possible for an author to create and edit temporal specifications, they do not provide information concerning the actual start and end times of each scenario object. Thus, the author has no means of verifying that the temporal specifications contained in the scenario will produce the intended presentation. Furthermore, an author might accidentally create some temporal specifications that either contradict each other or are impossible to render. In the next chapter, Scenario Object Rendering Time (SORT) graphs and system support for scenario creation, which help to alleviate some of the problems associated with temporal relations, are discussed. A comparison of MEDIADOC with related work is also provided.

Chapter 5
Scenario Verification and Layout Completion

5.1 Introduction

In Chapter 4, the multimedia document creation process was described. An author begins the creation of a multimedia document by selecting media objects from an information repository, such as a database or file system, and/or by creating content portions on a production server using, for example, a camera, a VCR, a microphone, and a scanner. These media items can then be assembled into a document and organized by creating a logical structure for the document.

More media items can be created and added to this structure, which can be modified in order to accommodate those new media objects. Next, the document's scenario is created

by first selecting media objects from the logical structure, and then by adding temporal information to those objects. Synchronization specifications, which make use of temporal relations, and duration specifications constitute the temporal information contained in each scenario.

Once a scenario has been created, two tasks remain before the document can be rendered. Firstly, the author must verify the correctness of the scenario. In other words, he/she must check that the scenario matches the intended rendering schedule. Secondly, the author must complete the layout process. However, before the scenario objects can be laid out properly it is imperative that the author knows which objects will be rendered simultaneously.

MEDIADOC provides support for the tasks of scenario verification and layout completion in two ways. Firstly, Scenario Object Rendering Time (SORT) graphs have been developed to show a timeline view of the scenario. Secondly, our model describes the system support that can be used to verify scenarios. In this chapter, SORT graphs and system support for scenario verification will be discussed. A comparison of MEDIADOC with related work is also provided at the end of the chapter.

5.2 Scenario Object Rendering Time (SORT) Graphs

Scenario Object Rendering Time (SORT) graphs, which are provided by MEDIADOC as an authoring tool for a multimedia document editor, can be used for both scenario correctness verification and the layout of scenario objects. These graphs are created by the document editor to provide authors with a timeline view of a scenario. Thus, our multimedia document model allows authors both to create scenarios using temporal relations, and to use SORT graphs as an analysis tool. The result is that authors can take advantage of the benefits derived from using not only temporal relations but also timelines.

In this section, an example of a SORT graph is presented, followed by a description of the scenario-to-SORT conversion process and a discussion of the benefits that result from the use of this type of graph.

5.2.1 Example

The SORT graph for the multimedia Travel Guide's rendering scenario is shown in Figure 5.1. In SORT graphs, scenario objects are represented as rectangles positioned along a timeline. Variable duration scenario objects are differentiated from those that have a fixed duration by a hatched background. In Figure 5.1, R1 and D1 each have a variable duration, which cannot be determined before playback since their rendering ceases only when the viewer has clicked a mouse button (as specified in the scenario). The document's other scenario objects each have a fixed duration since they either have or are related to an object that owns a duration specification.

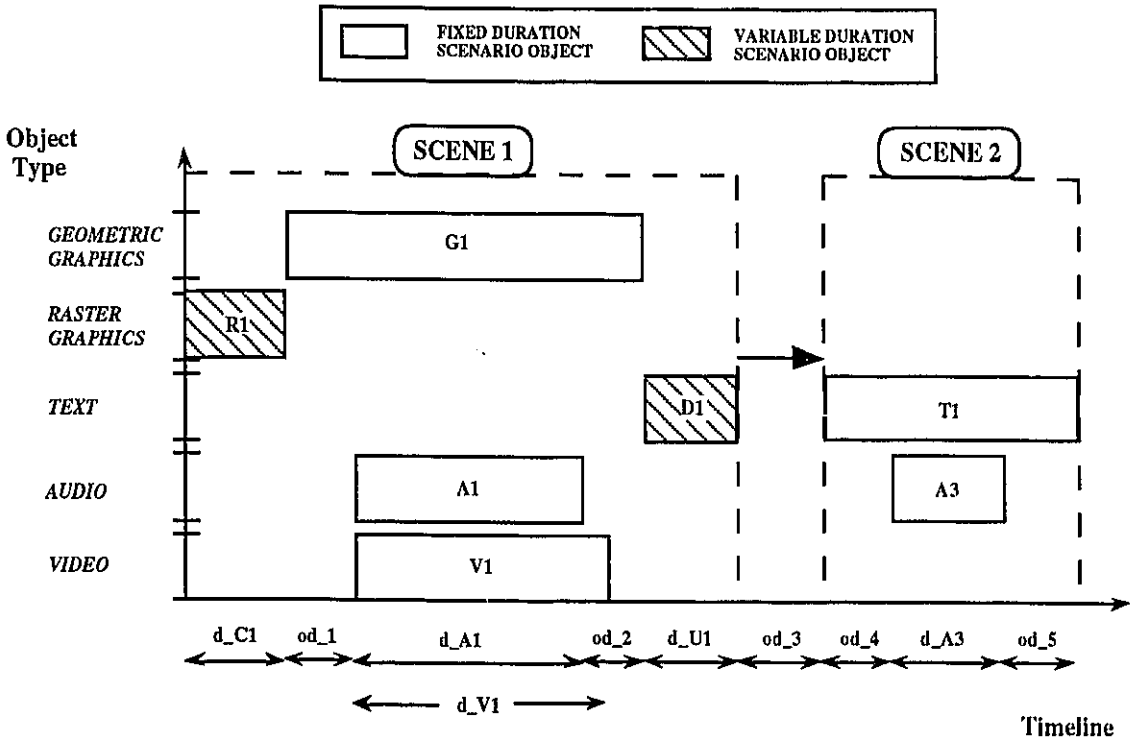


Figure 5.1. *SORT graph for Travel Guide example.*

5.2.2 Converting a Scenario into a SORT Graph

The position of each rectangle with respect to the timeline is determined by the temporal specifications found in the associated rendering scenario. In other words, a scenario is converted into a SORT graph. This conversion is possible if the temporal specifications are represented as a set of linear equations, which can be solved simultaneously to yield both the start time and end time of each scenario object. If these linear equations are placed in a matrix, row reduction can be used to produce the required start and end times. One matrix is required for each scene, since it has been defined that the temporal information in each scene can only reference scenario objects within the scene.

The terms in each row of a scene matrix are the variables representing the start and end of each scenario object as well as those of the scene. Thus, the matrix for a scene containing N scenario objects will have (2N + 2) columns representing the left-hand side of each equation, and one column for the constants on the right-hand side of the equation. The matrix also has at least (2N + 2) rows, since the number of rows must be greater or equal to the number of columns so that the values of the start and end times can be determined.

The matrix and solved start and end times of the scenario objects for scene 1 of the travel guide SORT graph are shown in Figure 5.2 and Figure 5.3, respectively. The start of an object is denoted by "S", and the end of an object is denoted by "E". For example the start and end of object R1 are represented respectively by S_R1 and E_R1. The matrix for scene 1 of the travel guide is shown in Figure 5.2. The temporal specification associated with each row of the matrix is also shown, and row numbers are displayed on the left-hand side of the matrix. The first row of each matrix identifies the start of the scene as the first instant (i.e., time = 0).

Legend:

S : Start E : End d : Duration od : Offset delay

Terms for each row in matrix:

S_Scene, S_R1, E_R1, S_G1, E_G1, S_A1, E_A1, S_V1, E_V1, S_D1, E_D1, E_Scene

Row#	Matrix for Scene 1													Temporal Specifications
1.	[1	0	0	0	0	0	0	0	0	0	0	0	]	S_Scene = 0
2.	[-1	1	0	0	0	0	0	0	0	0	0	0	]	S_R1 = S_Scene
3.	[0	-1	1	0	0	0	0	0	0	0	0	0	]	d_R1 = d_C1
4.	[0	0	-1	1	0	0	0	0	0	0	0	0	]	S_G1 = E_R1
5.	[0	0	0	-1	0	0	0	1	0	0	0	0	]	S_V1 = S_G1 + od_1
6.	[0	0	0	0	0	0	0	-1	1	0	0	0	]	d_V1
7.	[0	0	0	0	0	1	0	-1	0	0	0	0	]	S_A1 = S_V1
8.	[0	0	0	0	0	-1	1	0	0	0	0	0	]	d_A1
9.	[0	0	0	0	1	0	-1	0	0	0	0	0	]	E_G1 = E_A1 + od_2
10.	[0	0	0	0	-1	0	0	0	0	1	0	0	]	S_D1 = E_G1
11.	[0	0	0	0	0	0	0	0	0	-1	1	0	]	d_U1
12.	[0	0	0	0	0	0	0	0	0	0	-1	1	]	E_Scene = E_D1

Figure 5.2. *Matrix for scene 1 of Travel Guide example.*

The other rows represent the duration and synchronization specifications in scene 1 of the scenario. The specifications in the figure are represented with variables instead of values for durations and offset delays so that they can be more easily identified. For example, the duration specification, which can be found in row 8, is simply: d_A1. Normally, a value would be given for this duration (e.g., d_A1 = 53 seconds). In order to represent this specification in the matrix, we have to make use of the equation relating the

start, end, and duration of a scenario object. The equation is **Duration = End - Start**, and thus **E_A1 - S_A1 = d_A1**. For scenario objects whose duration is variable (e.g., R1 from Figure 5.2), either a random or an author-specified value is required so that the start and end times can be determined enabling the creation of a SORT graph.

Row 5 contains an example of a synchronization specification. The original temporal specification is: "**Start of V1 is od_1 after (>) Start of G1**", which can be represented by the following equation: **S_V1 = S_G1 + od_1** (see row 5). This equation can be rearranged (i.e., **S_V1 - S_G1 = od_1**) so that it can be easily represented in the matrix. Row reduction can be used to produce the reduced row echelon form of the matrix. The last column of this matrix represents the solution for each of the variables in the matrix. The value for each start and end time is shown in Figure 5.3. Clearly, each value is a function of one or more durations and offset delays. It is important to note that these values are dependent on the duration values chosen for scenario objects with variable durations.

S_Scene	=	0
S_R1	=	0
E_R1	=	d_C1
S_G1	=	d_C1
E_G1	=	d_C1 + od_1 + d_A1 + od_2
S_A1	=	d_C1 + od_1
E_A1	=	d_C1 + od_1 + d_A1
S_V1	=	d_C1 + od_1
E_V1	=	d_C1 + od_1 + d_V1
S_D1	=	d_C1 + od_1 + d_A1 + od_2
E_D1	=	d_C1 + od_1 + d_A1 + od_2 + d_U1
E_Scene	=	d_C1 + od_1 + d_A1 + od_2 + d_U1

Figure 5.3. *Start and end times for scene 1 of Travel Guide example.*

5.2.3 Purposes of SORT Graphs

Once an author has created a scenario for a multimedia document, he/she should verify that the temporal specifications contained in the scenario will result in a presentation that matches his/her desired rendering schedule. A SORT graph can be used for this purpose, since it shows the start and end times of each object. Another way to verify that the scenario has been correctly specified is to playback the document. However, this may be very time consuming, especially for documents with large durations. SORT graphs, on the other hand, provide a more efficient means of checking a scenario. If the SORT graph reveals that the scenario is incorrect, then the author must modify the scenario.

On the other hand, if the scenario matches the author's desired schedule, then he/she can proceed with final layout specification. In this phase of document creation, the author declares the position and extent of each media object. In order to do this, the author must know which objects will be rendered simultaneously so that he/she can decide how to lay out the objects in a manner that best displays those objects by maximizing the size of the visual media objects without causing overlap. This information can be obtained by taking time-slices over the duration of an object's rendering time in the SORT graph. Finally, after reviewing this information, the author can perform final layout specification by modifying the spatial properties of the visual media objects that intersect the time-slice markers on the SORT graph.

Another benefit derived from the use of SORT graphs is that they permit the rendering of time-selected portions a scenario. By positioning two markers along the time axis of a SORT graph, an author can request the rendering of only a part of a scenario. This selection capability is analogous to the *page-selection* or *print-selected-pages* function in traditional documents, and it is particularly useful when the author wants to playback only a small portion of scenario that has a large duration.

5.3 System Checking for Illegal Temporal Specifications

Multimedia document scenarios have the potential of becoming very large and complicated especially as the number of scenario objects and/or the degree of concurrency increases. Although temporal relations provide authors with helpful synchronization specification power, they also cause some problems since they allow the possibility of creating a scenario in which one or more of the scenario objects are unrenderable. A scenario object is unrenderable if, for example, the temporal specifications imply that the end of a scenario object should occur before the start of the same scenario object. Clearly this situation represents an illegal specification since the rendering system must start presenting a multimedia object before it can cease to render the object.

Multimedia document systems supporting MEDIADOC should perform synchronization checks to detect illegal temporal specifications. Detection occurs following every temporal specification entered by an author. If an illegal specification is detected, the system informs the author of the error and it rejects the specification. The purpose of this checking for and rejection of illegal specifications is to transfer the burden of debugging scenarios from the author to the multimedia document editing system.

5.3.1 Negative Durations

Clearly all durations must be positive in order for them to be meaningful to a multimedia rendering system. At this point it is important to recognize that the system must monitor two types of durations: *explicit* and *derived*. Figure 5.4 will help to illustrate the difference between the two duration types. In this example, object B's duration is explicit since it appears in the list of temporal specifications (line 1). However object A's duration must be derived from object B's duration by means of the two synchronization specifications (lines 2 & 3).

Detecting negative explicit durations involves the simple task of checking the sign of the duration's value (d_B in this example). On the other hand, this verification is not so simple in the case of derived durations. Firstly, the value of the derived duration must be determined. In the example, object A's duration is produced from the equation **d_A = d_B - (od_1 + od_2)**. Then a sign verification is performed on the derived duration (d_A in the example). In more complicates examples, where the endpoints of an object are synchronized to two different objects, extra steps are required. The first step is to find a chain of objects in the scenario that joins those two synchronization points. Then the chain is traversed and a summation of all the explicit durations and offset delays yields the required duration. Finally, sign verification can be performed.

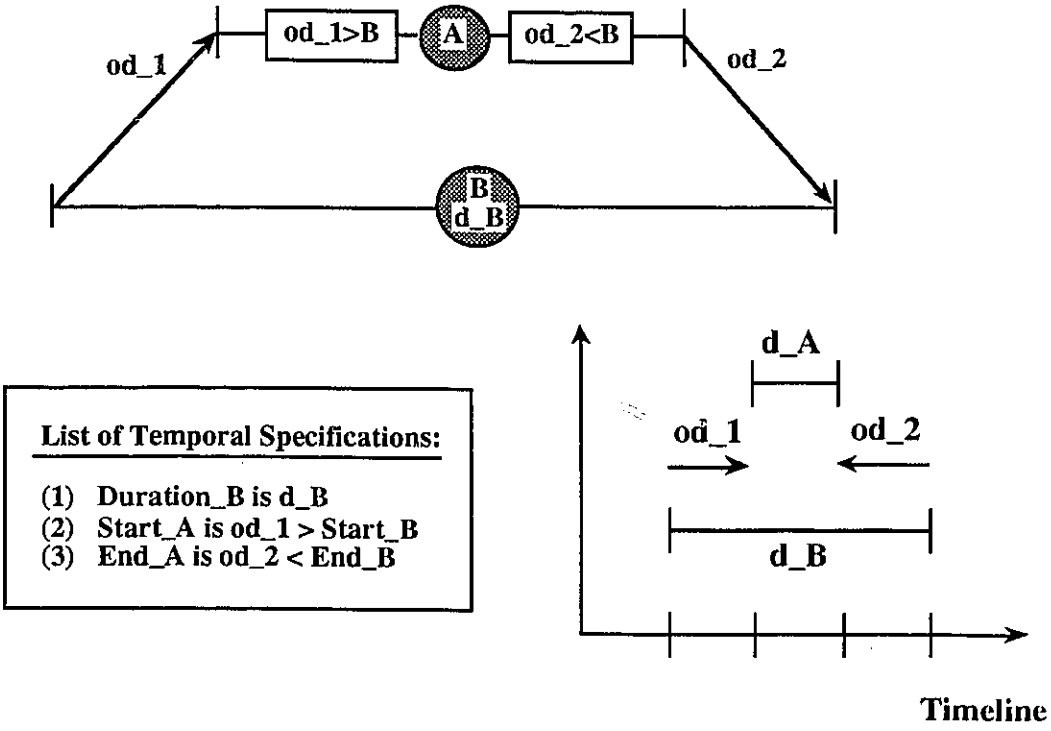


Figure 5.4. *Explicit and derived durations.*

5.3.2 Improper Before Relation Specification

The before relation is the most complicated temporal relation to support from a multimedia document editor's point of view. The difficulty lies with the rendering conductor, which must coordinate the playback of a document after extracting the required temporal information from a scenario. The equals and after relations are simple to realize since the resulting event occurs at the same time or after the synchronizing event. On the other hand, the rendering conductor must predict the time of occurrence of a synchronizing event that is temporally related to a resulting event with the before relation.

This prediction capability is necessary since it would be impossible to render a resulting event before a synchronizing event if the rendering conductor waited for the occurrence of the synchronizing event before it rendered the resulting event. Figure 5.5 will help to clarify this argument. Object B is temporally related to object A with the after relation. Thus, the task of starting to render B (resulting event) at the correct time simply involves starting a timer as soon as the end of A (synchronizing event) occurs. When a duration of od_2 has elapsed, the rendering conductor will start rendering object B. However, this method of starting a timer when the synchronizing event occurs is clearly not possible for object C.

In order to realize a before synchronization specification, the rendering conductor must find a reference point that it can use in order to render the resulting event at the appropriate time. In this example, Start_B might be a suitable reference point. The rendering conductor can start a timer when Start_B occurs. When a duration of **od_1* = (d_B - od_1)** has elapsed, object C will start to be rendered. If od_1* is negative, then Start_B is unsuitable as a reference point, and the multimedia document editor will have to find another reference point (i.e., Start_A or End_A).

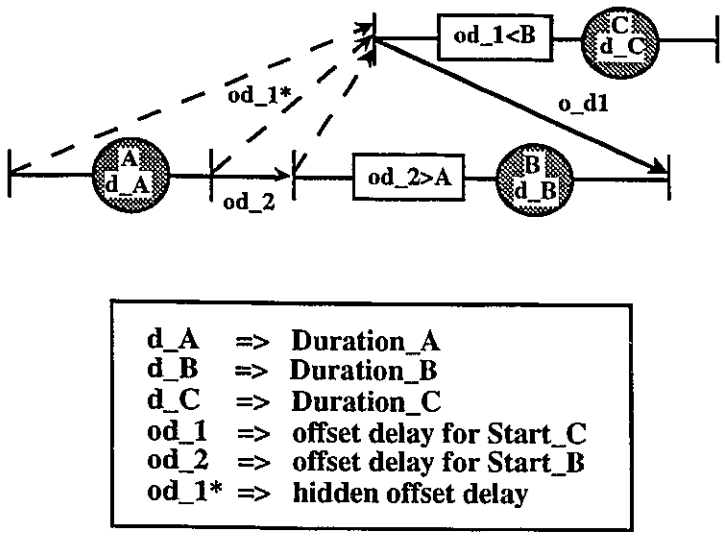


Figure 5.5. *Before temporal relation example.*

5.3.3 Other Illegal Synchronization Specification Types

Redundant or contradictory information:

MEDIADOC prevents contradictory temporal information so that the scenario is unambiguous to the rendering conductor. Redundant information is also prevented, since it is unnecessary and may become contradictory in any temporal specification is modified. By not allowing an author to create a contradictory or redundant temporal specification, the multimedia document editor is helping to ensure that no scenario object is unrenderable and that future corrections will be unnecessary. Examples of these illegal temporal specifications are contained in Figure 5.6. The duration specification for object A is redundant, and it is also contradictory unless **d_A = d_B**.

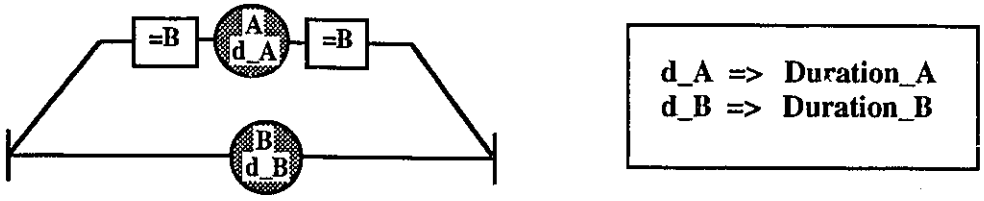


Figure 5.6. *Contradictory and redundant information example.*

Improper scene endpoint specification:

MEDIADOC prevents the use of both the before relation when synchronization to the start of a scene and the after relation in a synchronization specification that involves the end of a scene. Thus, scene autonomy is maintained, since both the start and the end of each scenario object must occur within the scene to which it belongs.

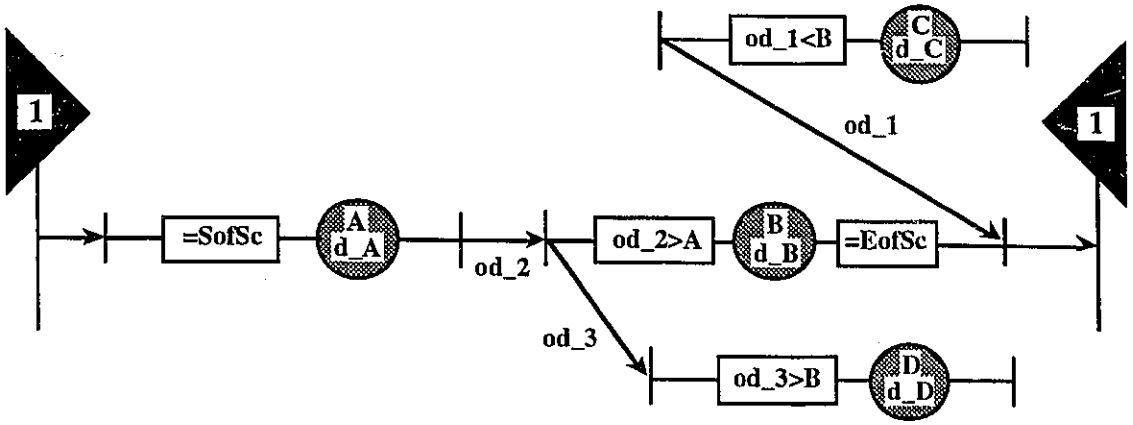
5.4 Final Renderability Verification

After the author has completed a scenario and he/she is ready to save or execute it, the document editing system performs a final verification to determine if the scenario is renderable. This process is relatively simple, since illegal temporal specifications were detected and prevented during the scenario creation stage, and thus the scenario does not contain any invalid synchronization specifications. Final renderability verification involves only determining if the author has specified enough information to execute the scenario.

Final renderability verification is performed on each scene in a scenario, since they are autonomous entities as a result of improper scene endpoint detection and prevention. For a scenario to be renderable, each of its scenes must have three rendering properties: a start of scene synchronization specification, an end of scene specification, and a chain of scenario objects. This chain of scenario objects must join the scenario object(s) in the scene endpoint specifications.

Once these verifications have been performed, the scenario is renderable. Document playback is possible since the rendering conductor will be able to execute the scenario as a sequence of scenes, each containing a properly specified chain of scenario objects. However, another verification is needed to ensure the renderability of dangling scenario objects. Figure 5.7 shows two dangling scenario objects (C and D). When a scenario object owns exactly one synchronization specification and it does not have an endpoint that

is a synchronizing event for another scenario object, it is referred to as a dangling scenario object. These objects require a duration specification to be renderable.



d_A	=>	Duration_A
d_B	=>	Duration_B
d_C	=>	Duration_C
od_1	=>	offset delay for Start_C
od_2	=>	offset delay for Start_B
od_3	=>	offset delay for Start_D

Figure 5.7. *Dangling scenario objects example.*

5.5 Summary

In this chapter, SORT graphs, system checking for illegal temporal specifications, and final renderability verification were discussed. Multimedia document editors can create SORT graphs automatically by converting the temporal information found in scenarios. It was shown that SORT graphs serve several purposes including scenario verification, layout completion, and the playback of selected portions of a document. Scenario verification is achieved simply by examining a SORT graph, which is the timeline

representation of the scenario of a document. An author completes the layout process by identifying which visual objects will be presented simultaneously, and he/she can layout the media objects based on this information. Finally, selected portions of a scenario can be played by placing markers on SORT graphs.

In MEDIADOC, illegal temporal specifications are detected and rejected in order to relieve authors from the burden of debugging scenarios. The illegal specification types are negative durations, improper before relation specifications, redundant or contradictory information, and improper scene endpoint specifications. When a scenario has been completed, the multimedia editing system can perform final renderability verification to ensure that the scenario is renderable. Preventing the creation of illegal temporal specifications greatly simplifies final renderability verification, since it implies that the verification is limited to checking if enough temporal information has been specified by the author.

5.6 Related Work

The Hytime (Hypermedia/Time-Based Document Structuring Language) standard [GOL 91] has been developed to allow the creation of multimedia documents and event schedules. Although multimedia documents can be specified using markup, Hytime provides neither a Global Logical Structure nor a Global Layout Structure for the document creation process. MEDIADOC's Global Logical and Layout Structures facilitate document creation by providing both a framework and a comprehensive set of standardized multimedia objects. Hytime also does not include multimedia authoring tools and support such as graphical scenarios, illegal temporal specification detection, and SORT graphs. The standard does, however, provide extensive support for addressing, hyperlinking, and event scheduling [NEW 91].

Another standard [ISO 93] is currently being developed by the Multimedia Hypermedia Expert Group (MHEG) to facilitate the interchange of multimedia and hypermedia objects (MH objects) between heterogeneous systems. MH objects are intended to be used as low level interchange objects for multimedia/hypermedia applications. The MHEG standard provides a coded representation for these objects by means of the object-oriented concept of classes [COL 93]. However, only limited synchronization specification support is included; MHEG suggests that other standards, such as AVIs (AudioVisual Interactive scriptware), should be used to handle the more complex synchronization requirements of presentation scheduling [KRE 92].

Buchanan and Zellweger [BUC 92a, BUC 92b] and Little and Ghafoor [LIT 90a] have investigated the synchronization requirements for the presentation of multimedia objects. Firefly, the multimedia document system, developed by Buchanan and Zellweger, includes an authoring tool for temporal specification. A scheduler solves duration and temporal synchronization constraints to produce a rendering schedule. The schedule is created by a linear program that minimizes a cost function for altering the duration of each media item. Although Firefly's scheduling algorithm is an interesting way of creating a schedule by shrinking and stretching media item durations, it is not clear how media item duration constraints are determined and how to ensure that the shrinking/stretching effects will not severely degrade the presentation. Documents in Firefly are not organized with a logical structure, and the layout of media items is not discussed.

Little and Ghafoor have developed a storage and retrieval model for multimedia information objects. The storage and retrieval algorithms are based on the presentation synchronization specifications for which interval-based temporal relations [ALL 83] are used to relate two objects that are represented as time intervals. These temporal relations fall into two categories: *sequential* (e.g., before, meets) and *parallel* (e.g., overlaps, during). Little and Ghafoor [LIT 90a] also introduced the OCPN (Object Composition Petri Net)

model to provide a graphical synchronization representation for the temporal relationships between multimedia objects. OCPNs are similar to Petri nets which model synchronization using symbols: *places* (circles), *transitions* (bars), *tokens* (dots), and *directed arcs*.

The main weakness of the OCPN model lies in its inability to model user interaction and branching. Furthermore, it is not possible to temporally relate two instantaneous events, such as the start and end times of media objects, since interval-based temporal relations are used. The paper also does not show how media object layout can be achieved.

Chapter 6
Implementation

6.1 Introduction

In order for authors to create and manipulate multimedia documents, an editor is required. Editors are responsible for providing a set of commands that can be used by authors to, for example, open, create, modify, browse, and save documents. Once a command has been issued, an editor must either execute the command, or notify the user that the command cannot be executed. Many of the user commands and actions performed by an editor in executing those commands are governed by the rules and guidelines contained in document models. Thus, the design of an editor is greatly influenced by the particular document model to which it is associated.

A multimedia authoring system, called MAS, is currently being developed as part of the MEDIABASE project. One of the main components of the MAS is a multimedia document editor called MDE, which is responsible for managing and enabling the creation of multimedia documents. MDE is based on MEDIADOC, the multimedia document architecture that was presented in the previous two chapters. Applications in the MEDIABASE project run on a distributed platform that is mainly composed of SPARC workstations and server, FDDI and Ethernet LANs, digital video and audio facilities, and an object-oriented database management system (OODBMS).

In this chapter, the implementation of MEDIADOC is presented. It is described in terms of MDE, since MEDIADOC is implemented as an integral part of MDE. The chapter begins with a description of MDE and MAS whose architecture is also presented in order to show the overall configuration of our multimedia authoring system. Next, the authoring tools, which comprise the MEDIADOC portion of MDE, and playback module of MDE are described. MEDIADOC was implemented with an object-oriented methodology. Finally, the working version of MDE, which is currently being used in our laboratory, is presented including several "snapshots" of the GUI during different stages of the creation of a sample multimedia document.

6.2 A Multimedia Document Editor

6.2.1 Part of a Multimedia Authoring System

A multimedia authoring system, called MAS (Figure 6.1), is currently being developed as part of the MEDIABASE project whose goal is to develop a distributed multimedia information and communications system. This project includes several applications that have been developed using the generic architecture of our multimedia information system. MEDIABASE's architecture consists of SPARC workstations and a

server that are interconnected by Ethernet and FDDI LANs. Multimedia objects can be created and edited using a production server and media editors (e.g., word processor, graphics editor, audio editor). These objects are transferred to a distributed object-oriented database for storage, and they can be retrieved for playback. The use of a database provides users with important services such as querying, concurrent access, and versioning.

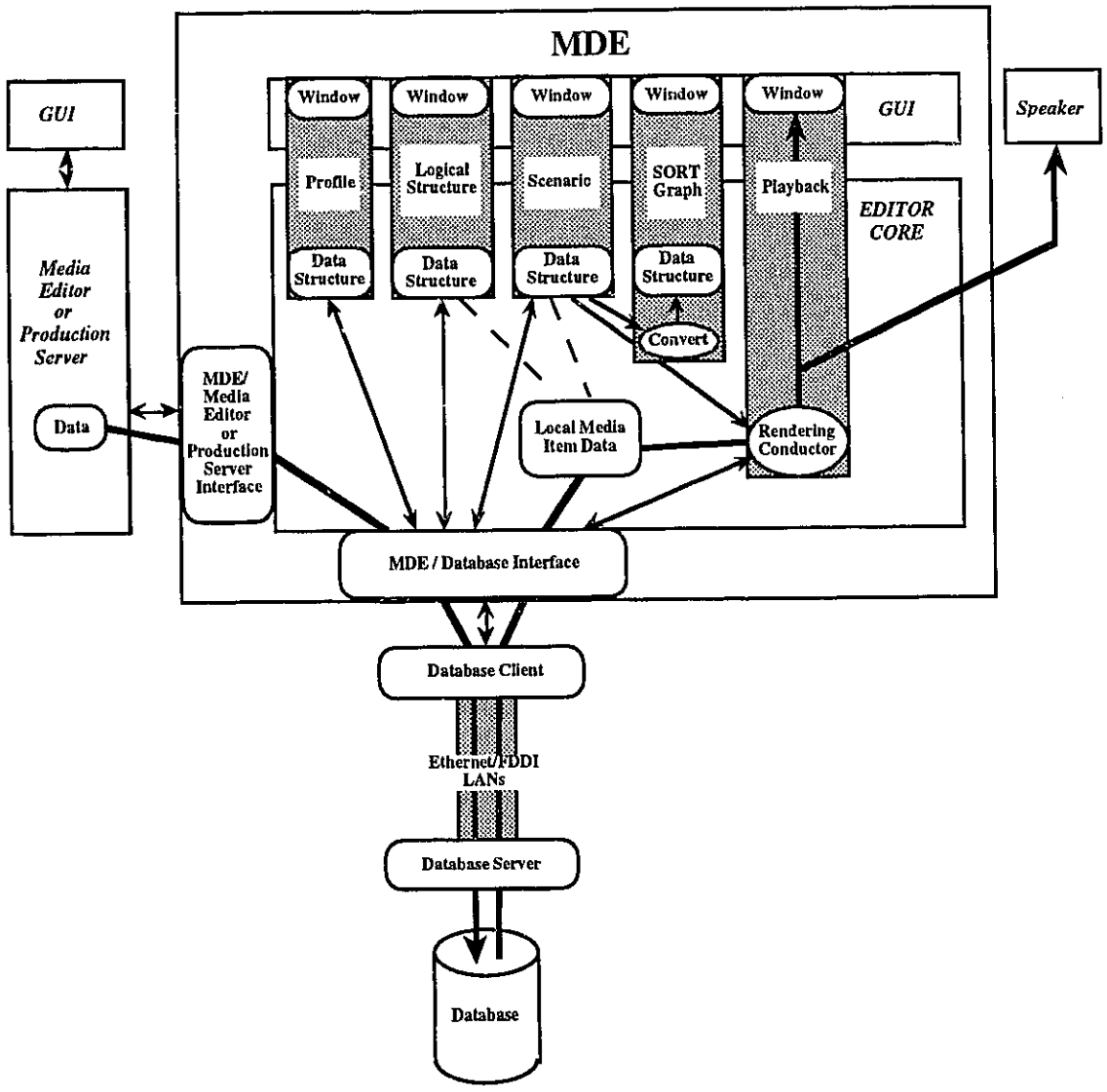


Figure 6.1. *Architecture of MAS (Multimedia Authoring System).*

One of the main components of MAS is MDE, our multimedia document editor. The final version of MDE is intended to make use of all of the system resources, such as database and network, in order to take advantage of all the capabilities offered by our multimedia information and communications system. In its final form, MDE will communicate with several software modules that manage the various resources. These software modules can be divided into two main categories: user applications and system services. User applications include database browsers, file browsers, and media editors. Database browsers are required by authors who wishes to search for either a particular or a group of documents or media items. Systems services, which are transparent to the user, include the transport of information over the network and the specialized physical storage of multimedia data.

However, the first step in the development of MDE is to design the core and GUI of MDE (see Figure 6.1). Since the design of user applications, system services, and the corresponding interfaces are outside the scope of this thesis, therefore they are not discussed in the design of MDE. The following description of MDE's design is limited to the design of the editor core and GUI. Furthermore, only a brief description of the playback component of MDE is described, since presentation schedulers are outside the scope of this thesis. The editor core and GUI software modules combined will use the rules and guidelines in MEDIADOC to enable the creation and manipulation of multimedia documents.

6.2.2 Authoring Tools and Playback Module

The main components of MDE's editor core and GUI are the authoring tools and playback module. Referring back to Figure 6.1, we can see that there are four authoring tools: profile, logical structure, scenario, and SORT graph. These authoring tools allow authors to modify or view a specific part of the document, which is represented in the

editor by four data structures (one for each authoring tool). Each authoring tool consists of a software module in the GUI and another one in the editor core. Authoring tool modules in the GUI include a window that displays some information about the multimedia document currently being edited. Each window also contains a menu of user commands that allows authors to modify either a part of the document (e.g. delete a section) or the display in the window (e.g., update window to show media items in the next section).

Once a user command has been issued, a message is sent to the corresponding authoring tool module in the editor core. If the author has issued a command to modify a document, the authoring tool module in the editor core must first verify that the modification is valid. For example, editors should check that authors do not try to create the same chapter (i.e., chapter number and name) twice. Author confirmation should also be requested for issued commands that will cause significant effects such as the deletion of a chapter including all of its sub-components and media items. Once a command has been verified and confirmation has been received in cases where it has been requested, the command is executed by the appropriate authoring module in the editor core. Executing a command for document modification involves modifying the data structure for which the authoring tool is responsible. Authors are permitted to edit the profile, logical structure, and scenario of a document.

On the other hand, the SORT graph authoring tool allows an author to analyze the presentation schedule of a document in terms of specific start and end times; author editing is not performed on the data structure for the SORT graph. MDE is responsible for creating this data structure by converting a document's scenario into a SORT graph. The method for this conversion, which was presented earlier, has three main steps. The first step is to create a matrix in which each row represents a temporal specification in the scenario. The next step is to produce the row reduced echelon form of the matrix. Finally, the resulting start and end times of each scenario object can be placed in a two-dimensional array.

After an author has created a document with a scenario, he/she can playback the document. The playback module of MDE is responsible for managing the playback of a document. The most important component of this module is the rendering conductor, which controls when media item data is rendered. By extracting temporal information from a document's scenario, the rendering conductor takes the data of media items and sends it to the appropriate rendering location (e.g., speaker, playback window) at the appropriate time in order to present the media items according to the scenario. Layout information stored in each scenario object for visual media is also required to position the rendered data on the monitor.

If the data is not available locally, then it has to be pre-fetched. In MAS, since media data is stored in a distributed database, the rendering conductor must make a request to the MDE/database interface in order to retrieve this data. It is very important that the rendering conductor make each request for media data early enough so that the delays incurred in searching the database(s), packetizing the data for transport across the network, transmitting, propagation over the network, receiving, depacketizing, buffering, and rendering do not cause the data to be presented after its scheduled presentation time [LIT 90b]. As mentioned earlier, the design of a rendering conductor is outside the scope of this thesis. However, it is a component that is required for playback, and therefore the design of a rendering conductor is an important area for further research.

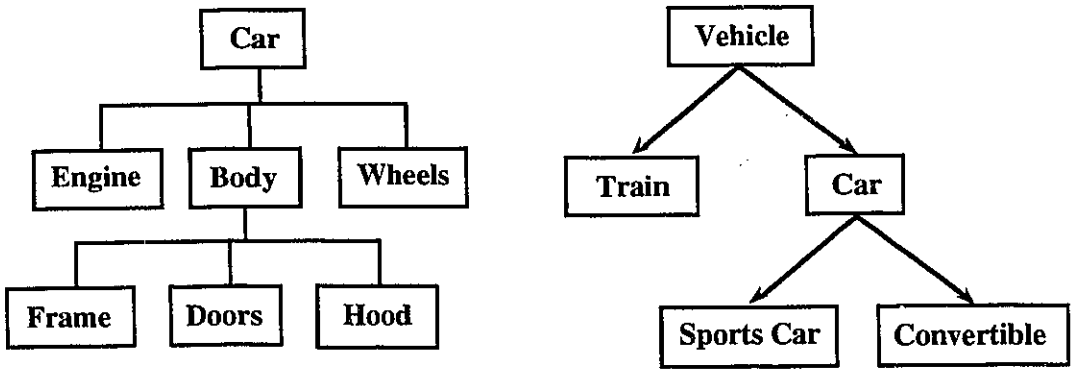
6.3 Implementation of MEDIADOC

6.3.1 Object-Oriented Methodology

Object-oriented (O-O) methods were used for implementing MEDIADOC. The benefits of using these methods, which are well known, have been found very useful for implementing multimedia document models [WOE 86, BAN 87, KLA 90]. In particular,

object-orientation simplifies software development for systems with complex data models [JAC 92]. Furthermore, O-O designs are very extendible due to both the inheritance mechanism and data/behaviour abstraction found in the O-O paradigm. The following is a brief summary of O-O concepts.

Every *object* encapsulates a state and a behaviour. *Encapsulation* means that an object can only be seen by its interface (i.e., the operations that can be performed on the object). The state is the set of values for the attributes of an object, and the behaviour is the set of operations that can be used to examine or modify this state. The value of an attribute can also be an object that has its own state and behaviour. Thus, objects that consist of other objects form an *aggregation hierarchy* (Figure 6.2(a)). All objects that possess the same set of attributes and operations are grouped together as a *class*. An *instance* is an object created from a class, and each instance of a class has the same set of attributes and operations. However, the set of values for those attributes can be and are usually different for different instances.



(a) *Aggregation hierarchy.* (b) *Class hierarchy.*

Figure 6.2. *Hierarchy examples.*

Classes are organized in a *class hierarchy* (Figure 6.2(b)), and each class *inherits* the attributes and operations from its direct and indirect ancestors (*superclasses*) in the hierarchy. Inheritance is a very powerful tool that allows programmers to create new classes by sub-classing. A *sub-class* is a specialization of the classes from which it inherits. In the figure, "Sports Car" and "Convertible" are sub-classes of "Car", which is one of the sub-classes of "Vehicle". Since classes can inherit the attributes and operations of previously defined classes, inheritance promotes *reuse* of code and improves the extendibility of applications [JAC 92]. The class at the top of the class hierarchy is called the *root class*, and classes that have no sub-classes are termed *base classes*.

Figure 6.3 shows the class hierarchy that was developed for MEDIADOC. The base classes of this hierarchy are all objects that are found in MEDIADOC, and they are the classes that will be instantiated (*concrete classes*). In contrast with concrete classes, the *abstract classes* in the hierarchy are represented by the remaining classes that will not be instantiated. These abstract classes serve to maintain the common attributes and operations of their sub-classes including the base classes. For example, the root class "Generic Object" maintains what is common to all classes, such as the user-readable print name, which can simply be referred to as the attribute "name".

However, a simplified hierarchy (Figure 6.4) was used in the implementation of our first prototype. The two most important simplifications are the use of start time and duration instead of temporal relations for temporal specifications, and the absence of mid-layer objects (e.g., Seq, Con, etc.). These classes were implemented in the C++ programming language, so C++ terminology will be used to describe the classes. In C++, attributes are called *data members*, and operations are called *function members*.

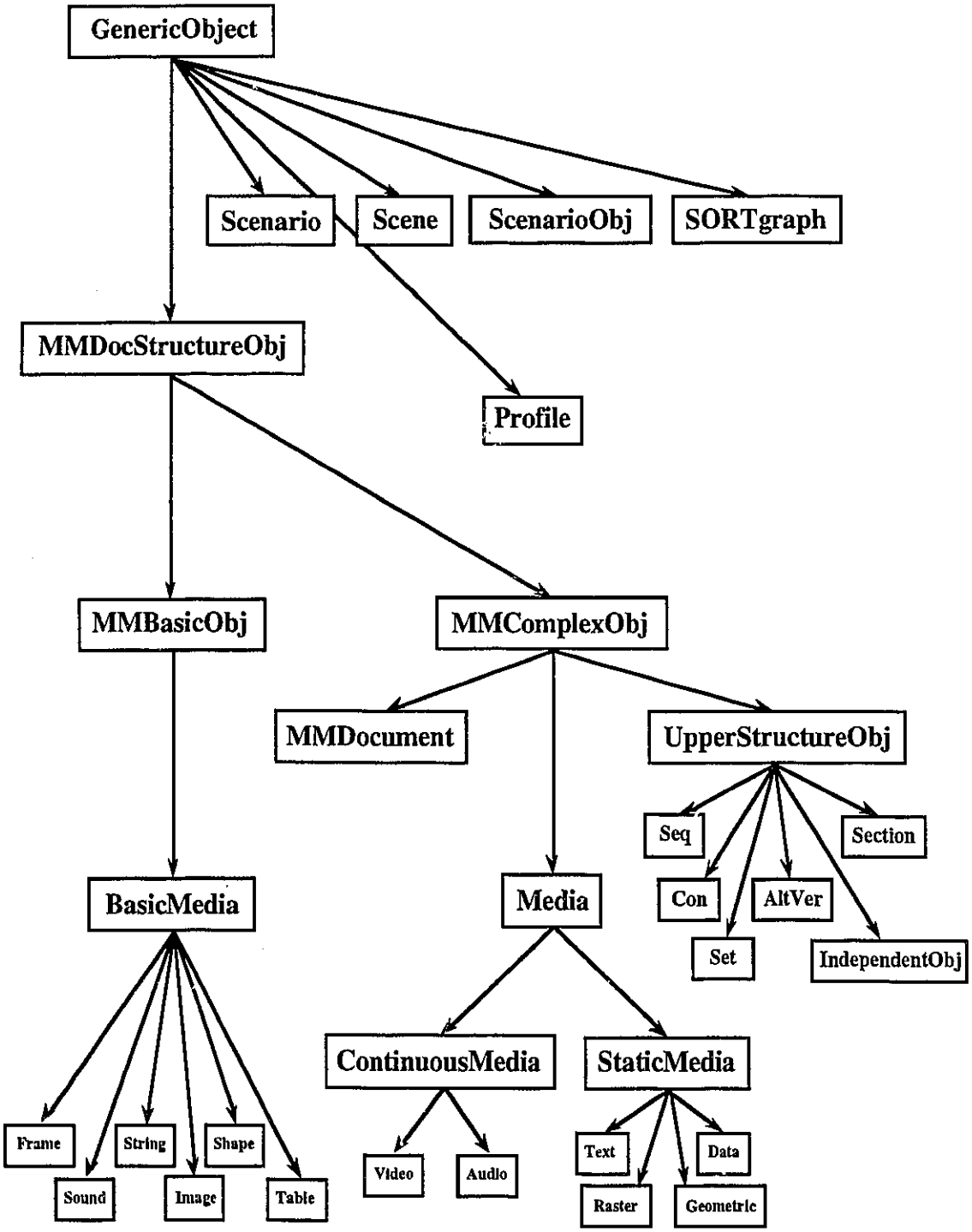


Figure 6.3. *Class hierarchy for MEDIADOC.*

The data members for each class are also shown in Figure 6.4. On the other hand, the function members for the classes are not shown because most of them are simple functions that involve the creation, deletion, or modification of a class' data members. For example, class "PROFILE" includes three member functions for handling its keyword list, namely *add_keyword*, *delete_keyword*, and *modify_keyword*. Each of these member functions make use of *keyword_exists* to verify the existence of the keyword before it applies the requested operation.

Before we continue the description of the classes in this hierarchy, the software components and their interaction should be presented. The three main components are the GUI, MDE, and the set of classes from the class hierarchy. An author interacts with the editor through the GUI. In other words, the menu commands provided by the GUI are issued by the author and given to MDE by the GUI. Then MDE executes the command using the classes and their member functions. Finally, control returns back to the GUI which updates the contents of its windows. For example, if the author has deleted a section, this change should be reflected in the appropriate window. Sometimes menu commands are not passed on to MDE. For example, an author may simply wish to close or open a particular window.

Each document consists of four main parts: a profile, a logical structure, a scenario, and a list of media items. The profile contains the author's first and last names, the date of creation of the document, and a list of keywords. The logical structure is represented by a list of UPPER_STR_OBJs that can either be composed recursively with UPPER_STR_OBJs or contain media items. Scenarios are composed of a list of scenes that contain scenario objects. Each scene is associated with a basic UPPER_STR_OBJ, and each scenario object references a media item in the logical structure. A document's list of media items can be used either to represent independent objects or to act as a container for the media items of documents that have no logical structure.

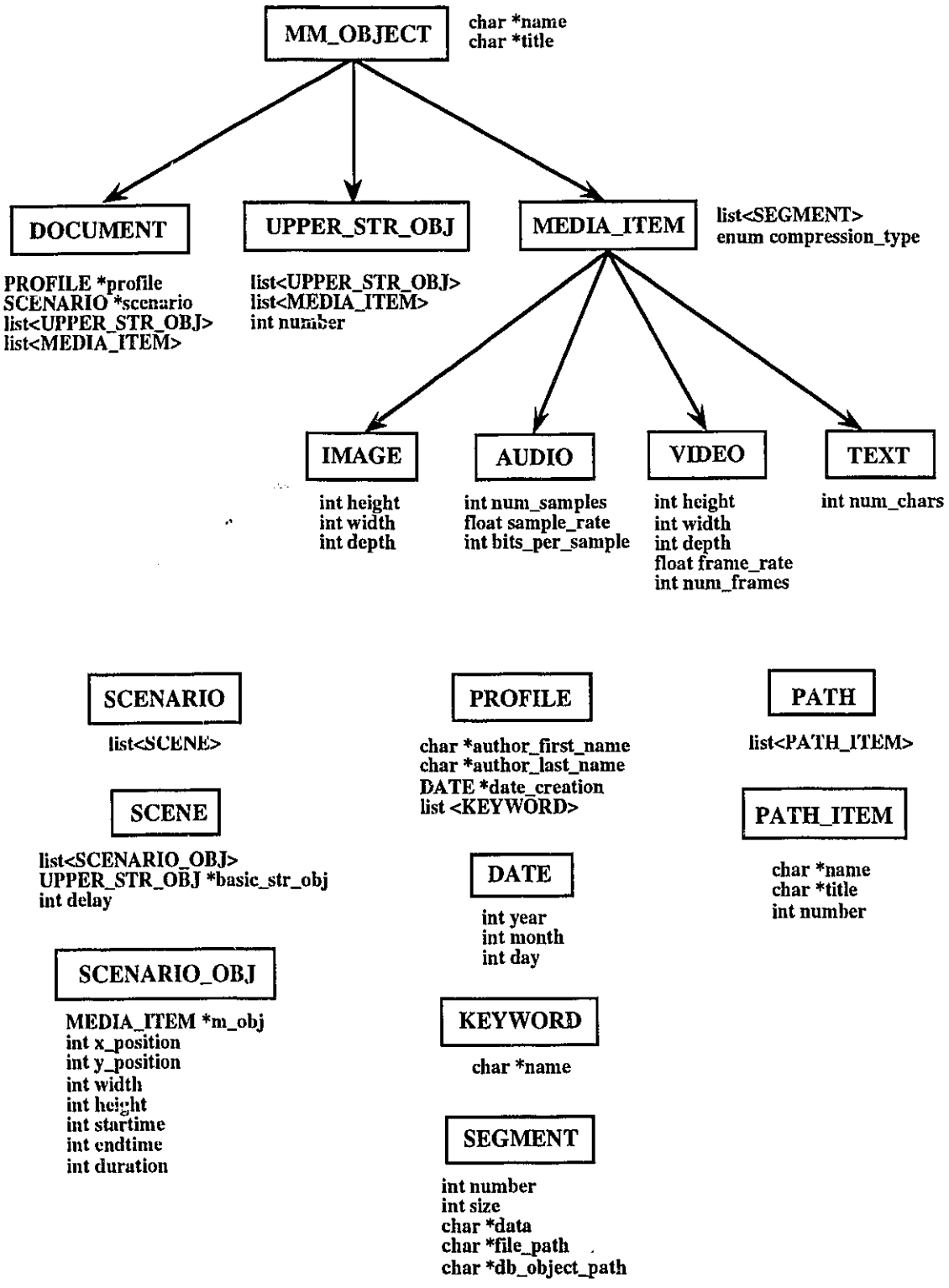


Figure 6.4. *Simplified class hierarchy with data members.*

MDE keeps track of the document that is currently open. When the GUI passes a message (calls a function) to MDE, MDE will pass this message on to the current document instance. If the issued command is to, for example, delete a media item or section, a path must be given by the GUI so that MDE can locate the appropriate object and delete it. The path is composed of a list of path items each of which includes the name, title, and number of an object in the upper layers of the document structure (i.e., UPPER_STR_OBJ). Examples of UPPER_STR_OBJ names are "Chapter", "Section", and "Article". To add an upper-layer object, the name, title, and number of the object must also be included in the messages. Similarly, to add a media item, the name, title, and media type must be passed to MDE and the document instance.

Temporal and layout information is added to scenario objects instead of media items. These information types must be stored independently from media items that serve simply to represent and describe the data of a multimedia object. Scenario objects, which are the presentation versions of media items, describe how the data from these media items will be rendered (i.e., where and when). It is important to separate temporal and layout information from the logical structure of a document to allow both multiple presentation instances of a media items and multiple scenarios for a document.

6.3.2 Creating a Sample Document

The following six figures show how a sample multimedia document was created using the current version of MDE. Each figure is a "snapshot" of the graphical user interface at various stages of document creation. SUIT (Simple User Interface Toolkit) [PAU 91], developed at the University of Virginia, facilitated the creation of the GUI for our multimedia document editor.

The first figure shows the *main menu area* (at the top of the figure) and an *entry board*. The main menu area consists of four buttons representing a document and its three

main components: a profile, a composition, and a scenario. The word "composition" is used instead of the MEDIADOC term "logical structure", since it is more understandable from an author's point of view. When an author clicks on one of the buttons, a menu appears and he/she can select one of the menu items. In this figure, the author has selected "Create Document" causing the entry board to be displayed. This new document is called "Sports Magazine".

After the title, the author's first and last names, and some keywords have been entered, four windows are added to the GUI (Figure 6.6). The *profile window* shows the profile information including the date of creation that is generated automatically by MDE. A *composition window* and two related windows, namely the *media items window* and the *media editors window* are also produced. The composition window shows the current logical structure of the document, and the media items windows displays the icons for the media items that have been created for a particular section. However, we will not see these icons until the last figure, since media items have not been created thus far. Finally, the media editors window includes four icons representing the editors for audio, video, text, and images.

Figure 6.7 shows the entry board for creating new components. Authors can select a pre-defined component type, such as abstract, chapter, and section, or they can enter their own component type. A number and title for the component can also be entered. Section 1 entitled "Hockey" is currently being added to the document. The next figure shows the new section as well as some other sections and articles that have also been created. In this figure, the author is adding a media item (image) entitled "Potvin" to the third article in Section 1. The component to which a media item is to be added is identified by the highlighted line in the composition window; this line is highlighted by MDE after being selected by an author.

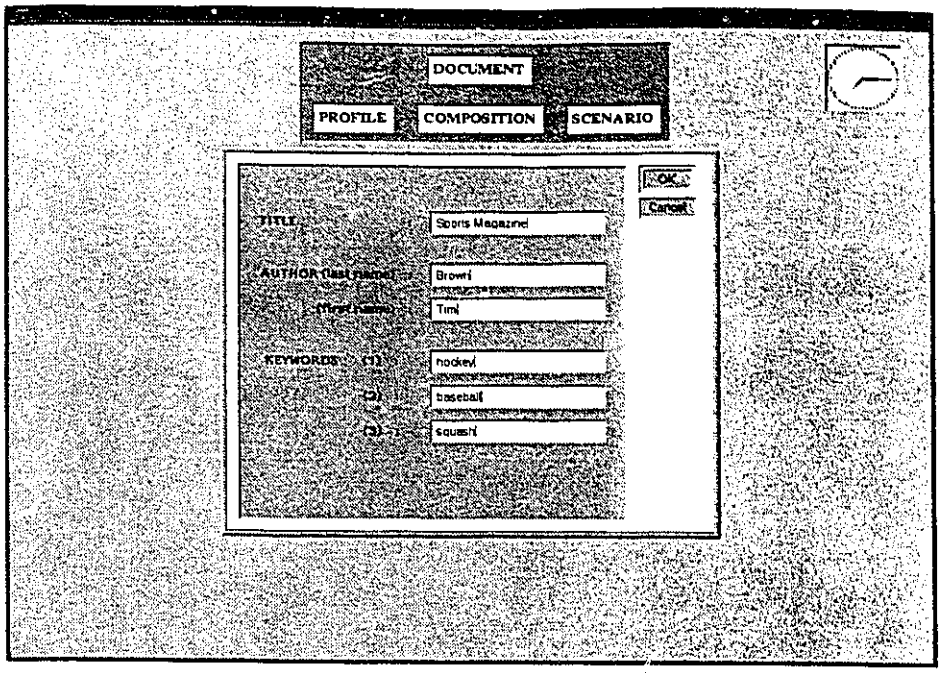


Figure 6.5. *Creating a multimedia document.*

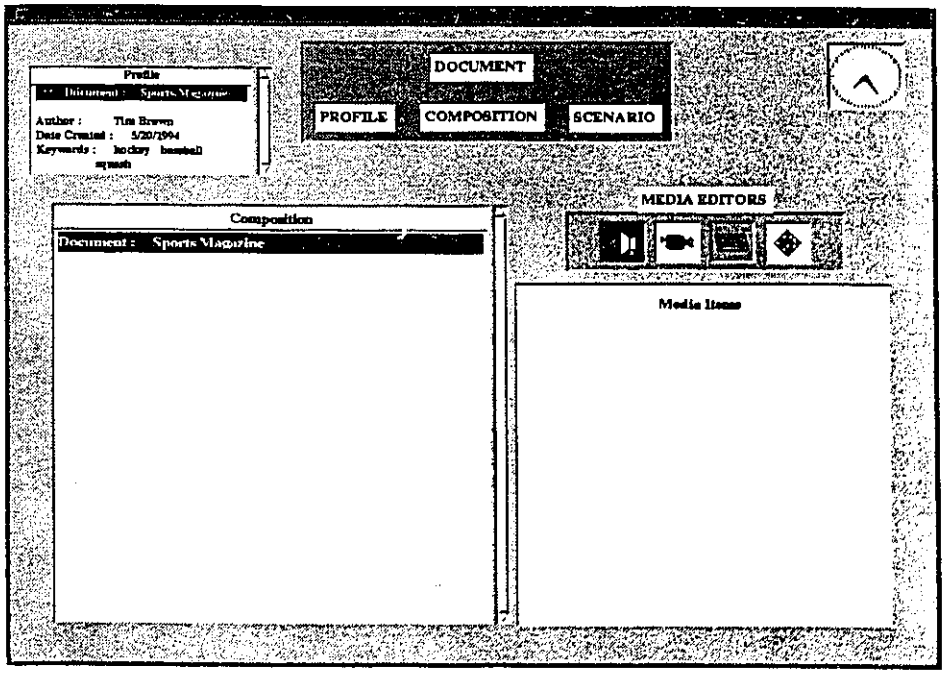


Figure 6.6. *New multimedia document just created.*

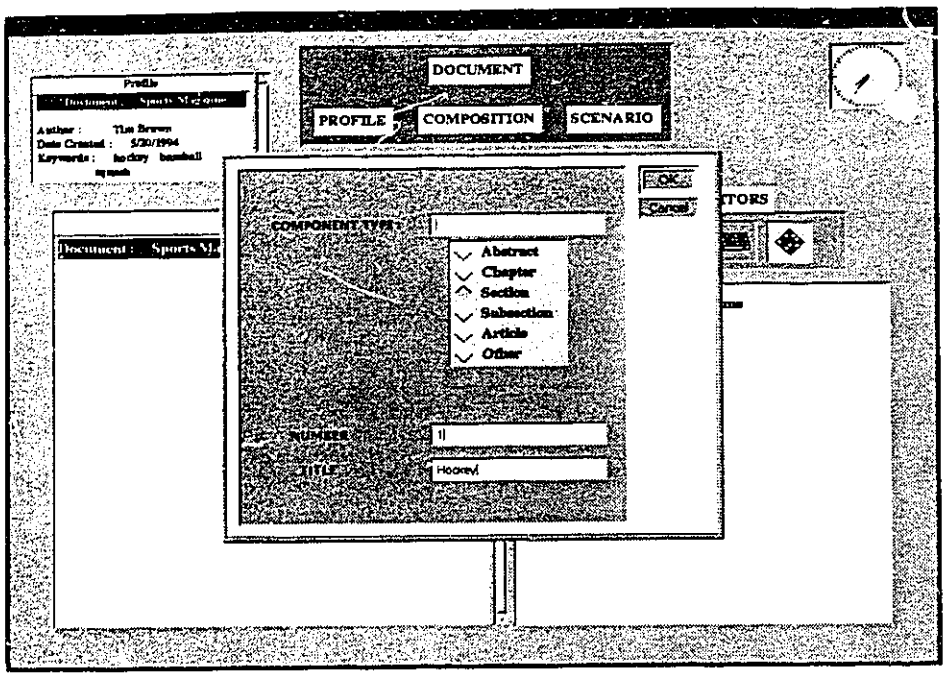


Figure 6.7. *Creating a section.*

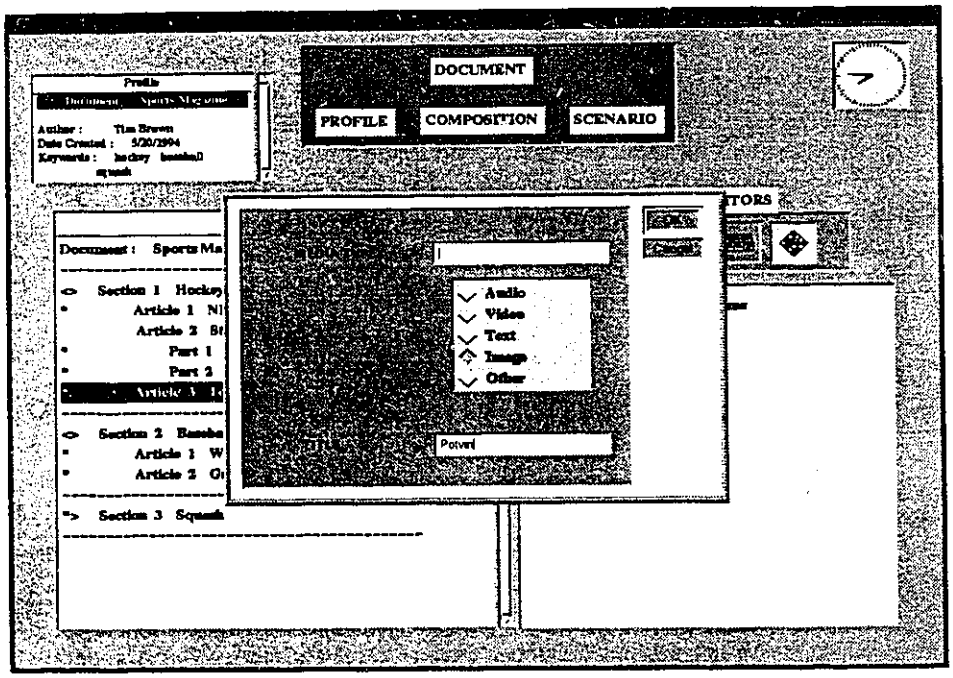


Figure 6.8. *Creating a media item.*

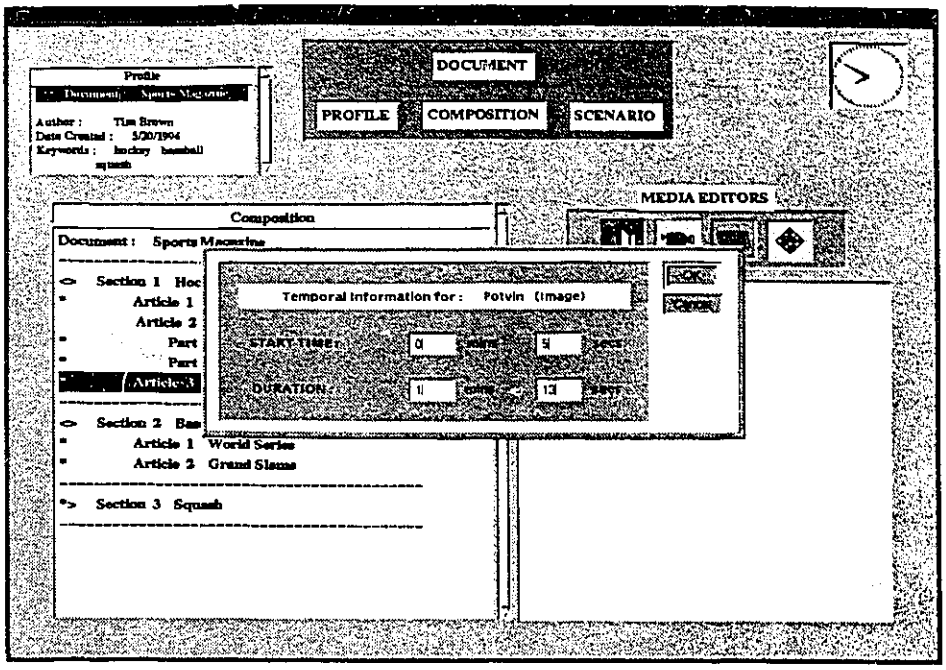


Figure 6.9. *Specifying temporal information for a media item.*

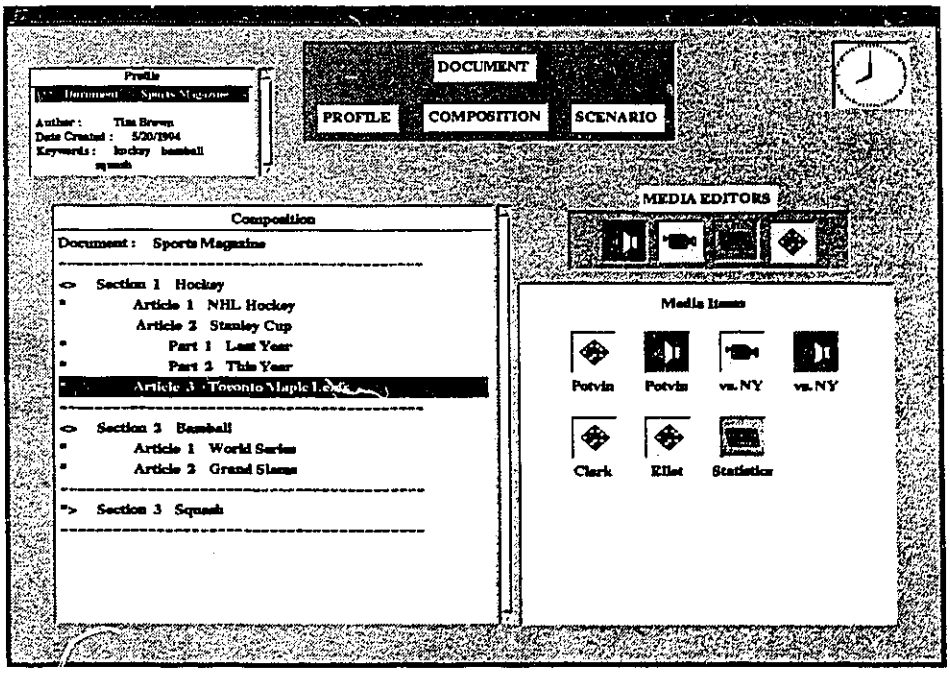


Figure 6.10. *Completed document.*

In Figure 6.9, temporal information is being added to the image of Potvin. The author has specified that this media item should be rendered five seconds after the start of the scene, and it lasts for a duration of one minute and thirteen seconds. The last figure shows the completed document. The media items for the article on the Toronto Maple Leafs are also displayed.

6.4 Summary

In this chapter, the implementation of MEDIADOC was presented. It was shown that this multimedia document architecture is an integral component of MDE, our multimedia document editor. This editor is a part of our multimedia authoring system called MAS, which includes several major components: a document editor, a LAN, SPARC workstations, and an object-oriented database. The architecture of MAS was presented to show the relationships among these major components. The authoring tools and playback module of MDE were also discussed.

Next, the implementation of MEDIADOC was detailed. The MEDIADOC portion of MDE is responsible for managing the creation and manipulation of multimedia documents. In other words, authors interact with this portion of MDE in order to specify information that is required for creating multimedia documents. An object-oriented methodology, which has been widely used for implementing document models, was employed. After a brief description of the main concepts from the object-oriented paradigm, details of the implementation were provided. Two class hierarchies were presented. The second hierarchy, which is a simplified version of the first one, was discussed in detail since it was the hierarchy that was implemented.

Since MEDIADOC is being developed as part of MDE, a graphical user interface also had to be created. Thus, the current version of MDE includes not only the C++ classes for

implementing MEDIADOC, but also a GUI that was created with the support of SUIT (Simple User Interface Toolkit). This prototype allows the creation and manipulation of multimedia documents as seen by the six "snapshots" of the GUI showing the various stages of the creation of a sample document.

The prototype has a number of limitations that result mainly from the fact that a simplified version of our multimedia document architecture was used for implementation. In particular, temporal relations and mid-layer objects (Seq, Con, etc.) would improve the authoring environment. Furthermore, components are shown in the composition window as a hierarchical list. Although not currently offered by MDE, another view of the composition should consist only of graphical objects representing the components of a document. It should be possible to select components with a mouse and modify the structure simply by dragging the graphical objects across the window.

Chapter 7
Conclusions

7.1 Summary

In this thesis, a multimedia document architecture called MEDIADOC was presented. It contains rules and guidelines for the creation and representation of active multimedia documents. We designed this architecture to assist multimedia information systems in supporting applications that involve the prescheduled presentation of multimedia information. The first part of our architecture, which is called the document creation process, outlines how authors can create multimedia documents including presentation schedules using an editor that supports MEDIADOC. It includes steps for document structuring, scenario creation and verification, and layout.

Another significant component of MEDIADOC is its data models. For each of the three main data structures, namely logical structure, layout structure, and scenario, a data model was described. Each data model outlines not only the pieces of information that are included in the corresponding data structure and the relationships among those pieces of information, but also some rules to guide the creation and manipulation of the structure. A graphical representation is also provided for each data model so that they can be visualized. This representation is also useful for display to authors during document editing. However, the graphical representations should be optimized to suit the needs of authors.

The multimedia document architecture's rendering synchronization scheme allows authors to specify temporal information for synchronizing the presentation of multimedia documents. Thus, the architecture is referred to as a time-based architecture. Authors create presentation schedules, called scenarios, with duration and synchronization specifications. Temporal relations are used in synchronization specifications to identify the relationship between the endpoints of media objects.

Since our architecture has been designed with the authoring environment in mind, authoring tools and support have been proposed. A timeline authoring tool (SORT graph) is used by an author to analyze a document's temporal information not by the relations found in the scenario, but rather by the start and end times of the media objects. This timeline view is generated automatically by the multimedia document editor. By analyzing a SORT graph, an author can decide how to lay out the media objects of a document. Authors can also request the playback of a selected portion of the document by placing two temporal markers on a SORT graph.

MEDIADOC was implemented as part of a multimedia authoring system (MAS) consisting of a multimedia document editor (MDE), SPARC workstations, an object-oriented database, and a LAN. The MEDIADOC potion of MDE was implemented in C++

code. Three main software components have been designed; they are the GUI, the classes from the class hierarchy that was developed for MEDIADOC, and the editor core.

The implementation includes three authoring tools for the profile, composition, and scenario, respectively. Each authoring tool is divided into two parts; one part belongs to the GUI, and the other resides in the editor core. The GUI offers commands and displays information about documents to authors. When a command is issued, the GUI passes the message to the editor core. After verifying that the command is legal, the editor executes the command by manipulating one of the document's data structures. Finally, the GUI updates the information in its windows to reflect the changes to the document.

7.2 Suggestions for Further Research

Since MEDIADOC is an architecture for multimedia documents, it is limited to the rules and guidelines for the creation and representation of documents containing multimedia information. Many other issues and areas must be studied before a complete multimedia document editor, which uses this document architecture, can be developed. The most important areas include optimization of the multimedia document editor's GUI, and development of the rendering conductor.

Special GUIs are required for multimedia document editors. These GUI must provide features that support the needs of authors and viewers during playback. The authoring environment for multimedia documents is more complicated than for traditional documents due to the inclusion of temporal specifications. In particular, support is required for improving the authoring of graphical scenarios. Fisheye views [NOI 93] could be used for this purpose by potentially eliminating the scalability problems associated with graphical representations. A fisheye view preserves the size of the graphical objects in the area(s) of interest while proportionally reducing the size of the other objects based on the distance to

the area(s) of interest. This technique can be more effective than zooming or scrolling, since the resulting view simultaneously displays both local detail and global context.

The design of a rendering conductor is essential, since it is needed for document playback according to a scenario. Buchanan and Zellweger have done some research in this area. A good discussion of automatic temporal layout mechanisms is provided in [BUC 93]. *Automatic temporal formatters* use the temporal specifications in a rendering schedule to produce a *temporal layout* consisting of a list of times and events. A runtime support system can assist a formatter by providing it with runtime information such as user interaction.

The rendering conductor should also know about the environment in which it is operating. The capabilities of the components in the environment, such as workstation, network, and database, have an effect on the playback of a document. These capabilities and effects should be taken into account by the rendering conductor as its coordinates the retrieval and display of media objects. The most significant effect is delay. If it is known when an object is it to be presented, the rendering conductor should request the retrieval of the object with sufficient time to compensate for network, database, and rendering delays, in order for the object to be presented by its presentation deadline [LIT 90b, LIT 91, LIL 92]. However, if the media object arrives late, some action should be taken to minimize the degradation of the presentation [BUC 92a, BUC 92b, STE 90].

There are also some ways in which MEDIADOC could be extended or improved. An interesting area for further research is the use of document classes that can assist authors in creating documents. Document classes are useful if many of the created documents exhibit structural similarity, such as company documents of a particular type. In this case, a generic logical structure could be used as a template for creating documents. It would also be useful to investigate if generic structures could be applied to scenarios. However, it is likely that authors will find this notion too complicated.

Algorithms also need to be developed for both the system checking of illegal temporal specifications and the final renderability verification portions of MEDIADOC. The discussion of system checking was limited to identifying several types of illegal temporal specifications. Algorithms are required so that these checks can be performed by the multimedia document editor. For final renderability verification, the algorithm must determine if the scenario is renderable. If it is not renderable, the algorithm should identify the locations in the scenario where more temporal information is required.

Bibliography

[ALL 83] J.F. Allen, "Maintaining Knowledge about Temporal Intervals," *Commun. ACM*, Vol. 26, No. 11, Nov. 1983, pp. 832-843.

[BAN 87] J. Banerjee et al., "Data Model Issues for Object-Oriented Applications," *ACM Trans. on Office Information Systems*, Vol. 5, No. 1, Jan. 1987.

[BOR 91] U. Bormann and C. Bormann, "Standards for Open Document Processing: Current State and Future Developments," *Computer Networks and ISDN Systems*, Vol. 21, 1991, pp. 149-162.

[BRO 89] H. Brown, "Standards for Structured Documents," *The Computer Journal*, Vol. 32, No. 6, Dec. 1989, pp. 505-514.

[BUC 92a] M.C. Buchanan and P.T. Zellweger, "Specifying Temporal Behavior in Hypermedia Documents," *Proc. European Conference on Hypertext '92*, Milan, Dec. 1992.

[BUC 92b] M.C. Buchanan and P.T. Zellweger, "Scheduling Multimedia Documents Using Temporal Constraints," *Proc. Third Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, Nov. 1992.

[BUC 93] M.C. Buchanan and P.T. Zellweger, "Automatic Temporal Layout Mechanisms," *Proc. ACM Multimedia '93*, Anaheim, California, Aug. 1993, pp. 341-350.

[CHA 87] Y. Chauvel and P. Maurice, "The Architecture of an Electronic Document," *Proc. ICC '87*, 1987, pp. 30.1.1-30.1.5.

[CHR 86] S. Christodoulakis et al., "Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System," *ACM Trans. on Office Information Systems*, Vol. 4, No. 4, Oct. 1986, pp. 345-383.

[COL 93] F. Colaitis and A. Leger, "MHEG: An Emerging Standard for the Interchange of Multimedia and Hypermedia Objects," *Proc. ICC '93*, Geneva, May 1993, pp. 542-546.

[DAV 91] N.A. Davies and J.R. Nicol, "Technological Perspective on Multimedia Computing," *Computer Commun.*, Vol. 14, No. 5, June 1991, pp. 260-272.

[FOX 91] E.A. Fox, "Advances in Interactive Digital Multimedia Systems," *IEEE Computer,* Vol. 24 , No. 10, Oct. 1991, pp. 9-21.

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

[GAI 93] B.R. Gaines and M.L.G. Shaw, "Open Architecture Multimedia Documents," *Proc. ACM Multimedia '93*, Anaheim, California, Aug. 1993, pp. 137-146.

[GOL 91] C.F. Goldfarb and S.R. Newcomb, ANSI Project X3.749-D, "Hypermedia/Time-based Document Structuring Language (Hytime)," X3V1.8M/SD-7 (ISO/IEC DIS 10744), April 1991.

[HEH 90] D.B. Hehmann et al., "Transport Services for Multimedia Applications on Broadband Networks," *Computer Commun.*, Vol. 13, No. 4, May 1990, pp. 197-203.

[HOE 92] P. Hoepner, "Synchronizing the Presentation of Multimedia Objects," *Computer Commun.*, Vol. 15, No. 9, Nov. 1992, pp. 557-564.

[HUN 89] R. Hunter et al., "ODA: a Document Architecture for Open Systems," *Computer Commun.*, Vol. 12, No. 2, April 1989, pp. 69-79.

[ISO 86] ISO, Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML), International Standard 8879, 1986.

[ISO 88] ISO, Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format (ODIF), Parts 1-8, International Standard 8613, March 1988.
Part 1 - Introduction and General Principles.
Part 2 - Document Structures.
Part 3 - (not used).
Part 4 - Document Profile
Part 5 - Office Document Interchange Format (ODIF).
Part 6 - Character Content Architecture.
Part 7 - Raster Graphics Content Architecture.
Part 8 - Geometric Graphics Content Architecture.

[ISO 93] ISO, Multimedia and Hypermedia information coding Expert Group (MHEG), ISO/IEC JTC1/SC29/WG12, MHEG Working Draft "WD.1.0", Version 1.0, Feb. 1993.

[JAC 92] I. Jacobson et al., *Object-Oriented Software Engineering*, Addison-Wesley, Reading, MA, 1992.

[JOL 89] V. Joloboff, "Document Representation: Concepts and Standards," In *Structured Documents*, J. André et al. (Editors), Cambridge University Press, Cambridge, 1989, pp. 75-105.

[KAR 93] A. Karmouch, "A Multimedia Information and Communications System: MEDIABASE," *Proc. Multimedia Commun.'93*, Banff, Alberta, April 1993, pp. 234-243.

[KLA 90] W. Klas et al., "Using an Object-Oriented Approach to Model Multimedia Data," *Computer Commun.*, Vol. 13, No. 4, May 1990, pp. 204-216.

[KRE 92] F. Kretz and F. Colaitis, "Standardizing Hypermedia Information Objects," *IEEE Commun. Magazine*, May 1992, pp. 60-70.

[LEG 91] D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Commun. ACM*, Vol. 34, No. 4, April 1991, pp. 46-58.

[LIL 92] L. Li et al., "Synchronization in Real Time Multimedia Data Delivery", *Proc. ICC'92*, Chicago, June 1992, pp. 587-591.

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

[LIT 90b] T.D.C. Little and A. Ghafoor, "Network Considerations for Distributed Multimedia Object Composition and Communication," *IEEE Network Magazine*, Vol. 4, No. 6, Nov. 1990, pp. 32-49.

[LIT 91] T.D.C. Little and A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services," *IEEE JSAC*, Vol. 9, No. 9, Dec. 1991, pp. 1368-1381.

[MOO 90] D.J. Moore, "Multimedia Presentation Development Using the Audio Visual Connection, " *IBM Systems Journal*, Vol. 29, No. 4, 1990, pp. 494-508.

[MOU 91] M.M. Mourad, "Object Representation for Multimedia Document Interchange," *Proc. ICC '91*, 1991, pp. 532-539.

[NAF 90] N. Naffah, "Multimedia Applications," *Computer Commun.*, Vol. 13, No. 4, May 1990, pp. 243-249.

[NAG 92] A. Nagasaka and Y. Tanaka, "Automatic Video Indexing and Full-Video Search for Object Appearances," In *Visual Database Systems II*, E. Knuth and L. Wegner (Editors), Elsevier Science Publishers B.V., Brussels-Luxembourg, 1992, pp. 113-127.

[NEW 91] S.R. Newcomb et al., "Hytime: Hypermedia/Time-based Document Structuring Language," *Commun. ACM*, Vol. 34, No. 11, Nov. 1991, pp. 67-82.

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

[NOI 93] E.G. Noik, "Exploring Large Hyperdocuments: Fisheye Views of Nested Networks," *Proc. CASCON '93*, Toronto, Oct. 1993, pp. 661-676.

[OGA 90] R. Ogawa et al., "Scenario-based Hypermedia: A Model and a System," In *Hypertext*, A. Rizk et al. (Editors), Cambridge University Press, Cambridge, 1990, pp. 38-51.

[PAU 91] R. Pausch et al., "SUIT: The Pascal of User Interface Toolkits," *Proc. ACM Symposium on User Interface Software Technology*, Hilton Head, South Carolina, Nov. 1991, pp. 117-125.

[RAY 92a] D.R. Raymond, "Evolutions in Typesetting Systems," *Proc. CASCON'92*, Toronto, Nov. 1992, pp. 19-28.

[RAY 92b] D.R. Raymond et al., "Markup Reconsidered," *Proc. First Intl. Workshop on Principles of Document Processing*, Oct. 21-23, 1992.

[SCH 88] G. Schulze, "Office Document Architecture and its Use in an Office Environment," In *Information Technology for Organizational Systems*, H.J. Bullinger et al. (Editors), Elsevier Science Publishers B.V., Brussels-Luxembourg, 1988, pp. 1025-1030.

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

[WAL 91] G.K. Wallace, "The JPEG Still Picture Compression Standard," *Commun. ACM*, Vol. 34, No. 4, April 1991, pp. 30-44.

[WIL 94] N. Williams and G.S. Blair, "Distributed Multimedia Applications: A Review," *Computer Commun.*, Vol. 17, No. 2, Feb. 1994, pp. 119-132.

[WOE 86] D. Woelk et al., "An Object-Oriented Approach to Multimedia Databases," *Proc. ACM-SIGMOD 1986*, Washington, D.C., May 1986, pp. 311-325.

Publications

1. J. Emery and A. Karmouch, "A Multimedia Document Architecture and Rendering Synchronization Scheme," *Proc. Second Intl. Conf. on Broadband Islands*, Athens, June 1993, pp. 59-67.

2. J. Emery and A. Karmouch, "A Multimedia Document Model for Continuous Media," *Proc. CCECE '93*, Vancouver, Sept. 1993, pp. 640-643.

3. J. Emery and A. Karmouch, "A Document Model for Multimedia Information," *Proc. CASCON '93*, Toronto, Oct. 1993, pp. 715-728.

4. J. Emery and A. Karmouch, "A Time-Based Multimedia Document Architecture," (to be submitted to *ACM Multimedia*).