National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

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

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

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

AVIS

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

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

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

Canada

Design and Implementation of A Persistent Multimedia Object-oriented Storage System

by

Jie Li

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

Master of Applied Science

in

Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

June 1995

ISBN 0-612-04947-7

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Abstract

Multimedia information requires novel database architecture and models for its efficient storage, manipulation, retrieval, and playback. Multimedia database system must provide facilities to model complex objects, manage the temporal relationships among different media and guarantee the synchronization as well as the continuity requirements during retrieval.

In this thesis, we investigate the object-oriented database concepts and their ability to support multimedia applications and more specifically the support of real time audio/video media. A number of multimedia characteristics and requirements are analyzed and identified. Motivated by the challenge to meet the requirements, a Persistent Multimedia Object-oriented Storage System called MEDIASTORE is designed and implemented using an object-oriented database model for the management of multimedia document. A multimedia document architecture is used by MEDIASTORE to describe and model complex objects such as audio, video, image, and text. The temporal and spatial relationships between objects are also described in the document. A synchronized retrieval algorithm for playback of multimedia document is presented. The algorithm pays great attention to various unique synchronization requirements in retrieval and playback of multimedia documents. As part of an advanced multimedia OODBMS, MEDIASTORE is rich in features for the storage and manipulation of document. Amongst the features is its Graphical User Interface (GUI). The design and implementation of the GUI for MEDIASTORE is presented.

Given the ever evolving nature of multimedia requirements and the time limit on this work, MEDIASTORE is far from perfect. Future trends and work are also presented.

Acknowledgments

First of all, I wish to express my gratitude to Dr. A. Karmouch, my supervisor, for his skilled advice, his encouragement and his tireless support throughout this work.

I am grateful to the Telecommunications Research Institute of Ontario for its financial support.

I would also like to thank all the members of the MEDIABASE project: Ruihong Wang, James Emery, Omar Megzari, Philippe Galvez, Habib Khallfallah, Bala Brahmannandem Madaparthi, Glen Warnock, Nael Hirzalla, James Rody, Yao Kang, Lian Li and Li Li.

I am truly grateful to my family for their constant support, without which this work would not have been possible.

Contents

Figures

Chapter 1

Introduction

A number of significant advances are now taking place that are facilitating the development of multimedia applications. These advances include developments in high bandwidth networks and protocols facilitating real time transfer of digital video and audio such as broadband ISDN and ATM; improvements in storage media such as high capacity magnetic disks and optical disks; better video and audio processors on workstations; and development of real time compression and decompression technology for digital video and audio. The compressed video has data rates comparable to bus and disk bandwidths and therefore opens the possibility of video recording and playback from conventional secondary storage device [GIB93].

Multimedia technologies aim at providing the ability for people to communicate, store, retrieve and exchange multimedia information in a "natural" way. This will be achieved by integrating multimedia information processing and communications [KAR93a].

The multimedia information research laboratory (MIRLab) in the department of electrical engineering at the University of Ottawa, is exploring a vision of multimedia research that combines many aspects of telecommunications and information processing. The vision of multimedia information system 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 particular focus on document architecture, database models, high-level communication and synchronization protocols, and real-time physical storage of multimedia data [KAR93a][KAR93b][EME93a] [EME93b][EME93c][WAN93][LIL92][LIL93a][LIL93b][LIL93c][KAR90]. Objects manipulated in MEDIABASE are composed of text, graphics, still image, motion video, audio, and voice [KAR93c].

One of the biggest challenges in the multimedia information and communications system is the need to store, manipulate and retrieve multimedia information in a database system so that the multimedia applications can benefit from database technology in managing the data in the multimedia information system [RAM93].

Multimedia information is composed of complex objects such as audio, video, graphics, image, voice, and text. Each type of information requires appropriate tools for acquisition, processing, transfer, storage and retrieval. Semantic and temporal relationships may exist between these types of information that require a uniform, homogeneous representation and storage management within the database. The characteristics of multimedia information require novel database architectures and models for efficient storage, manipulation, retrieval, and playback. Multimedia database system must provide facilities to model complex objects, manage the temporal relationships among different media and guarantee the synchronization as well as the continuity requirements during retrieval.

To represent and manage data of various types and origins uniformly, an object-oriented database system is considered as a possible candidate since object concept is suitable for this purpose [MAS87]. Therefore, an exploration of using object-oriented approach for the design and development of the multimedia database system is worthwhile.

Although the existing object-oriented database system is more suitable for multimedia applications than the traditional database system, it does not support real time audio/video media. In particular, how to represent and manage temporal relationships among different media, and how to store and retrieve real time audio and video in a database are still unsolved issues.

The work reported in this thesis, as one of the main components of the MEDIABASE project, is the first step towards the design and development of the multimedia object-oriented logical storage system, which will be used to manage the multimedia documents. This project is called MEDIASTORE.

In this thesis, we investigate the object-oriented database concepts and their ability to support multimedia applications and more specifically the support of real time audio/video media. We identify a number of multimedia characteristics and requirements. Based on the requirements, an object-oriented database schema for the management of multimedia document is proposed. A multimedia document architecture is used to describe and model complex objects such as audio, video, image, and text. The temporal and spatial relationships between objects are also described in the document. A synchronized retrieval algorithm for playback of multimedia document is presented. A number of database features for the storage and manipulation of document are introduced. The implementation of the MEDIASTORE database schema uses ObjectStore [OBJ93a][OBJ93b] which is a

commercial object-oriented database management system. The database schema will also be used to evaluate the main features of ObjectStore and its ability to manage multimedia information such as a multimedia document.

The thesis is organized as follows. Chapter 2 is devoted to traditional database concepts and the relational technology. This includes the analysis of the characteristics and the shortcomings of the relational database model for multimedia applications. The discovery of the shortcomings gives rise to the need of investigating more suitable database technology: the object-oriented database system. Chapter 3 explains the concepts of object-oriented approach and discusses the features of object-oriented databases.

Before the design of our database model, we need to review the multimedia characteristics, identify the requirements that our system should fulfill, and study the existing database systems in this research area. This is done in Chapter 4. We then describe, in Chapter 5, MEDIASTORE which is a multimedia object-oriented logical storage system. A multimedia object-oriented database schema for management of continuous media in a document is proposed. A synchronized document retrieval algorithm for playback is presented. The features for the storage, manipulations of the document are described.

Finally, we summarize the present work and propose a few ideas for further research in Chapter 6.

Chapter 2

Traditional Database Systems

Our Persistent Multimedia Object-oriented Database Storage System is first of all a database system. It must provide database features. In this chapter, first, we will examine the reasons why the databases are needed. Then, we will briefly analyze the characteristics and the shortcomings of the relational database model for supporting multimedia applications. The discovery of the shortcomings give rise to the need of investigating the new database technology: object-oriented database systems.

2.1 Why Databases?

Before database management systems were introduced, file systems were used in complex applications. In a file system, each application program has its own set of files, each of which has its own format and structure. Although a file system can be efficient for small, single-user applications, it shows significant deficiencies that make it impractical for large, multi-user applications in which,

- Data are stored redundantly, making them difficult to keep consistent.

- Programs depend on the structures of the stored data, making these structures difficult to change.
- Sharing work among a team of people is difficult.

The development of databases and database management systems provides solutions to all these challenges facing the file systems.

A database is a collection of related data. A typical database represents some aspects of the real world and is used for specific purpose by one or more groups of users. A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. Hence, the DBMS is a general-purpose software system that facilitates the processes of defining, constructing, and manipulating databases for various applications. Defining a database involves specifying the types of data to be stored in the database, along with a detailed description of each type of data. Constructing the database is the process of storing the data itself on some storage medium which is controlled by the DBMS. Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the real world, and generating reports from the data. The database and software (DBMS) together form a database system [ELM89].

A database management system provides long-term, reliable storage of data which outlives the process that created the data and helps manage large amount of data that exceed the memory available to the application software. This characteristic is called *persistence*. For example, an airline reservation system provides facilities to book a seat on an airplane. It is necessary for the system to have the information stored permanently so it can be updated and reviewed by others [OBJ93b].

In addition to persistence, a database management system provides features that facilitate (1) the description of data, (2) the maintenance of correctness and integrity of the data, (3) efficient access to the data, and (4) the correct executions of query and transaction executions in spite of concurrency and failures. Specifically [RAM93],

- Database schemas help avoid redundancy of data as well as of its description;
- Data management support, such as indexing, assists in efficient access to the data;
- Transaction support, where transactions have ACID (atomicity, consistency, isolation, and durability) properties, ensures correctness of concurrent transaction executions and ensure data integrity maintenance even in the presence of failures.

2.2 The Relational Model

During the past three decades, the database technology for information systems has evolved from hierarchical, network database systems to relational database systems. The new generation of database technology is currently under development.

The relational model of data was introduced by Codd [COD70][COD72][COD74]. A set of commercial systems has been developed since late 70s and early 80s, such as Oracle, SQL/DS [ANS86] and DB2, and INGRES [STO86]. The relational model represents the data in a database as a collection of relations. Informally, each relation resembles a table. Each row in the table represents a collection of related data values. These values can be interpreted as a fact describing an entity or a relationship instance. The table name and column names are used to help in interpreting the meaning of the values in each row of the table. All values in a column typically are of the same data type. In relational database terminology, a row is called a tuple, a column name is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each

column is called a domain. A domain is a set of atomic values which is indivisible as far as the relational model is concerned [RLM89]. With the atomicity constraint, the value of any attribute may be an integer, float, string of characters, etc., but not a relation for instance.

Comparing with other database models such as hierarchical model and network model, relational model has some distinguishing characteristics [ELM89]:

- the simplest and the most uniform data structures,
- the well defined relational algebra, which is a collection of operations such as: SELECT, PROJECT, JOIN, and UNION, and
- the declarative query languages.

Relational database technology is characterized by the notion of a declarative query. The introduction of declarative queries in relational databases relieves application programmers of the tedious chore of programming navigational retrieval of records from the database.

However, relational database technology, just as each of the previous generation database technologies, was developed for the conventional business data-processing applications, such as inventory control, payroll, accounts, and so on. Attempts to make use of relational database technology in a wide variety of other types of applications have quickly exposed several serious shortcomings of the relational and past-generation database technology [KIM90]. These applications include multimedia systems which deal with video, audio, images, voice, and multimedia documents.

Several of the well-known shortcomings of the relational database technology can be summarized as follows [KIM90]:

- The relational model is too simple for modeling complex nested entities, such as multimedia documents; relational database systems do not provide mechanisms to represent and manage such entities.
- Relational database systems support only a limited set of atomic data types, such as integer, string, etc.; they do not support general data types found in programming languages. In particular, they do not even allow the storage and retrieval of large data such as images, and audio.
- The relational model does not include a number of frequently useful semantic concepts, such as generalization and aggregation relationship which is essential in a multimedia document. Therefore, application programmers must explicitly represent and manage such relationships in their programs, since the database system does not provide the necessary functions.
- The impedance-mismatch between the algorithmic programming language (such as C, and C++) and the database language (such as SQL, DL/1 or CODASYL DML) used in application programs. Database languages are very different from programming languages, in both data model and data structures.
- The model of transactions supported in relational database systems is inappropriate for long-duration transaction necessary in multimedia environments.

The discovery of the shortcomings of relational database technology for multimedia applications provides the motivation for us to investigate the new database technology: object-oriented database system.

Chapter 3

The Object-oriented Approach and Database Systems

In our work, we have developed a multimedia database system MEDIASTORE on top of the ObjectStore object-oriented database management system. It uses the capabilities provided by object-oriented technology for its own purposes, and benefit from the object-oriented approach. Although the object-oriented database system is more suitable for the modeling of multimedia information than the traditional database systems specially the relational database systems, it does not support real time audio and video media. Therefore, additional features for the multimedia database system to support multimedia applications are needed. To use and extend ObjectStore object-oriented database management system, the object-oriented mechanisms and characteristics have to be studied. In the first section of this chapter, the main benefits of object-oriented approach are described. Then, the basic concepts of object-oriented programming are introduced. Finally, the features of object-oriented databases are discussed.

3.1 Why Object Orientation?

Multimedia information includes a variety of complex data types such as audio, video, graphics, image, voice, and text. Each type of information requires appropriate tools for acquisition, processing, transfer, storage and retrieval. The traditional database management systems, such as relational database management systems, provide the ability to model simple alphanumeric information organized as records. Although these DBMSs could handle traditional business applications, they had some significant deficiencies (discussed in Chapter 2) that make them impractical for multimedia applications. There are three benefits for adapting object-oriented approach for multimedia database organization and management [MAS87]:

- Regarding everything as an object can provide a unified view in the data structure of a diverse set of the real world representations, i.e., a set of databases in different media and even those in a single media which are constructed corresponding to different acquisition methods.

- Accordingly, we can use a single entity called multimedia document to describe, organize and structure multimedia objects so that we can store, retrieve and manipulate multimedia document as a whole regardless of having various data types in it.

- Having an object-oriented approach in multimedia database systems can provide a single high level user interface which can access objects stored in different media in a uniform manner. This is possible because different methods can be specified in different classes using the same method name.

3.2 Object-Oriented Paradigm

A paradigm is a way of thinking, an approach to solving a problem. The traditional procedural programming such as structured programming is based on functions. It improves programming efficiency by restricting program code interactions [STE93]. The classic structured programming languages are C and Pascal. Unlike the traditional approaches, object-oriented programming focuses on the data to be manipulated rather than on the functions that do the manipulating [WIE88].

Object-oriented programming is an approach in which software organization corresponds closely to the real system [BOW94]. Supplanting conventional procedure-calls with the more general mechanism of sending messages, it greatly enhances the flexibility and reusability of software components [STE93]. The object-oriented approach is characterized by the features of data abstraction, encapsulation, inheritance, and polymorphism. The following sections will describe all these features in details.

3.2.1 Data Abstraction

Abstraction is the principle of ignoring those aspects of a subject that are not relevant to the current purpose in order to concentrate more fully on those that are [STE93]. Data abstraction is the concept of encompassing the data (attributes) together with the functions (methods) on the data. It provides the ability to build a data structure to define an object and then use it within a program without paying attention to its internal detail, much as a person might manipulate a real-world object (e.g., a multimedia document) without seeing its internals. It also simplifies the programmer's task by reducing the number of details to be dealt with at any given time. The principle of data abstraction separates the user of an object from its implementor. The implementation of an object is not visible to the user of the object. The users operate on the object by means of the functions which are made visible by the implementor. As a result of data abstraction, the implementor can

change the implementation of an encapsulated object without affecting the applications using it.

3.2.2 Encapsulation

Encapsulation is the mechanism to realize the concept of data abstraction [RAM89]. In other words, data plus the functions on the data are enclosed in an object. Encapsulation provides the enforcement necessary to make sure that only functions immediately associated with the data structure are allowed to access its internal details. Without encapsulation, data abstraction is merely a convention, a rule of convenience, that a function might break at any time. With encapsulation, functions cannot reach into the data structure any more than they are allowed.

In general, an object is said to have a public interface and a private part. The public interface determines the interaction of one object with other objects and the private part consists the hidden information. An object can read and modify all of its own attributes. Hiding information restricts the freedom of other objects to retrieve or replace its attribute value. An object can provide accessor functions which will enable other objects to inquire about or modify its attribute values. Accessor functions facilitate access control to preserve privacy and integrity of attribute values by allowing them to be read and replaced only when appropriate and by those authorized.

3.2.3 Inheritance

The concept of inheritance allows the creation of a new class from an existing class but perhaps with some changes (in terms of adding new operations and data, redefining existing operations, etc.). Inheritance thus provides a very powerful mechanism that allows the sharing of resources (data + operations) among classes.

Here is a typical example of inheritance. We may define a class Media. All the Media objects have common attributes like Name, Media_Type, and Data and common functions like Capture and Playback. We may also define some more specialized classes such as Audio and Video. Each Audio or Video object has some special properties associated with it. However, all of these different types of Media objects have the common properties of the class Media. Hence, all of these new special classes can inherit the attributes and functions from the class Media. In addition to these, each of the new classes has its own set of attributes and functions. The principle of inheritance serves two purposes. First, it saves effort because it is easier to describe only the properties unique to an Audio or a Video than to define all of its properties. Second, inheritance expresses a relationship between Media and Audio or Video that is not expressible on purely structured languages. This is known as an IS-A relationship: that is, an Audio or a Video is a Media.

3.2.4 Polymorphism

Polymorphism is defined as the ability of different objects belonging to different classes to respond differently to the same message [WIN90]. For example, there are two objects: an Audio object and a Video object. Sending a Playback message to these objects would invoke different 'playback' operations.

Polymorphism, along with inheritance, provides programming reusability. New objects can be easily added into an existing system if they respond to the same message (perhaps differently) as the existing objects of the systems.

3.3 Object-Oriented Database Systems

Prior to the existence of object-oriented database management systems, developers of the complex applications who did not take advantage of the benefits offered by object-oriented programming had two choices:

- Use a relational database.
- Use a dedicated file system.

By using a relational database, information on a complex object would be scattered into many relations, leading to the loss of a direct correspondence between a real-world object and its database representation. Besides, the need to explicitly copy and translate data between database and programming language data structures makes the system very slow.

With a dedicated file system, people find that such file systems can be fast because they only do what is required for the application. However, the file systems eventually become a nightmare. Not only do they have to be built, but also they have to be maintained and enhanced to add functionality. Periodically, they have to be redesigned to meet new requirements.

Now, a new generation of object-oriented database management systems, or ODBMSs, provide not only the traditional database features, but also the key ability to treat data as objects, and to navigate across those objects which is ideal for multimedia applications. Object-oriented database management system must first of all be a database management system, and as such must provide the traditional database features. In fact, an object-oriented database management system is a combination of object-oriented programming languages and the traditional database features such as transparently persistent data, concurrency control, data recovery, and associative queries. Thus, the main database issues for the object-oriented database systems must be studied.

3.3.1 Data Modeling Concepts

An object-oriented data model is a set of object-oriented concepts for modeling data. Object-oriented concepts have been embedded in various programming languages. Although there is no standard object-oriented data model right now, a set of commonly accepted and fundamentally important data modeling concepts can be extracted from various object-oriented programming languages. This set of modeling concepts forms an object-oriented data model [KIM90].

Object and Object Identifier

Any real-world entity is an object, with which is associated a system-wide unique identifier [KIM90]. Unlike traditional programming languages which separate the data and the procedure, object-oriented approach encompasses the attributes (data) together with the methods (functions) on the attributes in an object [WOE86].

Attributes and Methods

An object has one or more attributes, and one or more methods which operate on the values of the attributes. The value of an attribute of an object is also an object. An attribute of an object may take on a single value or a set of values.

Encapsulation and Message Passing

Messages are sent to an object to access the values of the attributes and the methods encapsulated in the object. There is no way to access an object except through the public interface specified for it.

Class

All objects which share the same set of attributes and methods may be grouped into a class. An object belongs to only one class as an instance of that class [WOE87a].

Class Hierarchy and Inheritance

Classes can have superclasses and subclasses; together the classes are organized into class hierarchies. For example, for a class Media and a set of lower-level classes {Video, Audio, ...} connected to Media, a class in the set {Video, Audio, ...} is a specialization of the class Media, and conversely the class Media is the generalization of the classes in the set {Video, Audio, ...} . The classes in {Video, Audio, ...} are subclasses of the class Media; and the class Media is a superclass of the classes in {Video, Audio, ...}. Any class in {Video, Audio, ...} inherits all the attributes and methods of the class Media and may have additional attributes and methods. All attributes and methods defined for a class Media are inherited into all its subclasses recursively. An instance of a class Video is also a logical instance of all superclasses of Video.

3.3.2 Transaction Processing

A transaction is a sequence of reads and writes against a database [KIM90]. A transaction as used in conventional applications has two properties: atomicity and serializability. The atomicity property means that the sequence of reads and writes in a transaction is regarded as a single atomic action against the database. It ensures that if a transaction cannot complete, the system backs out all writes which may have been recorded in the database; that if a transaction successfully completes, the system guarantees that all writes are recorded in the database. The serializability property means that the effect of concurrent execution of more than one transactions is the same as that of executing the same set of transactions one at a time. It ensures that a transaction is completely shielded from the effects of any other concurrently executing transactions.

The objective of transaction management is to ensure the atomicity and serializability properties of transactions. Transaction management consists of two

components: concurrency control and recovery. Concurrency control refers to automatic control of simultaneous accesses to a common part of the database by more than one transaction. Recovery refers to the ability to restore the database to a state which existed at some point before the failure of any transaction. Most commercial database systems use a locking protocol to implement concurrency control, and logging protocol to implement recovery. The locking techniques used in object-oriented databases are based on object, class, class hierarchy, and class-composition hierarchy.

The atomicity and serializability properties of transactions have been essential for conventional business data processing applications. However, they have highly undesirable consequences for long-duration transactions, that is, transactions whose duration is much longer than that of conventional transactions. The atomicity property means that if a long-duration transaction cannot complete, all the work that has been done must be backed out. The serializability property means that if a long-duration transaction holds a lock on an object, any other long-duration transaction that must access the same object in a conflicting mode must be blocked until the first long-duration transaction completes.

A long-duration transaction is often defined as a set of conventional short-duration transactions. The atomicity and serializability properties continue to apply to the short transactions which comprise long-duration transactions. Beyond this, however, there is still no clear-cut consensus about the semantics of long-duration transactions. since a transaction is a essential to database consistence, the difficulty with long-duration transactions lies in defining a practical notion of database consistency for interactive and cooperative application environments. This is where the object-oriented approaches find their ways [KIM90].

3.3.3 Object Identity

The object identifier of an object is system-wide unique. The uniform treatment of any real-world entity as an object simplifies the user's view of the real world. An object is recursively related to any number of other objects through some semantic relationships. In object-oriented systems, the relationships between an object and other objects are represented in terms of references to the objects. The references are the values of the attributes of the object.

Object-oriented data models are also characterized by the ability to make references through an object identity. This capability requires that there be something about the object that remains invariant across all possible modification of the object's value. Most modern programming languages can capture object identity, as can some early database models (e.g., network). The relational database models, however, tends to be value-based. In a relational model, an object is identified by a subset of its attributes, called its key. This approach leads to simpler computational structures, but makes some kinds of information difficult to express. Suppose a Video_Frame referred to its containing Video by its name. If the name of a Video changes, its constituent Video_Frames would no longer refer to it. If, however, the Video_Frame held the identity of the Video, a change to the Video will not break the connections to its Video_Frames.

3.3.4 Performance

Performance is an important issue in object-oriented databases. It can be improved by clustering and indexing techniques.

Clustering

Clustering is a database-design technique used to store a group of objects physically close together so that they may be retrieved efficiently [KIM90]. There are two basic alternatives for clustering in relational databases. One is to store tuples of only one relation in the same segment of disk pages, on the basis of the values of an attribute (or a combination of attributes) of the relation; for example, assuming a Video relation corresponding to our Video class, tuples of the Video relation which share the same value in the Name attribute may be clustered. Another is to store tuples of more than one relation in the same segment of pages, on the basis of the relationship between the values of the attributes of the relations; for example, tuples of the Video relation whose Playback_Device attribute shares the same value with the Device_Name attribute of the Device relation may be clustered to facilitate joining of the two relations.

In object-oriented databases, clustering is also an important technique to improve system performance; there are five basic alternatives for clustering. The two clustering alternatives for relational databases also apply to object-oriented databases. The third alternatives is to cluster instances of the class along a class-composition hierarchy; this may be regarded as a variation of the second alternative, namely, clustering tuples from more than one relation in the same physical segment. Since a class-composition hierarchy is equivalent to the specification of joins of the classes on the hierarchy, it may seem like a good idea to cluster instances of all classes on a class-composition hierarchy.

The class hierarchy results in the fourth alternative for clustering instances from more than one class. It may be desirable to cluster all instances of a class hierarchy rooted at some user-specified class. The fifth alternative is to cluster instances from any connected subgraph of the schema graph; namely a class hierarchy rooted at some class, and some of the class-composition hierarchies rooted at any of the classes on the selected class

hierarchy. As in relational systems, the users must specify the scope of the classes to be included in a cluster, namely, the root of a desired class hierarchy, the root of a class-composition hierarchy, and the leaves of the class hierarchies.

Indexing

Relational database systems often allow the users to create a secondary index on an attribute (or a combination of attributes) of a relation for use in evaluating queries against the relation. The data structure most often used for an index is a B tree. It costs only O(log N) index-page fetches to identify a set of tuples that satisfy a search condition involving the indexed attribute, where N is the total number of index pages.

Indexing is also an important technique for expediting query evaluation in object-oriented databases. However, the richness of the data model introduces at least three new types of index: a class-hierarchy index, a nested-attribute index, and a two-dimensional index [KIM90]. A class-hierarchy index is maintained on an attribute of classes on a class hierarchy, while a nested-attribute index is maintained on a class-composition hierarchy. A two-dimensional index is a nested-attribute index augmented with a class-hierarchy index for each class in the sequence of classes between the indexed class and the class to which the indexed attribute actually belongs.

3.4 Summary

In this chapter, the object-oriented paradigm has been described. The object-oriented paradigm is based on features of data abstraction, encapsulation, inheritance, and polymorphism. It provides a more natural way to model real-world complex objects than the traditional languages such as structured languages. It, as well, enhances extensibility of the software systems by providing reusability. Thus, the object-oriented programming technology is suitable for multimedia applications.

In the second part of this chapter, the key object-oriented database issues have been studied: data modeling, transaction, object identity, and performance. Object-oriented database technology combines object-oriented programming mechanisms with traditional database features. Since an object-oriented database system stores objects with the same formats as that in the applications, the "impedance mismatch" problem can be ignored. With the object-oriented modeling power, multimedia applications can store complex objects (e.g., audio, video, multimedia document) in the databases. Thus, these multimedia objects can take advantages of the database features such as: persistence, concurrency control, recovery, etc.

Chapter 4

Multimedia Requirements and Database Systems

The Multimedia Database System which we have developed is an information management system dedicated to multimedia information. Prior to designing the system, we must review the basic multimedia concepts and characteristics, identify the requirements and study the existing database systems in this research area.

4.1 Multimedia Basic Concepts

4.1.1 What is Multimedia?

The term *multimedia* is often defined and used differently. From the perspective of a system, *media* are information carriers which are used to express and communicate information [ROT93]. The typical types of *media* are text, still images, graphics, audio, and full-motion video. It is obvious that the *variety* of media types is an important feature of *Multimedia* systems. On the other hand, in order to deal with the variety, *integration* is a

critical concern. A *multimedia* system allows end users to share, communicate and process a variety of forms of information in an integrated manner [WIL94].

4.1.2 Need

Multimedia communication is important in daily life. When humans communicate with one another, we use a variety of media to interact including spoken language, gestures, and drawings. We take advantage of multiple human sensory systems of communication including vision, audition, and taction [NAF90]. Some media and modes of communication are more efficient or effective than others for certain tasks, users, or contexts (e.g., the use of speech to control devices in hand, the use of maps to convey terrain and cartographic information). Although humans have a natural facility for managing and using multiple input and output media, computers do not [MAY93]. The ability of computers to communicate, store, retrieve and exchange multimedia information in a "natural" way would be a valuable facility for us.

The development of computer technology allows us to store information so that it can be viewed at any time, interrupted, repeated, and changed or enhanced in a variety of ways. As well, the computer can merge the television news, for example, with the newspaper, and link X-ray images with annotation. However, the ultimate goal is not just to store the real world object and allow the user to make changes to it. Rather, it is to understand the function of the real world object and to use the computer system to execute the function more effectively. For example, the function of a book is to provide the reader with information. A book stored in a computer system can be more than just a passive holder of information. It can actively help the user to find and understand the information [WOE87b]. To achieve this function requires a better understanding of multimedia characteristics and requirements.

4.2 MEDIASTORE Requirements

Multimedia logical storage system faces a lot of challenges for the support of multimedia applications. The purpose of MEDIASTORE is to store, manipulate, and retrieve multimedia documents in an object-oriented database system. The multimedia document is composed of complex objects such as text, graphics, image, voice, audio and video. The temporal and spatial relationships may exist between these types of information. The MEDIASTORE must provide the traditional database features (e.g., storage, manipulation, and query) as well as new features (e.g., synchronization, and continuity) for the management of multimedia information within the multimedia document. The main requirements of a multimedia logical storage system are as follows:

4.2.1 Large Data

The size of a multimedia data object can be very large. For instance, a single *second* of high-quality digital video may consume tens of MBytes [BRE92]. Since the large storage requirement, more efficient storage model is needed. In order to store more data in the system, the copying of the huge multimedia data must be minimized. This can be achieved by sharing the huge multimedia data among multiple documents. For instance, a user may wish to copy an image from one document to another. In this case, a physical copy of the image should not be made. Instead, only an internal pointer should be created to allow the second document to share the image with the first one [WOE87c]. Data sharing is commonly identified as a key requirement for multimedia database systems [KLA90][OOM93][MAS89].

In the design of MEDIASTORE, we derive a temporal media stream used in a document from an original stream. The derived temporal stream contains a number of links to the elements within the original stream. Therefore, the original steam can be shared among multiple documents through derivations.

4.2.2 Synchronization

Synchronization is one of the most important requirements in a multimedia system. Synchronization is required to control the event orderings and precise timings of multimedia interactions [WIL94]. To illustrate the need for synchronization, consider the case of a multimedia presentation. In such a presentation, it might be necessary to have an audio track played back with a related video stream simultaneously. In analyzing the requirements of multimedia applications, two kinds of synchronization can be identified:

- intra-media synchronization
- inter-media synchronization

Intra-media synchronization is also called *continuity* on a single media stream. In the case of video playback for instance, the playback rate of the video must be the same as its recording rate. The playback rate must be met for display of successive frames for motion to flow smoothly without any gaps or interruptions [KAM93c]. To maintain continuity, the retrieval of the data stream must be balanced with the required playback rate. However, variations in retrieval are inevitable. As part of the retrieval algorithm, the continuity control algorithm is described in Chapter 5.

Inter-media synchronization corresponds to the situation where it is necessary to maintain the temporal relationship between multiple media streams during playback. A example of this form of synchronization is a 'lip-sync' relationship between a video and its separately stored audio track. The temporal related media are usually integrated into a multimedia document. The temporal relationships between the components have to be represented and maintained during retrieval for document playback.

To maintain synchronization among multiple data streams, the temporal related stream objects are grouped into a special object called concurrent object. This object launches multiple processes for the retrieval of its components. A component object regularly checks the retrieval progress of another relative component and adjusts its own retrieval pace accordingly. Several strategies are provided in MEDIASTORE for flexible synchronization control. The synchronization control among multiple data streams is presented in Chapter 5.

4.2.3 Multimedia Document for Integrating Heterogeneous Multimedia Information

Multimedia information is composed of various media types, such as video, audio, text, and graphics. Furthermore, each media type might be represented in many different formats in terms of the employed compress methods such as JPEG [WAL91], MPEG [LEG91], and DVI [GRE92][LUT91] for video. Each type of information requires appropriate tools for acquisition, processing, transfer, storage and retrieval. Semantic and temporal relationships may exist between these types of information that require a uniform, homogeneous representation and management within the database [KAR93c]. This requirement can be achieved by integrating multimedia information into a single entity called multimedia document. The architecture of the multimedia document represents semantic as well as the temporal relationships among the multimedia objects. In order to represent and manage heterogeneous multimedia objects in a uniform, homogeneous way, the object-oriented approach is employed for the design of a database schema according to the document architecture. With the inheritance mechanism of object-oriented technology, the system can be easily extended by reusing the existing classes. The details of the object-oriented database schema is described in Chapter 5.

The requirements of a multimedia logical storage system discussed above are the main design objectives of MEDIASTORE. The focus of MEDIASTORE is on the storage, manipulation, and retrieval of multimedia documents in an object-oriented database system.

4.3 Multimedia Database Systems Review

Several research centers and universities have been working towards the design and implementation of multimedia database systems. Most of the systems are based on the extended relational models or the object-oriented data models.

4.3.1 NF^2 Model

In Chapter 2, we have described the relational model briefly. In the relational model, the atomicity constraint on the value of attributes is also called First Normal Form rule (1NF). 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows "relations within relations" or relations as attributes of tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values from a domain of such values [ELM89]. This prevents users from dealing with structured objects directly. The structured objects must be flattened into tables and lose their structure.

To remove these restrictions, the relational model is thus extended by using the concept of Non-First-Normal-Form relations (NFNF = NF^2). In this model, the value of an attribute may be atomic or a relation. Relations may be nested in each other so that hierarchical structures are supported. The nested structure of records in NF^2 is also supported in object-oriented model. In addition, object-oriented model supports such concepts as a class hierarchy, inheritance, and methods; NF^2 obviously does not include these concepts [KIM90].

The Advanced Information Management system AIM from IBM is based on the NF^2 model. The purpose of the system is to manage a large variety of data of various types in a consistent and efficient way [DAD89]. The database uses the client-server architecture. The client is slightly enhanced compared to that of a pure relational system. On the server side, one layer is added for handling complex entities and passing their elements to the lower layer. This layer also deals with complex problems such as storage management for unstructured data called BLOBs [SHE90]. However, the system does not support object-orientation which is well suitable for multimedia applications.

4.3.2 OCPN for Temporal Modeling

Little and Ghafoor [LIT90a][LIT90b][LIT92][LIT93a][LIT93b][BER90] propose an approach for the formal specification and modeling of the temporal relationships involved in the multimedia integration. The proposed model is based on logical temporal intervals and Timed Petri Nets. In the following, we briefly describe this method.

Little and Ghafoor believes that given any two atomic processes specified by their temporal intervals, there exists a Petri Net representation for their relationship in time [LIT90b]. This Petri Net presentation is named Object Composition Petri Net (OCPN). The temporal interval here is characterized as a nonzero duration of time in any set of units. Seven basic temporal relations are provided in the Petri Nets as follows: *before, meets, equal, overlaps, during, starts,* and *finishes*. The OCPN model captures the basic temporal relations that exist between every two intervals.

In [LIT90b], the OCPN model is applied by Little to design a database schema to storage of synchronization information for complementary use in association with, or integrated into a DBMS for actual multimedia elements. The database schema called

synchronization schema preserves the temporal relationships by organizing multimedia objects into a binary tree hierarchy. Three types of nodes designed in the schema are *terminal*, *nonterminal*, and *meta* nodes. The *terminal* node is the leaf node which points to the data for playback. The *nonterminal* node specifies the temporal relation between its left and right children. The meta node groups many temporal intervals with a single temporal relation for the compact representation of a common relation among many similar process intervals [LIT90b]. A retrieval algorithm based on the binary database structure is provided. The retrieval process retrieves data from databases in terms of the timing information specified in each node. The concurrency is achieved by the multi-processing technique.

Their method provides a simple way to handle pairwise temporal relations with the binary database structure. On the other hand, the binary structure is not flexible to deal with the case of more than two parallel multimedia objects. As well, the method does not support flexible synchronization control. For example, the change of synchronization control unit may require the change the granularity of the leaf nodes. As a result, the database structure has to be modified. The synchronization schema is not represented by an object-oriented model. Instead, an extended relational model is adopted for utilizing the query languages.

4.3.3 IRIS

The IRIS system [FIS86] from HP is one of the commercial object-oriented database systems. IRIS provides interfaces to C and LISP programming languages. The users may access IRIS through an object-oriented extension of SQL, called OSQL. A graphical interface is also provided with a browser for object-by-object queries. The IRIS object manager currently runs on top of a relational storage system.

The IRIS data model is based on the functional data model. An attribute, called a function, may be single-valued or multi-valued; a multi-valued attribute may have a heterogeneous set of objects as its value. It also supports multiple inheritance. IRIS provides the consistent use of a function formalism in its data modeling. An attribute or a method in a class is defined simply as a function on the class. A function may include a query expression. IRIS supports a lot of database features. These include limited dynamic schema evolution, versions, a query language, and query optimization. As well, IRIS supports transactions, concurrency control, and recovery through HP-SQL. IRIS enhances extensibility by allowing the users to add foreign functions to the system. A foreign function is a user-defined program, for example, a new retrieval method, that extends the capabilities of the IRIS system [KIM90].

IRIS fulfills most of the requirements of multimedia data management. However, IRIS's relational storage system is not able to handle tuples larger than 4 kbytes. This would affect multimedia applications using audio and video.

4.3.4 MIM in the ORION Object-Oriented Database System

Woelk and Kim [WOE87a] describe the implementation of the Multimedia Information Manager (MIM) in the ORION object-oriented database system which is operational at MCC. MIM is part of the object subsystem of ORION. The design objectives of MIM include extensibility, flexibility, and efficiency in supporting many types of multimedia information. The implementation of the MIM contains lattices of classes which represent capture devices, storage devices, captured objects, and presentation devices. These classes may be specialized by the users to add the functionality of the MIM. In this way, the MIM is highly extensible. Their work thus provides one further proof that the object-oriented approach is suitable for multimedia information management.

Although their work introduces a valuable approach on the design and implementation of the MIM using the ORION object-oriented database system, it does not provide features to deal with real time multimedia information. More specifically, it does not support synchronization management and synchronized retrieval.

4.3.5 An Audio/Video Object-Oriented Database System

Gibbs et al.[GIB93][GIB91a][GIB91b][MEY92][MEY93][BRE92] introduce a theoretical framework for an audio/video, or AV, database. In [GIB93], an AV database is defined as a collection of AV values (i.e., digital audio and video data) and AV activities (i.e., interconnectable components used to process AV values). Two abstractions, temporal composition and flow composition, are introduced for aggregating AV values and AV activities respectively. An object-oriented data model is used for incorporating an AV data model and prescribing AV database/application interaction.

In order to avoid explicit descriptions to particular AV data representation, the AV storage and presentation requirements are specified by quality factors. An example of a video quality factor is the form as w x h x d @ r indicating a video resolution of width, w and height h, a depth of d bits per pixel and a rate of r frames per second. The AV values which are simultaneously played back are aggregated using *temporal composition.* Timing information (start time and duration) of each component is specified.

For processing real-time AV data, the database defines a number of AV activities to give applications control over AV data streams through the creation and manipulation of instances of these activity classes. The activity classes are based on a number of concepts, such as activity creation, activity location, activity ports, and activity binding. Those allow applications to link an AV data stream to a particular port. The examples of video activities are: video reader, video writer, video encoder, and video decoder. A set of activities are

grouped using *flow composition* to process so called composite AV values. Activities which process composite AV values will generally contain component activities for each component value. Such a composite of the activities would maintain the synchronization of its component activities, assuring that the streams corresponding to the different components remain temporally correlated.

In [GIB91], more details are provided on the design of the classes for the multimedia objects and the composite objects. Their framework is based on the environment supporting *multi-threads* and *active objects* which may spontaneously perform actions, even if no messages have been sent to the object. Two interesting concepts are introduced for synchronization as follows:

- **Two temporal coordinate systems: *world time* and *object time*.** The two timing systems are used by the methods for synchronization. The origin and units of world time are set by the application. The origin would normally be set to coincide with the beginning of multimedia activity. World time would run while the activity is in progress, and be stopped or resumed as the activity is stopped or resumed. Object time is relative to a multimedia object. In particular, each object can specify the origin of object time with respect to world time and the units used for measuring object time such as the playback duration for a frame as the unit for a video [GIB91]. The world time system provides a common reference to synchronize parallel data streams while the object time system allows the methods on a single data stream to use a reference more suitable for the stream.
- **Temporal composition for synchronization.** The temporal composition concept is introduced to group a number of temporal related objects into a *composite* multimedia object. One of the major responsibilities is synchronization of their components. Four synchronization modes are provided for supporting multiple synchronization methods: NO_SYNC, DEMAND_SYNC,

TEST_SYNC, and INTERRUPT_SYNC. These synchronization methods are based on the interaction between the composite and its components. However, no further details are provided on implementation.

4.4 Summary

In this chapter, the basic concept and the need of multimedia are described. The MEDIASTORE requirements are discussed. Then, some existing multimedia database systems based on extended relational data model or object-oriented data model are examined.

The purpose of this review of the existing systems is to help understand whether current database systems adequately support multimedia (audio and video) and multimedia documents. Although there is some support for simple multimedia applications, it appears that multimedia database systems, as described above, do not have enough facilities to support complex multimedia applications (e.g., multimedia documents). These applications deal with sequences of data which organized into one document, and must be retrieved, processed and presented subject to real-time constraints. The sequences typically require synchronization (e.g., an audio sound track to video) and may require sophisticated processing by the database. Yet support for multimedia documents with temporal sequences, their synchronization and processing, is not part of current multimedia database design. This gives rise to the design of the Object-Oriented Multimedia Logical Storage System.

Chapter 5

A Persistent Multimedia Object-Oriented Storage System

In this chapter, we present the core of our work: the design and implementation of the persistent multimedia object-oriented logical storage system called MEDIASTORE. MEDIASTORE provides an object-oriented database schema for the storage, manipulation, and retrieval of multimedia objects in a multimedia document. A multimedia document architecture is used to integrate and organize multimedia objects including audio and video, as well as to represent the temporal relationships between objects in a document. The focus of MEDIASTORE is on maintaining synchronization between objects in a document during retrieval.

First, we introduce MEDIADOC which is the multimedia document architecture partially supported in MEDIASTORE. Second, we describe the database schema. Third, we present the synchronized retrieval algorithm of multimedia document for playback. Finally, we describe the data sharing, playback and layout, queries, and a graphical user interface.

5.1 Multimedia Document Architecture as the Information Vehicle

The MEDIASTORE integrates multimedia information into a single entity called multimedia document. Accordingly, an object-oriented database schema is designed. The purpose of the schema is to store, and retrieve multimedia information in a multimedia document, which is structured basically in terms of the MEDIADOC multimedia document architecture [EME93a]. Therefore, it is necessary to introduce MEDIADOC first.

5.1.1 The Multimedia Document Architecture (MEDIADOC)

The MEDIADOC [EME93a][EME93b][EME93c] is another main component of the MEDIABASE project. Basically, the MEDIADOC is an extension of the ISO's Office Document Architecture (ODA) [ISO88] standard.

The ODA standard distinguishes the logical and layout structures, which are used to describe a document's contents. The logical structure provides a method for organizing the contents of the document, and it is intended to closely correspond to those aspects of the document structure related to its functional semantic. The content of the document is usually organized to add understanding [KAR93b]. A document may be divided into chapters, chapters into sections, and sections into paragraphs. The layout structure precisely defines where information is to appear on a plane surface (i.e., a terminal display device or paper). Although the current version of the ODA standard supports static media types such as text, raster graphics, and geometric graphics, it does not support continuous media such as video and audio as well as the temporal relationships (i.e., synchronization) between objects within a document [EME93c].

The MEDIADOC defines a more general architecture (compare to ODA) that provides a model for structuring continuous media (i.e., audio and video) as well as discrete media (text, image, graphics) [KAR93b]. In addition to the concepts of logical and layout structures found in ODA, the scenario concept is introduced to express the temporal relationships between objects within a document.

The Global Logical Structure in MEDIADOC identifies seven main object types as follows [EME93c]:

- **media objects.** The media object is the unit for each media in MEDIADOC. The unit can be the smallest unit (i.e., content portion) in MEDIADOC for the media such as frame, sound, string, image or a group of the smallest units such as video, audio, text.
- **sequential object blocks.** Sequential objects blocks define an ordering for objects such as chapters, sections, and paragraphs.
- **concurrent object blocks.** Concurrent objects blocks are useful in describing for example, a video object and its soundtrack.
- **alternate versions object blocks.** When a document contains different versions of an object, an alternative versions object block is used to identify them. For instance, an audio object may have both English and French versions.
- **set object blocks.** Set object blocks provide a means for grouping information without having to specify a specific relationship (sequential, concurrent, or alternate versions) for them.
- **independent object blocks.** When a media object is not logically related to any other media object, then an independent object block becomes its ancestor.

- **scene-related objects.** Scene-related objects provide a means of dividing a document into logical scenes in order to organize the pieces of information contained in the document.

From the viewpoint of organizing a document, the seven main object types are further classified into three types of layers in the logical structure [EME93c]:

- **Lower Layer.** The lower layers include media objects which describe the content portion and the data stream for each media.
- **Mid Layer.** The mid layers are composed of sequential object blocks, concurrent object blocks, alternate versions object blocks, and set object blocks. The mid layer objects describe how their children are related.
- **Upper Layer.** Independent object blocks and scene-related objects form the upper layers of a MEDIADOC document's logical structure.

An example of the MEDIADOC logical structure for a multimedia report is shown in Fig. 5.1. The document has three sections: Introduction, Analysis, and Conclusion. The Analysis section contains an Audio-Visual portion followed by a text portion. The video segment and the audio segment are required to be played back simultaneously. The document also contains an independent object block with an audio segment as the background music.

The Global Layout Structure in MEDIADOC separates visual portions from audio portions by *media group types* and allows a user to specify which media objects should be sent to which devices in an optional *multiple devices layer* [EME93c]. However, further study is necessary for the creation of the layout structure. MEDIADOC's Global Layout Structure is shown in Fig. 5.2.

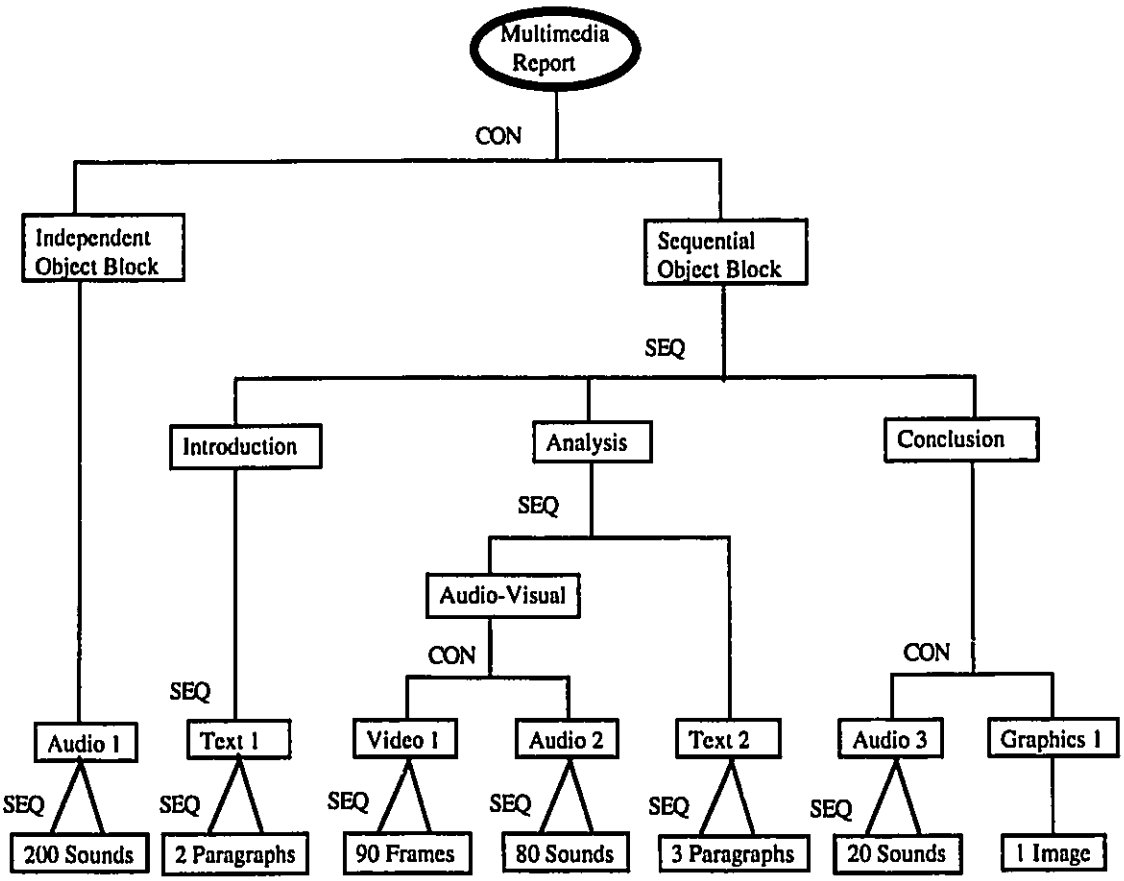

Figure 5.1 Logical Structure of a Multimedia Report

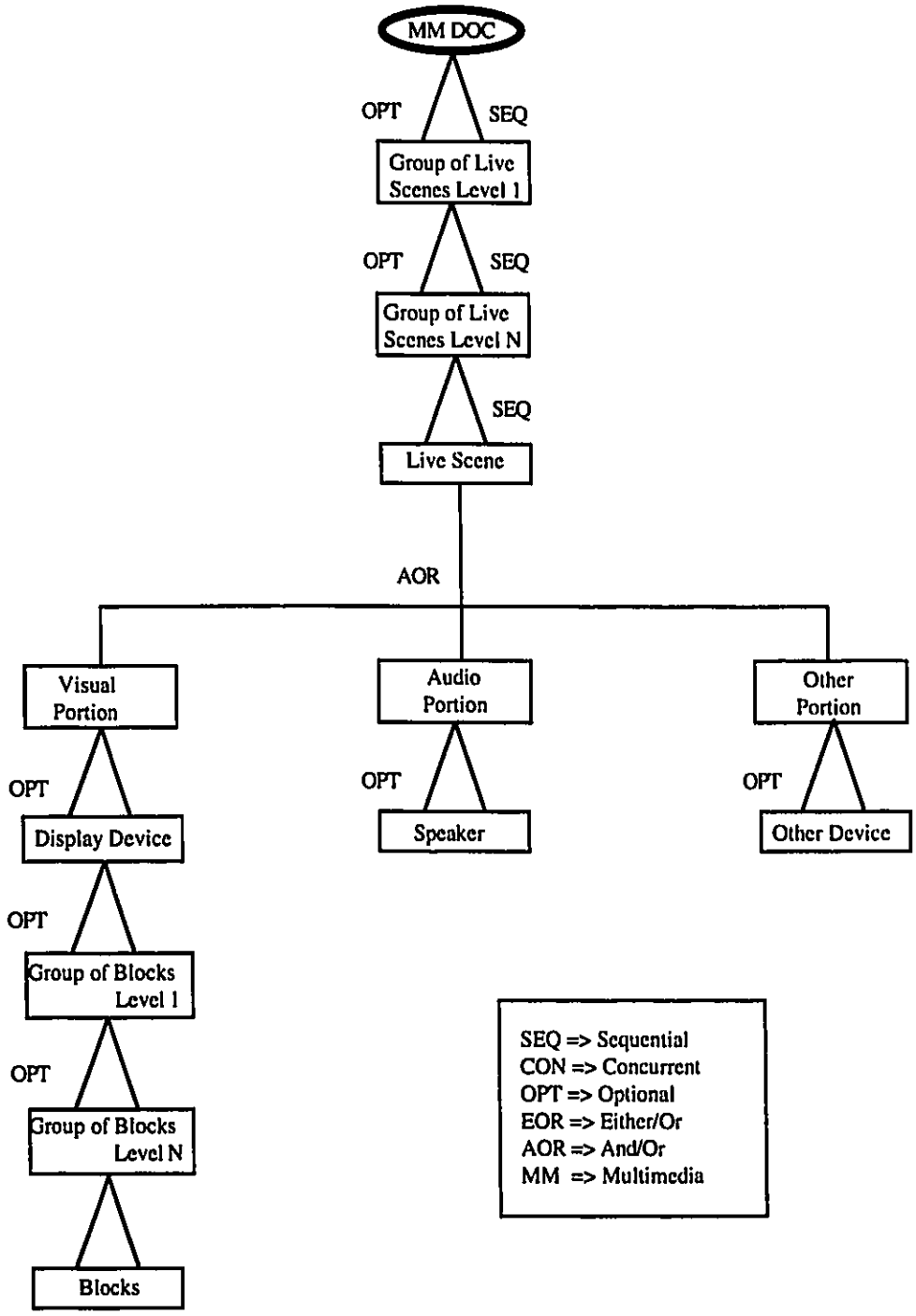

Figure 5.2 MEDIADOC's Global Layout Structure

The MEDIADOC's scenario is formed by selecting media objects from a document's logical structure and by adding temporal information to those objects. The scenario describes when and for how long each media object should be rendered. Three temporal characteristics are identified for each scenario object: start, end, and duration. These three characteristics are related by a simple formula: **duration = end - start.** MEDIADOC also provides three temporal relations for specifying how two objects are related in time. The three temporal relations are: *before* (<), *equals* (=), and *after* (>). The following is an example of a synchronization specification: **Start of A is 5 sec > Start of B**. Since *before* is the inverse relation of *after*, only two of the three relations are necessary. In addition, MEDIADOC supports graphical scenario, and illegal temporal specification detection. More details on MEDIADOC are provided in [EME93c].

It is possible to integrate both continuous media (i.e., audio and video) and static media (i.e., text and graphics) into one document and represent arbitrarily complex synchronization requirements with the MEDIADOC multimedia document architecture and scenario facilities.

5.2 MEDIASTORE's Global Logical Schema

To construct, manipulate and retrieve a multimedia document such as the Multimedia Report shown in Figure 5.1, an object-oriented database schema called the MEDIASTORE's Global Logical Schema is designed. The Global Logical Schema is a number of classes for modeling the logical structure of a multimedia document. With the object-oriented approach, the classes provide not only data but also methods operated on the data. Therefore, the methods for synchronized retrieval and data manipulation are also designed in the classes. The Global Logical Schema in MEDIASTORE is different from the Global Logical Structure in MEDIADOC on the following aspects:

- MEDIASTORE is a logical storage system rather than an application. The Global Logical Schema considers only the common requirements for modeling real-time multimedia applications. The individual application (e.g., MEDIADOC) can extend the schema for its special need.
- MEDIASTORE focuses more on implementation than on representation (e.g., MEDIADOC's Global Logical Structure). The Global Logical Schema provides not only the data structure but also the methods for the synchronized retrieval, data manipulation, etc.
- The design of the Global Logical Schema considers the data sharing issue which is critical for a multimedia database system.

5.2.1 The Classes in MEDIASTORE's Global Logical Schema

MEDIASTORE's Global Logical Schema contains classes for five level main object types as follows:

- **media element objects.** The media element object is the smallest unit for each media in MEDIASTORE. The unit can be video frame, audio segment, and text segment. An audio or text (i.e., caption) stream may be divided into segments for the control of synchronization between a pair of temporal related objects such as an audio track and its corresponding video stream. Using the method of audio segmentation, an audio stream is divided into a sequence of audio segments according to the playback rate of video (e.g., 30 frames/second). One audio segment corresponds to one video frame in terms of the playback time. The media element class is shown in Figure 5.3. The media element class has attributes which are a pointer which points to the data stored in the database, the media type of the data (i.e., audio, video, and text), and the data size for allocating disk space in the database. The Media_Element() and

~Media_Element() are the methods for creating a new element object or deleting the object from the database.

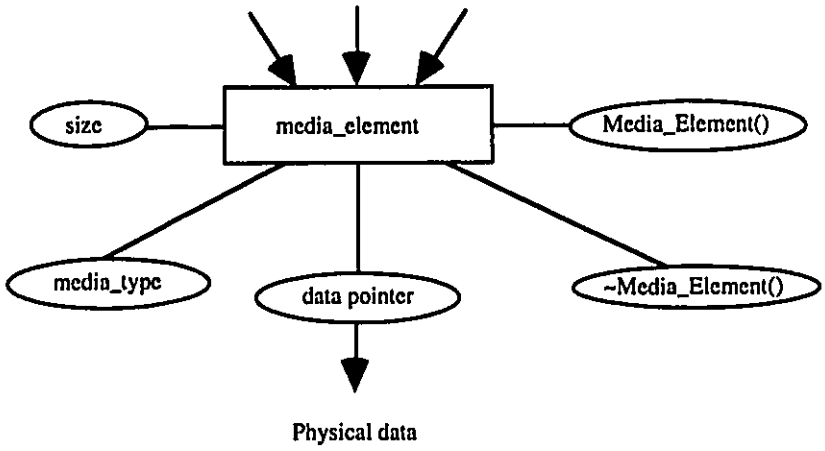

Figure 5.3 media_element class

- **media stream objects.** The media stream object is a sequence of media element objects for the same media with a constant playback rate. The media stream object can be video, audio, text stream, and graphics. The object is considered as the original data stream since it does not represent timing information specific to a document. When created, the initial method inserts the object into the corresponding collection for data sharing purpose. The attributes of the object represent the information about the stream such as the number of elements, playback rate, duration; element size, and the information on the data storage such as a list of storage segments which cluster multiple elements together physically in the disk. The methods of the object include ones for creating and deleting the media stream object, for creating a stream of elements and storing data in the database. Since various information and methods are required to manage different media streams, four media stream object types are designed as follows: video_stream, audio_stream, text_stream, and graphics.

- **temporal stream objects.** The temporal stream object is a sequence of media element objects in a document. The temporal stream object differs from the media stream object on three aspects: (1) possesses document specific temporal information such as start time of the object in the document and temporal relationships with other objects. (2) includes document management methods such as the synchronized retrieval methods. (3) makes use of the derived stream from an original media stream. We distinguish the original media stream object from the temporal stream object in the document for two reasons. First, this allows data sharing among multiple documents. Second, the manipulation on the stream is easier by deriving a new stream from the original one. Actually, this object is the most important one in MEDIASTORE, because the synchronization control is performed through the methods defined in the object. More details will be described in the following sections.
- **composite objects.** the composite object defines a group of objects. According to the time relationships for playback between the components, two object types are identified on this level: sequential, and concurrent objects. The components in the sequential object are played back one after another, while the components in the concurrent object are played back simultaneously. The component in both objects can be sequential, concurrent, and temporal stream object. Therefore, the document hierarchy may have arbitrary complexity. Again, differentiating sequential from concurrent is because of the different retrieval methods defined in the two classes. The retrieval method of the sequential object retrieves its components sequentially in one process. However, the retrieval method of the concurrent object creates multiple child processes for the retrieval of its components.
- **MMDOC.** MMDOC object is the one on top of the document. The object is responsible for representing the general information for the document. When

created, the initial method inserts the object into the MMDOC collection for future query.

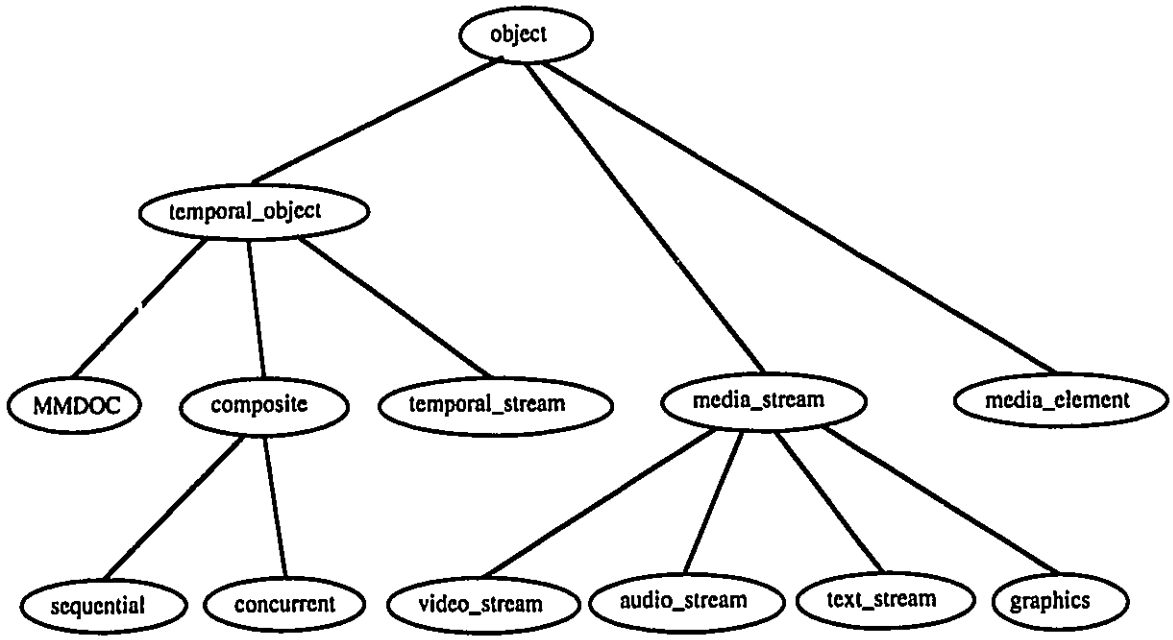

Figure 5.4 The class inheritance hierarchy for MEDIASTORE's Global Logical Schema

The classes in the schema is organized in a class inheritance hierarchy shown in Figure 5.4. The class *object* on the top of the hierarchy is the superclass of all others. It possesses the common attributes such as *global_ID*, name, *object_type*, *parent*, and *children*. The attributes *parent* and *children* are an *Object* and a collection of *Objects* respectively for constructing a composite hierarchy like a document. The value of *global_ID* is a database_unique integer assigned by MEDIASTORE when the object is created. The value of *name* is a string of characters specified by the application and the value of *object_type* can be MMDOC, sequential, concurrent, temporal_stream, video_stream, audio_stream, text_stream, graphics, and media_element. These attributes provide the common basis for the navigational query through the heterogeneous document hierarchy.

The class *temporal_object* is the superclass of the class *MMDOC*, *composite*, and *temporal_stream*. The values of the attributes defined in this class provide timing information necessary for the retrieval process. The attribute *start_time*, *duration*, *stop_time* indicate when and for how long the object should be played back in terms of the playback schedule. The attribute *intermission_delay* is the time interval between the start of an object and the end of its preceding object. The retrieval process will wait for the certain amount of time specified by the value of *intermission_delay* at the beginning of the retrieval for each object. The attribute *global_unit_interval* is the time interval for one global time unit which is used for measuring the whole document in terms of the time for playback. The *global_unit_interval* is measured by *second* in MEDIASTORE. The typical value of *global_unit_interval* is one second. The attribute *current_global_time* specifies the current retrieval time point in global time during retrieval. The *current_real_time* corresponds the *current_global_time* in real time (i.e., the time value obtained from the operating system).

5.2.2 An Example of Multimedia Document in MEDIASTORE

The classes in MEDIASTORE's Global Logical Schema can be used to construct a document directly or indirectly. In the case that the application requires further features, it can create a new class as the subclass of the corresponding class in Global Logical Schema so that the new class can inherit the property of the father class and add its own specific attributes and methods.

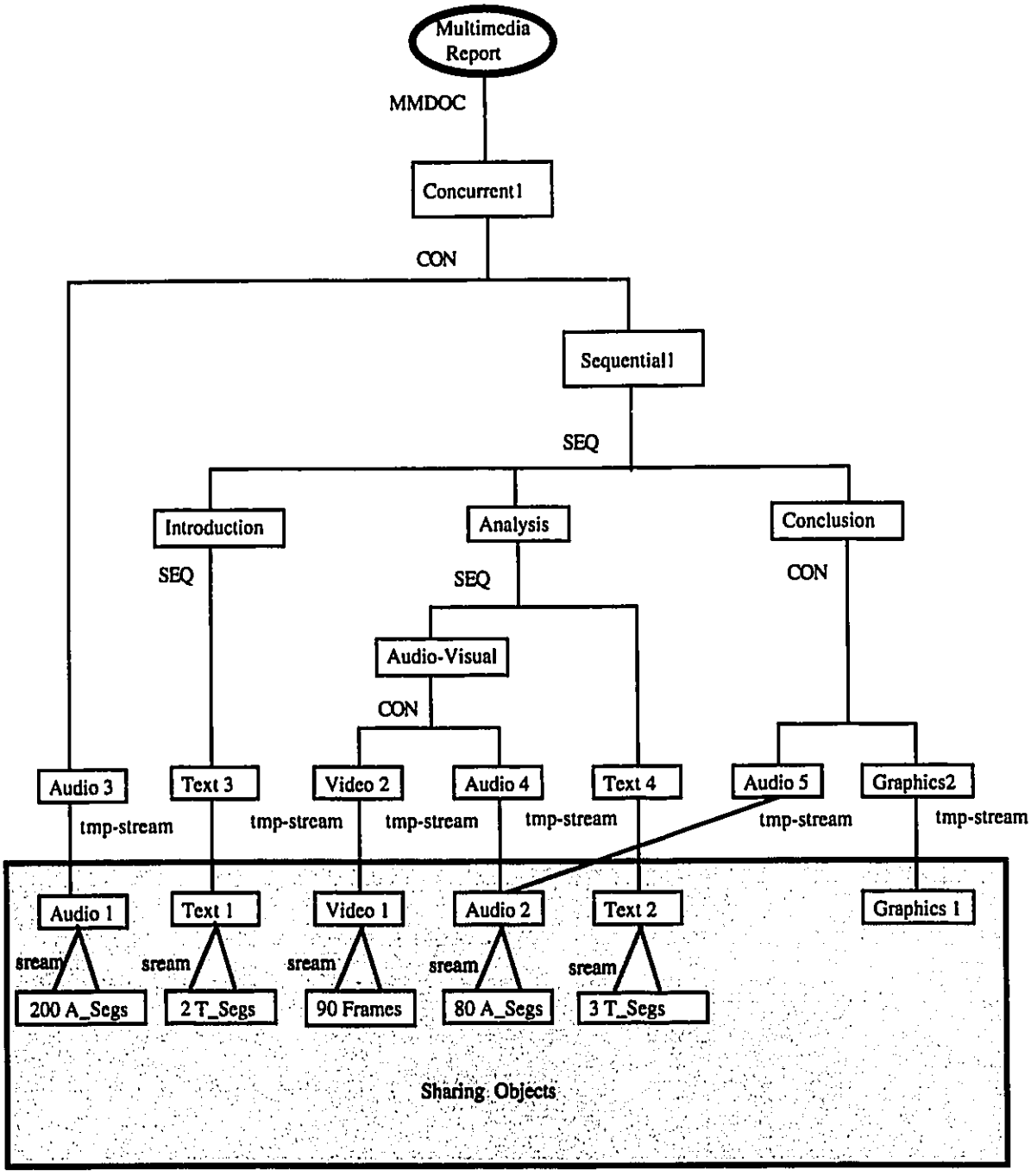

Figure 5.5 Logical Structure of a Multimedia Report in MEDIASTORE

A_Segs: Audio_Segments. T_Segs: Text_Segments

Figure 5.5 shows an example of the MEDIASTORE document structure for the multimedia report which is used above as an example for illustrating the logical structure in MEDIADOC. The document is composed of one MMDOC object: Multimedia Report, three Sequential objects: Sequential1, Introduction, and Analysis, three Concurrent objects: Concurrent1, Audio-Visual, and Conclusion, and seven temporal_stream objects: Audio3, Text3, Video2, Audio4, Text4, Audio5, and Graphics2. The seven temporal_stream objects are derived from the six media_stream objects: Audio1, Text1, Video1, Audio2, Text2, and Graphics1. The object Graphics1 is a special stream with only one component. The media_stream objects are a sequence of the Media_Element objects (i.e., video frame, audio segment, or text segment objects). The media_stream and media_element objects shown in the shadow can be shared by multiple documents or different temporal_stream objects within one document. In this example, Audio2 is shared by both Audio4 and Audio5.

Note that the object independent_object_block is removed from the document. This is because that the case is handled by the concurrent object. The object has an attribute called *control_mode* which allow the user to specify if interaction between the components is required for synchronization control. When the value of *control_mode* is NO_REFER, the retrieval on the components is independent.

5.3 MEDIASTORE's Synchronized Retrieval

5.3.1 An Overview of MEDIASTORE's Synchronized Retrieval for Document Playback

The MEDIASTORE's synchronized document retrieval algorithm is based on the document logical structure. The advantage of using document logical structure for

playback retrieval is that the structure is more natural and understandable than other structures such as the temporal structure which is constructed according to the playback schedule (i.e., scenario). The document logical structure represents both semantic and temporal relationships between the document components so that it is easier for the user to operate on the document in terms of *time* as well as logic. For example, the user can play back only the Analysis part in the multimedia report shown in Figure 5.5.

To maintain synchronization between components in a document for playback, one solution is to retrieve the whole document from a server to local memory or cache before its playback. However, with limited memory or cache space on local workstation, this method is not feasible because a multimedia document may include large objects such as audio and video. Therefore, we design a document retrieval strategy which retrieves and plays back the document simultaneously as well as maintains synchronization during playback.

The synchronized retrieval control is based on the idea that synchronization can be maintained during playback if the data retrievals of parallel streams keep the same pace in terms of the required playback rate. In order to support the retrieval of parallel data streams, the multi-processing technique is adopted.

A recursive retrieval algorithm is designed based on the MEDIASTORE's multimedia document structure. After the multimedia document is constructed and the playback schedule is put into the document, the retrieval algorithm can be invoked from the beginning of the document or any object in the document. In this section, an overall retrieval algorithm on the document structure is described briefly. This provides an overview of the synchronized retrieval control in MEDIASTORE. The algorithm is as follows:

Overall Synchronized Retrieval Algorithm:

Case #1 *sequential* object:

1.1 wait for Ti. (Ti is the intermission delay)

1.2 invoke the retrieval algorithm on the first child object.

1.3 repeat 1.2 on the second, third, ... until the last child object.

Case #2 *concurrent* object:

2.1 wait for Ti.

2.2 if (Control_Mode != NO_REFER) {need interaction between child objects}

create a shared table to record *Current_Global_Time*, and *Status* for every immediate child.

(used by child objects for synchronization monitoring and recovery)

2.3 create a new process and invoke the retrieval algorithm on the first child object

2.4 repeat 2.3 on the second, third, ... until the last child object.

2.5 wait until all its immediate child processes terminate.

2.6 if (Control_Mode != NO_REFER)

release the shared table

Case #3 *temporal_stream* object: (video, audio, text stream, and graphics)

3.1 wait for Ti

3.2 create a new process and invoke the playback algorithm.

3.3 retrieval data once a unit until the end of the object .

if required, conduct synchronization test and adjustment after each unit retrieval.

(detailed algorithm will be described in section 5.3.4 and 5.3.6)

The above algorithm achieves concurrency through the concurrent objects in a document by creating multiple child processes to retrieve the components. However, the synchronization between the component data streams within a concurrent object is controlled in the *temporal_stream* retrieval algorithm. In other words, it is up to individual data stream retrieval algorithm to monitor and adjust its own retrieval pace. Section 5.3.4 and 5.3.6 describe this algorithm in detail.

Note that the *temporal_stream* retrieval algorithm creates a new process for playback. The separation of playback from retrieval is for two reasons as follows:

- The retrieval algorithm can start data retrieval prior to the playback to prevent the variations in retrieval.
- The retrieval algorithm can choose a bigger data unit for retrieval than that for playback so that the minor fluctuations in retrieval within a retrieval unit can be absorbed.

5.3.2 Timing Coordinate System Conversions

MEDIASTORE makes use of two timing coordinate systems: *global time* and *local time.*

The origin and units of global time are set by the application. Usually, the origin will be set to the beginning of the document retrieval for playback. In a document, the values of attribute *start_time*, duration, *stop_time* of each object are set in global time in terms of the pre-determined playback schedule. The purpose of global time is to achieve the synchronization control among multiple parallel data streams by using a single timing coordinate system.

Local time is used in each multimedia stream object. The origin will be set according to global time. The units will be specified for measuring object time. Normally, the units will be set in terms of the data rates of the multimedia stream object.

To illustrate the conversions between the two timing coordinate systems, we define the time interval of one unit for global time as global_unit_interval and that for local time as local_unit_interval, and the ratio of global_unit_interval to local_unit_interval as global_to_local, then the local time is converted into the global time in terms of the following formula:

global_time = start_time + local_time / global_to_local

If both unit intervals are measured by second, letting global_unit_interval be 1 second, local_unit_interval for a video stream be 1/30 second (which is the inverted value of the playback rate 30 frames/second), the current local time be 60 which corresponds the sixtieth frame in the video stream, and the start time of the video stream object in global time be 2, then the corresponding global time will be 2 + 60 / (1/(1/30)) = 4 seconds.

Note that the global time is not the same as the *real* time. It is a logical timing coordinate system whose value is set in terms of the playback schedule and can be mapped to certain point in the document. For example, when the start time of an object is 2 seconds in global time, the start point of the object can be located with the value of the global time. With the global time, statues of the retrieval can be monitored. Two parallel data streams may have different values of global time at the same time point during retrieval. If the difference of the values is bigger than the value of synchronization tolerance, the two parallel streams are said to be out of synchronization. Therefore, the global time provides a basis for the synchronization control system.

The two timing coordinate system conversion is defined in temporal_stream class. A partial specification concerning the timing coordinate system conversion in class temporal_stream is as follows:

```
class temporal_stream: public temporal_object{
public:
//
// timing coordinates
//
//inherited attributes:
global_time  start_time =2.0;
global_time  duration=3.0;
global_time  stop_time=5.0;
global_time  intermission_delay=0.0;
global_time  current_global_time=4.0;
real_time  current_real_time;
real_time_interval global_unit_interval=1.0;
//attributes:
local_time  current_local_time=60;
real_time_interval local_unit_interval=0.033;
float  global_to_local=30;
//methods:
global_time  Current_Global_Time();
global_time  Local_To_Global(local_time);
local_time  Current_Local_Time();
local_time  Global_To_Local(global_time);
global_interval Local_To_Global_Unit(local_interval);
local_interval  Global_To_Local_Unit(global_interval);
//
// stream synchronization
//
//attributes and methods
//
// playbcak and layout
//
//attributes and methods
//
// data stream derivations from original media stream
//
//attributes and methods
};
```

The class *temporal_stream* inherits a number of attributes from its superclass *temporal_object*. These attributes have been described previously. In addition, a number of attributes and methods are defined in the class for the conversion between the global and

local times according to the conversion formula discussed above. To propagate the value of global_unit_interval from top to bottom on the document structure, a method is defined in class Composite. The recursive propagation method sets the common used value to every object through the *children* links.

The class temporal_stream also provides attributes and methods for synchronization, playback and layout, and data stream derivations. These methods will be discussed in the following sections.

5.3.3 Continuity Control over a Single Data Stream

One of the main requirements of MEDIASTORE is to ensure continuity on a single data stream. To be more specific, we define continuity as follows: A data stream with playback rate r is continuous to a tolerance Δt if the interval between two consecutive elements is between $1/r - \Delta t$ and $1/r + \Delta t$ where the $1/r$ is the required interval between two consecutive elements.

To maintain continuity, the retrieval process must ensure that the retrieval of the data stream balances with the playback rate. A data stream may fall out of continuity for two reasons. First, hardware delays and variations in performance may cause the data stream drift out of continuity. This is often called *jitter*. Second, a data stream retrieval may not be able to keep up with the required playback rate.

In the first case, the retrieval of a data stream is fast enough for the required rate. Specifically, the average retrieval time for an element in the stream is less than or equal to the required duration for playing back an element. However, fluctuations in retrieval may bring about continuity problem. To maintain continuity, one strategy is to separate the retrieval process from the playback process. The retrieval process may start retrieving data

a certain amount of time (e.g., a unit time for playback) before the start of playback, then retrieve a unit (multiple elements) instead of an element each time in order to absorb the minor fluctuations within a unit. At the same time, the playback process plays back the stream element by element in terms of the playback rate.

In the second case, the retrieval of a data stream is too slow for the playback rate. Two strategies can be employed for this situation. First, the buffering technique may be used to retrieve a certain amount of data in advance. In this method, the required buffer size may be huge. For instance, in the case of a video stream retrieval, if the retrieval rate is 10 frames/second, the playback rate is 30 frames/second, and the size of a frame is 1 MBytes in a video stream, for the retrieval of 2 seconds video, 40 MBytes buffer will be required. The huge buffer requirement makes this technique impractical for the systems with limited resources. The alternative methods are desirable for those systems. Second, a degraded quality for the playback may be necessary in order to reduce the retrieval requirement. For example, we may retrieve only 1 frame in every 3 frames in a video stream. In this way, the playback rate will be 3 times slower than the original rate. Therefore, there will be much more time available for data retrieval. MEDIASTORE allows a user to choose various quality levels of a data stream by using the derivation method Scale_Of() to derive a new stream from the original media stream. On the above example, the Scale_Of(3) method is used to create a new stream for playback. More details on the method are provided in 5.4 on data sharing in MEDIASTORE.

5.3.4 Continuity Control in MEDIASTORE

Since a variety of continuity strategies may be used, each stream object has a continuity mode attribute. According to the value of the attribute, the retrieval process uses different methods to control continuity. At present, two modes are supported: NO_CONT, and DO_CONT:

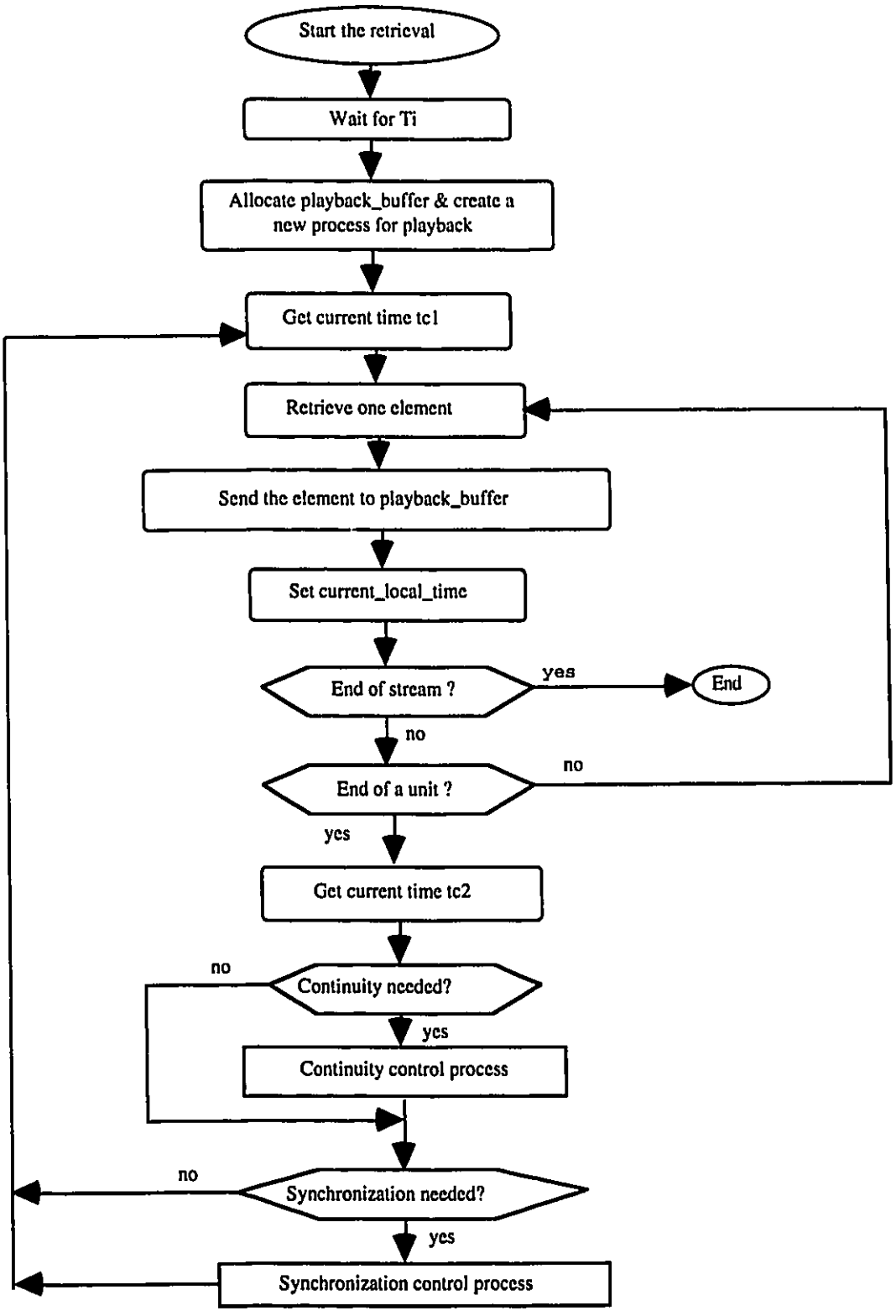

Figure 5.6 The flow chart of a single data stream retrieval process

- NO_CONT: the retrieval process ignores the continuity operations.
- DO_CONT: the retrieval process performs continuity measurement after each data unit retrieval. If the stream is out of continuity, actions can be invoked. The typical actions are waiting, sending an error message; or aborting the retrieval.

In MEDIASTORE, continuity control process is part of the retrieval process over a single data stream. In order to understand continuity control process, we first of all introduce the retrieval process briefly. The flow chart of one data stream retrieval process is shown in Figure 5.6. After the retrieval method on the stream is invoked, wait for a time interval specified by the intermission_delay value. When the *start_time* is reached, allocate playback_buffer and create a new process for the playback method. Then, change attribute *status* from INACTIVE to ACTIVE, get the current time t_{c1} and start retrieving an element After the element retrieval, set the *current_local_time*. If its parent object is a concurrent object and the synchronization mode is REFER_SYNC, set the *current_global_time* in the shared table so that the other streams can get the *current_global_time* of this stream for synchronization control. As well, send the element to the *playback_buffer*, which is used by the parallel playback process. If the element is the last one, set *status* to END. If the element is not the last one of the unit, continue to retrieve the next element. After the whole unit retrieval, get the current time t_{c2}. The unit retrieval time is: $t_{ri} = t_{c2} - t_{c1}$. If continuity control is required, perform the continuity control process. If synchronization control is required, perform the synchronization control process. Then, continue the retrieval by repeating the above steps.

MEDIASTORE allows the application to specify the *retrieval_unit*, Time_Out, and the continuity tolerance. The *retrieval_unit* can be, for instance, a single frame or multiple frames in the case of a video stream. The continuity tolerance can be, for instance, 0.1 second.

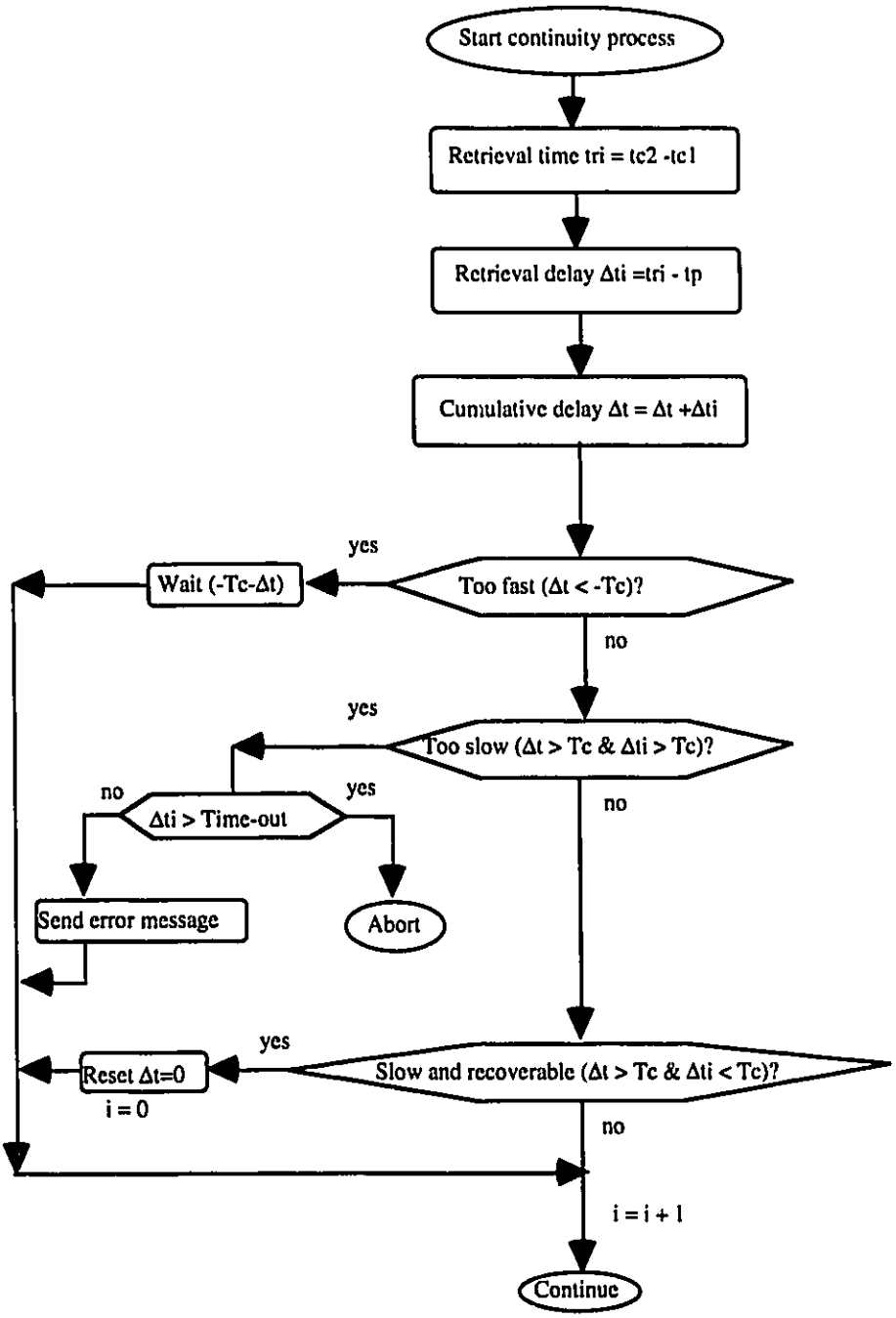

Figure 5.7 The flow chart of continuity control process

Tc: The continuity tolerance

t_p: The required playback time for a unit

The flow chart of one data stream retrieval process is shown in Figure 5.7. The continuity control is performed right after each unit is retrieved. As described above, the retrieval time for the *i*th unit is defined as $t_{ri} = t_{c2} - t_{c1}$. The playback time for a unit is t_p. The retrieval delay of the *i*th unit can be written as:

$\Delta t_i = t_{ri} - t_p$ and, the cumulative delay is:

$\Delta t = \sum \Delta t_i$

The continuity tolerance is defined as Tc. The continuity control algorithm is as follows:

Case #1 ($\Delta t < -Tc$) {retrieval is too fast}

wait for ($-Tc - \Delta t$) and continue; i = i + 1;

Case #2 ($\Delta t_i > Time\_Out$) {retrieval is extremely slow}

out of continuity, abort, set status = ABORTED;

Case #3 ($\Delta ti > Tc$ & $\Delta t_i < Time\_Out$) {retrieval is too slow}

out of continuity, send error message and continue;

Case #4 ($\Delta t > Tc$ & $\Delta t_i < Tc$) {retrieval is slow and recoverable}

reset $\Delta t = 0$ and continue; i = 0; i = i+ 1;

Case #5 ($-Tc < \Delta t < Tc$) {retrieval is in phase}

continue; i= i + 1;

The advantage of this algorithm is that it can absorb the minor discontinuity within and between individual units' retrieval because the delay is only accumulated on units.

5.3.5 Synchronization Control among Multiple Data Streams

MEDIASTORE provides synchronization control among multiple data streams. To be more specific, we define synchronization as follows [GIB91a]: Two data stream objects, m_i and m_j , are synchronized to a tolerance Δt if the difference between the current global time of m_i and the current global time of m_j is less than Δt.

The two reasons which cause continuity problem may also cause synchronization problem over multiple data streams. In addition, different streams may come from different sources such as a video stream and its 'lip-sync' related and separately stored audio stream. The mismatch between these streams during the retrieval may make them fall out of synchronization.

The strategy which retrieves data in advance can also be used to ensure synchronization among multiple data streams. However, as discussed above on continuity, this strategy requires huge buffer size which is impractical for the systems with limited resources. On the other hand, synchronization can be maintained if all the data streams can be retrieved on schedule. However, with current commercial database systems, it is very hard to control the time for retrieval. This is because that the retrieval time depends on external factors like the load of the processor, the number of other processes, the process scheduling policy [GIB91b], or the insufficient resources. Therefore, the mismatch between data streams in retrieval are inevitable. To maintain synchronization, the retrieval process must ensure that the related data streams are retrieved at the same pace. This can be achieved by using a common reference to measure and adjust the retrieval pace of multiple streams which require to be played back simultaneously.

5.3.6 Synchronization Control in MEDIASTORE

MEDIASTORE synchronization control is based on the use of multi-processors for retrieving concurrent data streams. Each data stream is retrieved by one retrieval process. To control synchronization, a reference is specified for each data stream by application. The reference can be either real-time clock or another data stream. It is the responsibility of the individual retrieval process to measure the synchronization status against the reference and adjust the retrieval pace accordingly when synchronization is required for the data stream.

5.3.6.1 Three Complementary Synchronization Strategies

To maintain synchronization between multiple streams, MEDIASTORE provides three complementary strategies:

- synchronize all the streams to real-time clock.
- synchronize one or several streams to another stream (by using method this.ReferTo(referred_object)).
- synchronize several streams with each other (by using method: m_i.ReferTo(m_j) in object m_i; and m_j.ReferTo(m_i) in object m_j).

The first strategy uses real-time as the reference. All the streams must keep in pace with real time. If a stream falls out of synchronization with real-time clock, it needs to adjust its retrieval pace by waiting, jumping, or aborting. However, if the average retrieval speed of the system is slower than the required playback speed, some of the streams have to keep jumping all the time. This makes the playback quality unacceptable.

The second strategy chooses a specific stream as the reference. For example, an audio stream may use the corresponding video stream as its reference. When the audio retrieval is faster or slower than the video retrieval, the audio retrieval process needs to adjust its retrieval pace by waiting or jumping. In this way, the synchronization between the video and the audio streams are maintained by adjusting only the audio retrieval when both streams are slower than the required rate. This may be preferable because different media streams have different requirements for maintaining the playback quality. Usually, a video stream can tolerate delay by playing back the previous frame longer. Therefore, the second method provides the application more choices to control playback.

The third strategy synchronizes multiple streams with each other. When a video stream and its audio track refer with each other, both streams need to adjust its own retrieval pace to follow the other. This strategy is favorable when the synchronization requirement is strict such as the synchronization between a video and its sound track. Although *jump* may be issued on both streams for the synchronization recovery, the frequency of *jumps* is much lower than that using the first strategy if the retrieval rate is slower than the playback rate. Consequently, the playback quality of this method is better than the first method, but may be worse than the second one, if the retrieval is slow.

5.3.6.2 Synchronization Control Algorithms

As described in the synchronized retrieval overview, the synchronization control is achieved through concurrent object and temporal_stream object. The concurrent object creates a new process for each component for its retrieval, and a shared table for the communication between components. It is up to the individual stream to check the retrieval progress of the related stream and adjust its own retrieval pace.

Similar to continuity control, each stream object has a synchronization mode attribute. There are three synchronization modes: NO_SYNC, REALTIME_SYNC, and REFER_SYNC. The operations a retrieval process takes for a data stream object of each mode can be described briefly as follows:

- NO_SYNC: the retrieval process ignores the synchronization operations for the data stream retrieval.
- REALTIME_SYNC: the retrieval process uses real-time clock (default value) as its reference. After each data unit retrieval, the process compares the retrieval time with the required playback time. If the stream is out of synchronization, the corresponding actions can be invoked. The typical actions are wait, jump, and abort.

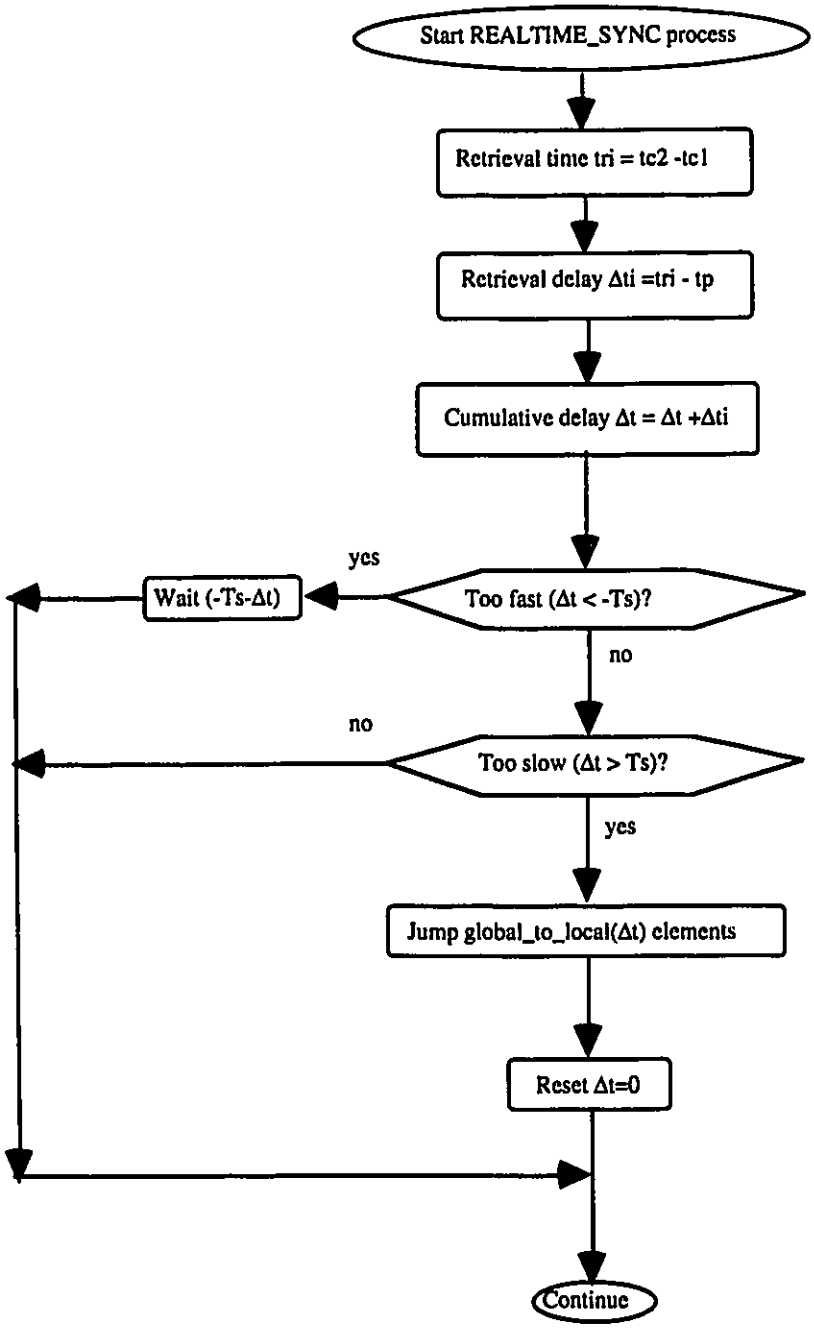

Figure 5.8 The flow chart of REALTIME_SYNC synchronization control process

Ts: The synchronization tolerance

t_p: The required playback time for a unit

- REFER_SYNC: the retrieval process uses another parallel stream as its reference. After each data unit retrieval, the process compares its own *current_global_time* with the *current_global_time* of the referred object. If the stream is out of synchronization, the corresponding actions can be invoked. The typical actions are wait, jump, and abort.

The flow chart of REALTIME_SYNC synchronization control process is shown in Figure 5.8. The REALTIME_SYNC synchronization control is performed after each unit is retrieved. As defined earlier in continuity control process, the retrieval time for the *i*th unit is defined as $t_{ri} = t_{c2} - t_{c1}$ and the playback time for a unit is t_p. The retrieval delay of the *i*th unit can be written as:

$\Delta t_i = t_{ri} - t_p$ and, the cumulative delay is:

$\Delta t = \Sigma \Delta t_i$

The synchronization tolerance is defined as Ts. The REALTIME_SYNC synchronization control algorithm is as follows:

Case #1 ($\Delta t < -Ts$) {retrieval is too fast}

wait for ($-Ts - \Delta t$), and continue;

Case #2 ($\Delta t > Ts$) {retrieval is too slow}

out of synchronization, jump global_to_local (Δt) elements, reset $\Delta t = 0$ and continue;

Case #3 ($-Ts < \Delta t < Ts$) {retrieval is in phase}

continue;

In Case #2, when the retrieval is too slow, the algorithm issues a jump to recover the synchronization. The Global_To_Local (Δt) method translates the delay (global time) to the number of elements (local time) for the jump.

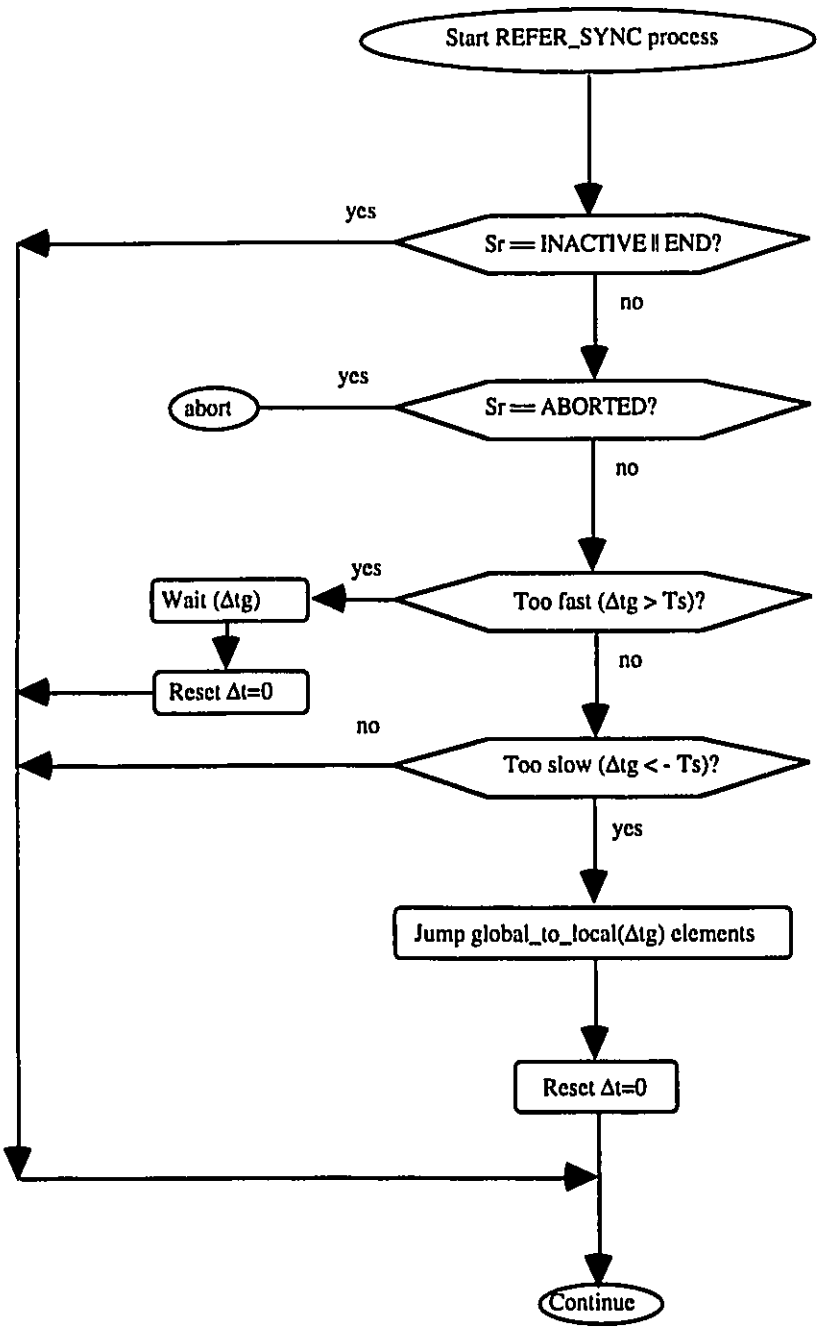

Figure 5.9 The flow chart of REFER_SYNC synchronization control process

Ts: The synchronization tolerance. Sr: The status of the referred stream.

Δtg: The global time delay.

The algorithm looks similar to that of the continuity control since both of them are measured by the real-time clock. Although they all prevent the retrieval from going too fast, only the REALTIME_SYNC control algorithm conducts synchronization recovery by jumping ahead when the retrieval is too slow. The selection of the control methods is based on the requirements of applications. When the data stream does not have temporal relationships with other streams, usually, the continuity control algorithm is preferable. Otherwise, the other control methods have to be selected.

The flow chart of REFER_SYNC synchronization control process is shown in Figure 5.9. The REFER_SYNC synchronization control is performed after each unit is retrieved. The current_global_time of the referred data stream is defined as t_{rg}, current_global_time of its own is defined as t_g, and global time delay is defined as $\Delta t_g = t_g - t_{rg}$. The retrieval status of the referred data stream is defined as S_r. The synchronization tolerance is defined as Ts. As defined on continuity control, the cumulative delay is Δt. The REFER_SYNC synchronization control algorithm is as follows:

Case #1 (S_r == INACTIVE || S_r ==END) {reference is inactive}

continue;

Case #2 (S_r == ABORTED) {reference is aborted}

set status = ABORTED and abort;

Case #3 ($\Delta t_g > Ts$) {retrieval is too fast}

wait for (Δt_g), reset Δt=0, and continue;

Case #4 ($\Delta t_g < -Ts$) {retrieval is too slow}

out of synchronization, jump global_to_local (Δt_g) elements, reset Δt=0 and continue;

Case #5 (-Ts < Δt_g < Ts) {retrieval is in phase}

continue;

In Case #1, when the retrieval of the referred stream object has not being started or has already being ended, the retrieval status of the referred stream is INACTIVE or END respectively. In Case #2, the algorithm aborts the retrieval when the retrieval of the referred stream is aborted. Considering the case of the retrieval of a video stream when its reference is its audio track, if the retrieval of the audio track is aborted, it is reasonable to abort the video stream too.

5.4 Data Sharing

Multimedia information is usually very large. For instance, one second of high-quality digital video may consume tens of MBytes. Very large capacity storage systems are required. On the other hand, keeping multiple copies of large objects such as video and audio data can cause a serious waste of disk space. In MEDIASTORE, we separate the original data stream from a specific document. A document has a temporal stream object derived from the original data stream through the links. The advantages of this method are as follows:

- The original data stream can be shared by multiple documents through the links to it so that copying immense amount of data is avoided.
- The editing on the temporal stream in a document is easier by changing the links to the original data stream.

The data stream derivation methods are part of class temporal_stream. A partial specification on data stream derivations in class temporal_stream is as follows:

```
class temporal_stream: public temporal_object{
public:
    //other attributes and methods
    //
```

```
    // data stream derivations from original media stream
    //
    //inherited attributes
    os_List<Object*>        &children;
    //attributes
    media_stream            *original_stream;
    int                     rate;
    int                     size;
    //methods
    void                    Copy_Of();
    void                    Part_Of(int,int);
    void                    Scale_Of(int);
};
```

The Copy_Of, Part_Of, and Scale_Of are the derivation methods provided in MEDIASTORE. These methods derive a new stream from the *original_stream*. The result stream is the *children* which is used in retrieval. The attribute *rate* specifies the playback rate of the new stream. The attribute *size* specifies the data size of an element in the new stream. The value of the attributes is derived from that in the *original_stream* in the derivation methods. The derivation methods can be used during the document construction and modification. The Copy_Of() method copies all the element pointers from the original stream to the *children* list. The Part_Of () method duplicates part of the original stream. In the case of Part_Of (4, 14), the method reproduces the element links for a substream from the 4th element to the 14th element in the original stream. The Scale_Of (3) method makes links to one element in every 3 elements. As a result, the new stream contains the 1st, 4th, 7th, etc elements in the original stream. Accordingly, the rate is set to original_rate/3.

5.5 Playback and Layout

To maintain continuity over a data stream, we separate the playback process from the retrieval process. This is because of the unpredictability in retrieval with current physical storage system. As described in 5.3.1 the separation strategy allows the retrieval process to retrieve data in advance for a certain amount of time and to use bigger data unit for retrieval in order to prevent variations in retrieval.

The playback and layout attributes and methods are defined in class temporal_stream. A partial specification about playback and layout in temporal_stream class is as follows:

```
class temporal_stream: public temporal_object{
public:
    //other attributes and methods
    //
    // playback and layout
    //
    //attributes
    global_time                 playback_delay;
    os_List<Object*>            &playback_buffer;
    int                         rate;
    layout                      *stream_layout
    //methods
    void                        playback();
};
```

The playback process is as follows:

1. Get current time t_{c1}

2. Setup playback device and layout

3. Get current time t_{c2}

4. If (Δt = playback_delay - (t_{c2}-t_{c1}))> 0) then wait for Δt

{playback should start after the retrieval for the time interval specified by playback_delay}

5. If (not end of the stream, and *playback_buffer* is not empty) then playback an element

6. wait for 1/rate

7 Repeat step 5, 6 until the end of the stream

8. Release *playback_buffer*

9. Disconnect the playback device

10. End of playback

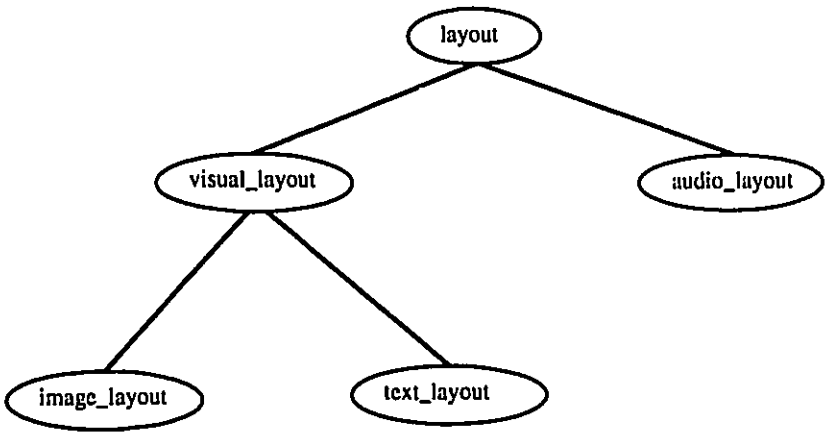

Figure 5.10 The layout class inheritance hierarchy

Simple layout structure is defined in MEDIASTORE to specify the playback device, and the position where the information is to appear on the screen for visual objects. The layout class hierarchy is shown in Figure 5.10. Figure 5.11 shows details of part of the class hierarchy for the layout, visual_layout, and audio_layout class. Each layout object has a pointer to a playback device object specified in the device attribute. Three methods

are defined in the layout class. The Setup() method sets up the playback environment which is everything necessary before the data transfer. In the case of visual object, the playback environment is setup in terms of the spatial layout information. The attributes *up_left_x* and *up_left_y* describe the position on the screen to display the object. The attributes *width* and *height* specify the size and shape of the display area. The attribute *depth* specifies the number of bits stored for each pixel in the spatial object. The Playback_Element() put an element data to playback device. The Disconnect() method disconnects the playback device and releases the resources. Since different device requires different methods to setup, playback, and disconnect, we do not have intention to implement these methods at present.

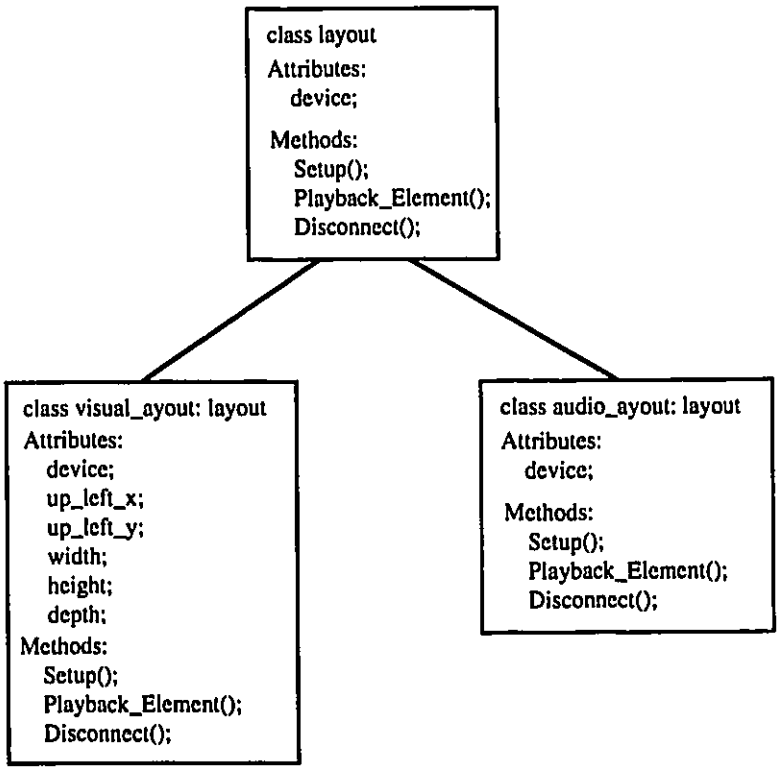

Figure 5.11 The partial detailed layout class inheritance hierarchy

5.6 Queries on Documents and Class Extents

The capability of finding out a required object from a database is a basic feature for a database system. Two complementary methods are used for querying multimedia objects from the database. The first method is based on the document structure. The multimedia objects may be searched from the database by navigating within the document hierarchy. This is called navigational querying method. The second method is based on the different class extents. The multimedia objects may be searched by querying over class extents. This is called associative querying method. In our system, both querying methods are available.

5.6.1 Navigational Queries on Document Logical Structure

The purpose of a navigational query is to retrieve a component within a specified document. When a user want to modify a document, the first thing to do is to find out the component object required to be changed. With the navigational querying method, a user may find out the component by using the children pointers to navigate along with the document structure. The common texture attributes (e.g., name, ID) can be used as querying conditions. The reason is that the document is a multimedia document which is composed of objects of different classes. Every class has a number of common attributes as well as a set of special attributes. These common texture attributes can be used as the common basis for the navigational querying over diverse objects within a multimedia document.

5.6.2 Associative Queries on Class Extents

In addition to the navigational querying, the associative querying is provided. In our system, a database may have many multimedia documents. When a user wants to find out a specific document, he needs to do querying over the multimedia document extent which is a collection of multimedia document objects, each contains the root of the

document hierarchy. Moreover, when a user creates a new document, he may want to reuse some objects which already exist in the database. In this case, the user needs to find out the existing object he wants to reuse. Sometimes the user does not have the knowledge about which document the object belongs to, he only knows the object itself, such as type, name, etc. Therefore, we need the associative querying over the specific class extent.

5.6.3 The Collections for the Associative Queries

The associative querying is based on the class extent, which is a collection of all the instances of a class. Unfortunately, ObjectStore does not provide the feature to maintain such class extents. Therefore, we have to create and maintain some collections for associative querying. In our database system, we have designed six collections: global, MMDOC, video, audio, text, and graphics collections

The global collection contains pointers of all the objects within the database. The initiation method in the top superclass Object assigns a number as the value of the attribute global_ID for every object when the object is created. The value of global_ID is unique within the database, and cannot be modified by users.

The MMDOC collection contains the pointers which point to the roots of multimedia documents. This collection is used to query a multimedia document. The video, audio, text, and graphic collections are for video , audio, text, and graphic objects respectively. The initiation method in the component classes assign a local_ID to the object contained in a local collection when the object is created. For example, the initiation method in the MMDOC class assigns a MMDOC_ID to the MMDOC object when the MMDOC object is created. In the MEDIASTORE, five local_IDs are maintained in a database: MMDOC_ID,

video_ID, audio_ID, text_ID, and graphics_ID. These local_IDs are unique within the local collections.

The database creation method creates these collections when the user initiates a database. When an object is created during the document creation or modification, the initiation method in the class for the object will insert the object pointer into the global and the corresponding local collections automatically. For example, when the user creates a video object, the initiation method in the video class will insert the video object pointer into the global and the video collections.

The navigational as well as the associative querying allows the user to query based on either the document structures or the collections. The two querying methods offer more flexibility for users to do query.

5.7 The User Interface and the Management in MEDIASTORE

In this work, we developed a user interface for MEDIASTORE using SUIT, the Simple User Interface Toolkit, which is developed by the University of Virginia. SUIT is a subroutine library which helps C programmers create graphical user interfaces that may be modified interactively. SUIT acts as a window manager for screen objects such as buttons, scroll bars, and menus. When SUIT-based programs execute, users may change attributes of the screen objects including an object's location and appearance. The changes to these attributes are then saved with the program [CON92]. In this section, we describe the user interface for further details. Through the introduction of the interface, an overview on the features of the system is provided.

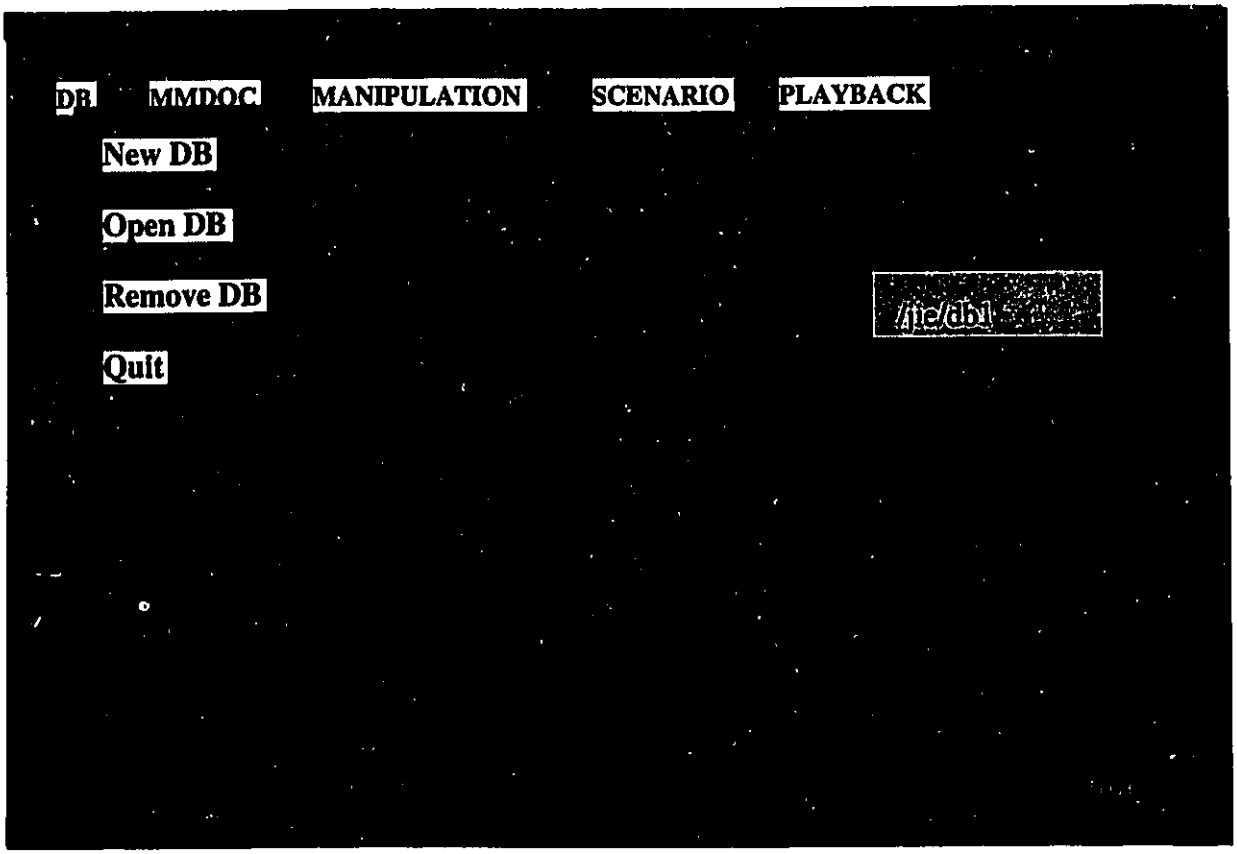

Figure 5.12 UI for Opening a DB

5.7.1 DB Menu

To manage the multimedia information in MEDIASTORE, the first thing is to create or open a database. As shown in Figure 5.12, four functions are provided in the DB menu.

• New DB

The create_database function is invoked by clicking on New DB in the DB menu. This function creates a database with the user specified name, initiate the MMDOC, global, video, audio, text, and graphics extents, assigns database roots to them, initiate the value of the IDs for each of the extents, and assign database roots to them.

• Open DB

The open_database function is invoked by clicking on Open DB in the DB menu. This function opens a database with the user specified name. If the database does not exist, tell the user that the specified database is not found.

After the database creating or opening, the database name is assigned to the global variable MMDB as its value.

• Remove DB

The remove_database function is invoked by clicking on Remove DB in the DB menu. This function deletes a database with the user specified name.

• Quit

The close_database function is invoked by clicking on Quit in the DB menu or Done button at the bottom right corner in the user interface window. This function close the current database and user interface window.

5.7.2 MMDOC Menu

After the database creating or opening, the next thing is to create or open a multimedia document. As shown in Figure 5.13, three functions are provided in the MMDOC menu.

• New MMDOC

When the user clicks on the New MMDOC in the MMDOC menu, a xterm window comes up and the create_document function is invoked in this window. This function is menu-oriented interactive function. It, first at all, creates a MMDOC object as the document root. Then creates the components and link them together to form the document

hierarchy in interactive manner. During the creation, the function clusters the document components in terms of the clustering strategy which clusters all the logical pointers in a document into one segment to improve performance. As well, the audio segmentation is conducted for the synchronization purpose.

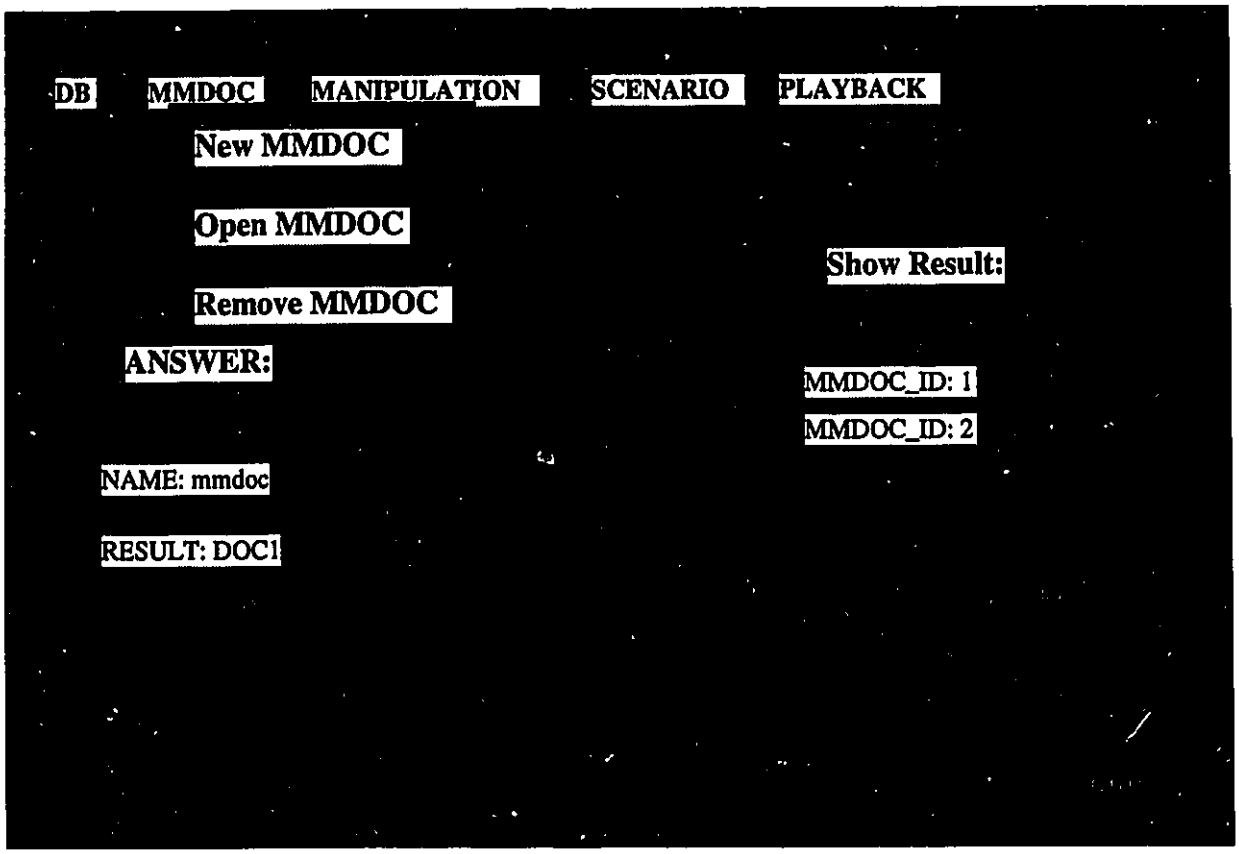

Figure 5.13 UI for Opening a MMDOC

• Open MMDOC

When the user clicks on the Open MMDOC in the MMDOC menu, the Answer window comes up. As shown in Figure 5.13, the user needs to provide the document name and to specify the name to store the retrieved document root pointer. In MEDIASTORE, we assign five global cross transaction variables (i.e., DOC1, DOC2, ..., DOC5) to store the retrieved document root pointers. With these global variables, the user may perform the

cross document operations such as insert the component of the other document as its component. Furthermore, using these global variables we may avoid to retrieve the documents repeatedly.

After the answering, click on the OK button. Then, the open_document function is invoked. This function is to retrieve the specified document over the MMDOC extent. If the document name is not unique in the database, the Show Result window comes up. All the documents whose name is the same as the specified name appear in the window. To choose the document with special MMDOC_ID, click on the entry.

• Remove MMDOC

The remove_document function is invoked by clicking on Remove MMDOC in the MMDOC menu. This function is to remove all the data and links of the document recursively. If the component object is used only in this document, then it is also deleted from the corresponding class extent, otherwise, only the links for connecting to the document are deleted.

5.7.3 Manipulation Menu

To modify the existing documents, basic functions are needed to manipulate the document components. As shown in Figure 5.14, five functions are provided in the Manipulation Menu.

• Navigative Retrieval

The purpose of the Navigative Retrieval is to find out the component from the specified document. When the user clicks on the Navigative Retrieval in the Manipulation menu, the Answer window comes up. As shown in Figure 5.14, the user needs to provide the document pointer, the component name and to specify the name to store the retrieved

component object pointer. In MEDIASTORE, we assign ten global cross transaction variables (i.e., OBJ1, OBJ2, ..., OBJ10) to store the retrieved component object pointers.

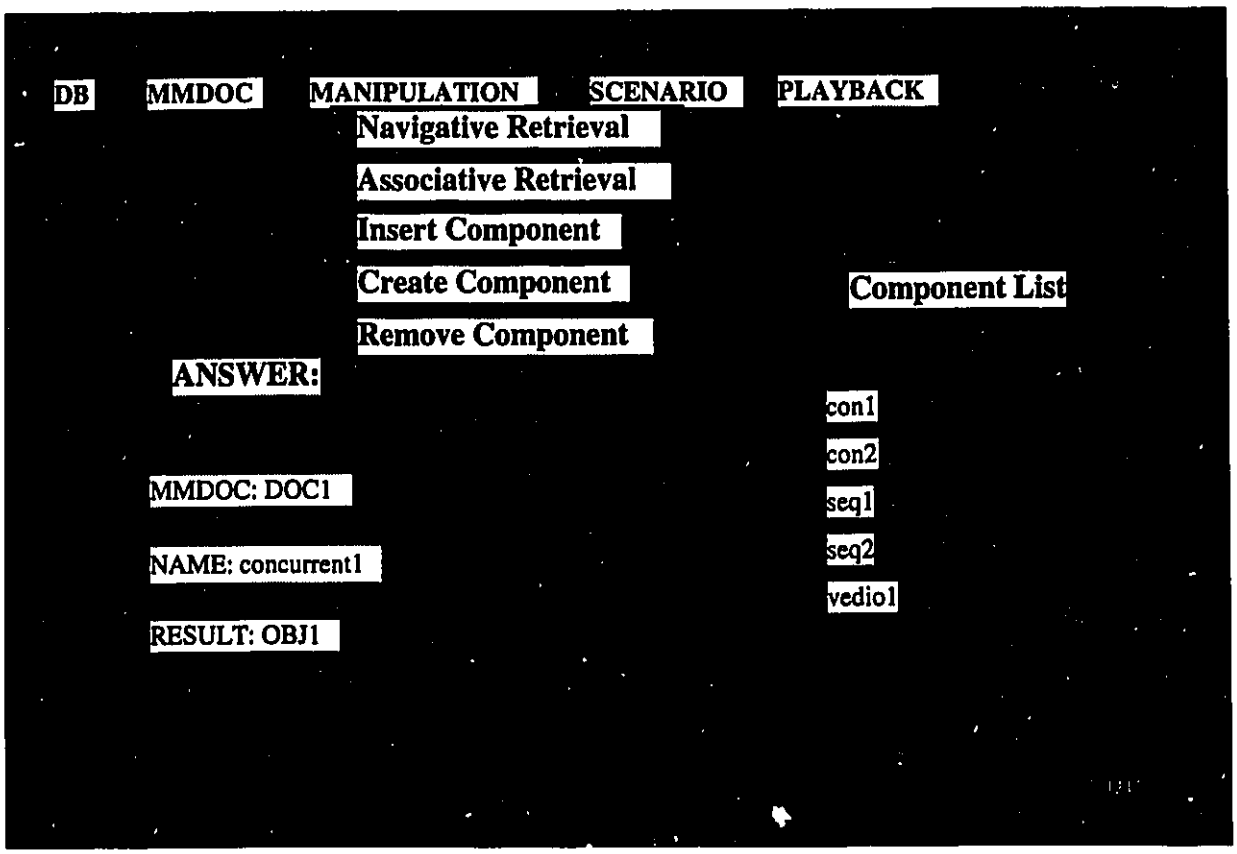

Figure 5.14 UI for Navigative Retrieval

After the answering, click on the OK button. Then, the retrieve_component function is invoked. This function starts the search from the document root and then its children through the document hierarchy. The search is based on the values of the common attribute across the diverse multimedia objects in the document. The typical common attributes used for the navigative query through the document structure are Name, and ID. If the component is not found, the Component List window comes up with a list of all the component names. To choose the component, click on the entry.

• Associative Retrieval

The purpose of the Associative Retrieval is to find out the object from the specified class extent. When the user clicks on the Associative Retrieval in the Manipulation menu, the Answer window comes up. As shown in Figure 5.15, the user needs to provide the extent type, the object name and to specify the result name to store the retrieved object pointer.

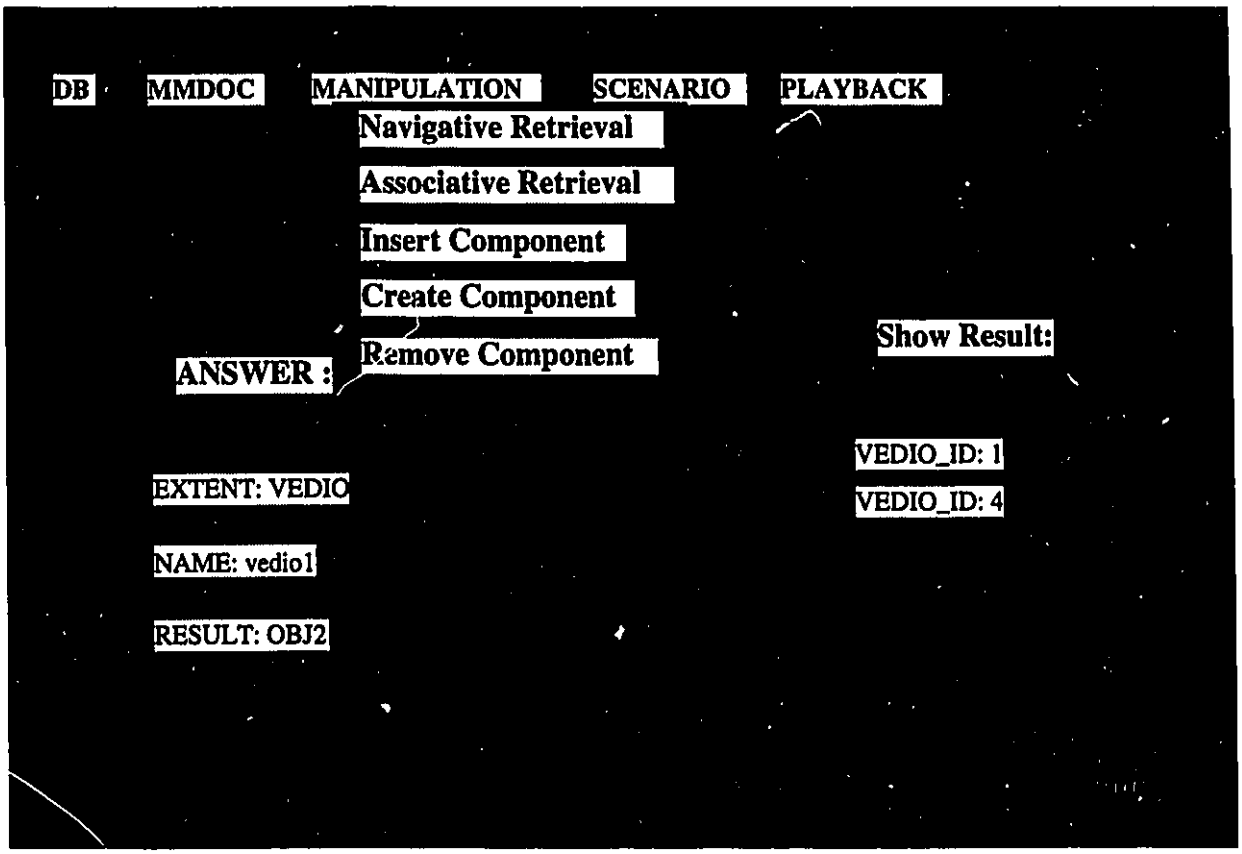

Figure 5.15 UI for Associative Retrieval

After the answering, click on the OK button. Then, the retrieve_extent function is invoked. This function retrieves the object over the specified class extent. If the object

name is not unique in the class extent, the Show Result window comes up. All the objects whose name is the same as the specified name appear in the window. To choose the object, click on the entry.

• Insert Component

When the user clicks on the Insert Component in the Manipulation menu, the Answer window comes up. As shown in Figure 5.16, the user needs to provide the father pointer, the child pointer, and the position which is the sequence number of the child in the children list of the father object. The father should be a component of a document. However, the child can be either a component or a new object.

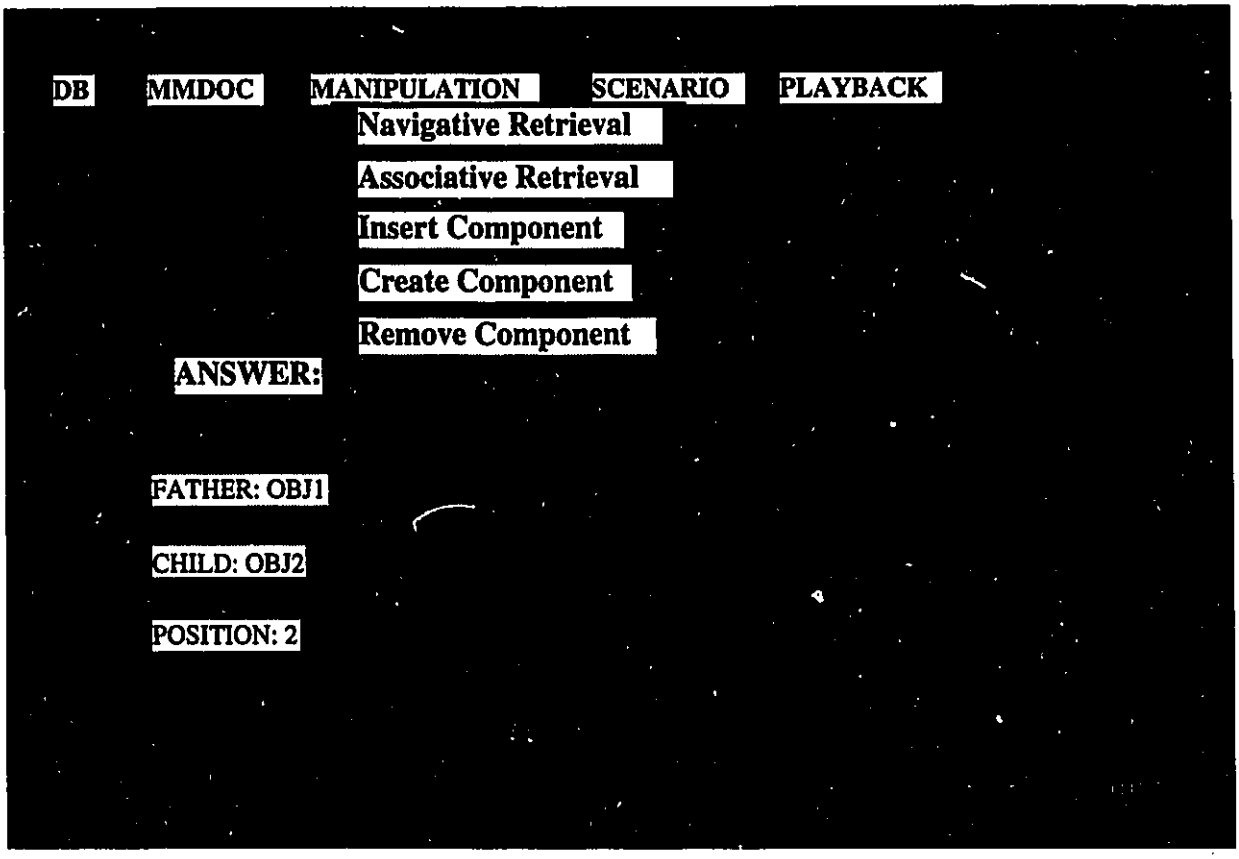

Figure 5.16 UI for Component Insert

After the answering, click on the OK button. Then, the insert_component function is invoked. This function inserts the child object into the children list of the father object.

• Create Component

When the user clicks on the Create Component in the Manipulation menu, a xterm window comes up and the create_component function is invoked in this window. This function is similar to the create_document function. It is also a menu-oriented interactive function. It may create one object or a hierarchy in interactive manner.

• Remove Component

The remove_component function is invoked by clicking on Remove Component in the Manipulation menu. This function is to remove the component and its children. The strategy is similar to that of the remove_document function. However, this function starts the removal from the component object instead of the root of the document.

5.7.4 Scenario Menu

The document scenario and synchronization management is the preparation for the synchronized playback. The management is based on the temporal stream objects list which is a list of all the temporal stream objects in the document. The list is extracted during the document creation and stored in the MMDOC object as the value of temporal_stream_list attribute. As shown in Figure 5.17, two functions are provided in the Scenario menu.

• Scenario Setup

The scenario_setup function is invoked by clicking on Scenario Setup in the Scenario menu. This function is to assign the values of the start time, stop time, and

intermission delay of each temporal stream object within the temporal stream object list. Then, propagate the scenario to the whole document hierarchy by calling the scenario_propagation function.

• Scenario Adjust

The scenario_adjust function is invoked by clicking on Scenario Adjust in the Scenario menu. This function is to adjust the value of the start time of the specified temporal stream object which have been retrieved. The function first changes the start time value of the object and then propagates the scenario to the whole document hierarchy by calling the scenario_propagation function.

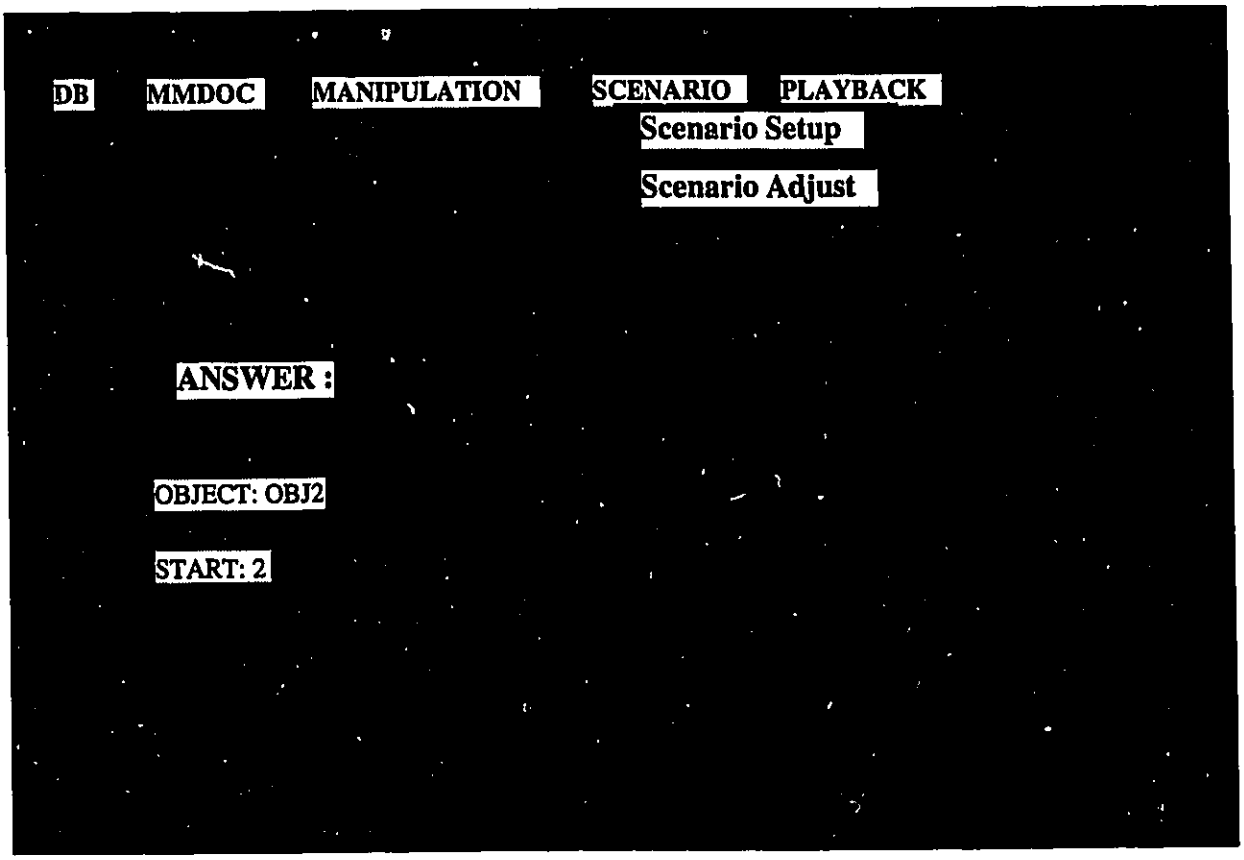

Figure 5.17 UI for Scenario Adjustment

The scenario_adjust function is extremely useful for synchronizing the video and the corresponding audio stream. The user may playback the portion of the document which includes the video and the related audio stream. If the audio does not match the video perfectly, the user can change the start time of the audio stream to fix the mismatch problem.

• scenario propagation

The scenario_propagation function is used in both the scenario_setup and the scenario_adjust function. This function propagates the scenario from temporal stream object list up to all the composite objects at the higher levels. In other words, it calculates and assigns the values of the start and stop time for every component object in the document. The propagation is needed before the document playback since the playback function is based on the time values of each component object.

5.7.5 Playback Menu

One of the important purposes of the system is to retrieve the multimedia document in synchronized mode for the playback. As shown in Figure 5.18, the playback function is provided in the Playback menu.

• Playback

The purpose of the Playback is to retrieve the specified object in synchronized mode for playback. The object may be a document or a component of document. When the user clicks on the Playback in the Playback menu, the Answer window comes up. As shown in Figure 5.18, the user needs to provide the object pointer, which have been retrieved and set as the value of the global variables, as the start point for the synchronized retrieval.

After the answering, click on the OK button. Then, the playback function is invoked. This function is to retrieve the specified document or the component of the document in synchronized mode for playback. As we introduced in previous sections, the function retrieves the object piece by piece in terms of the time values. If the specified object is a document pointer, the function invokes the synchronized_retrieval function from the root of the document. Otherwise, the function invokes the synchronized_retrieval function from the specified object.

In fact, whether to start retrieval from a document root or a component of the document makes no difference for the user. The object-oriented approach allows the user to treat a document or a part of the document in the same way. This is one of the important reasons to use object-oriented database for our system.

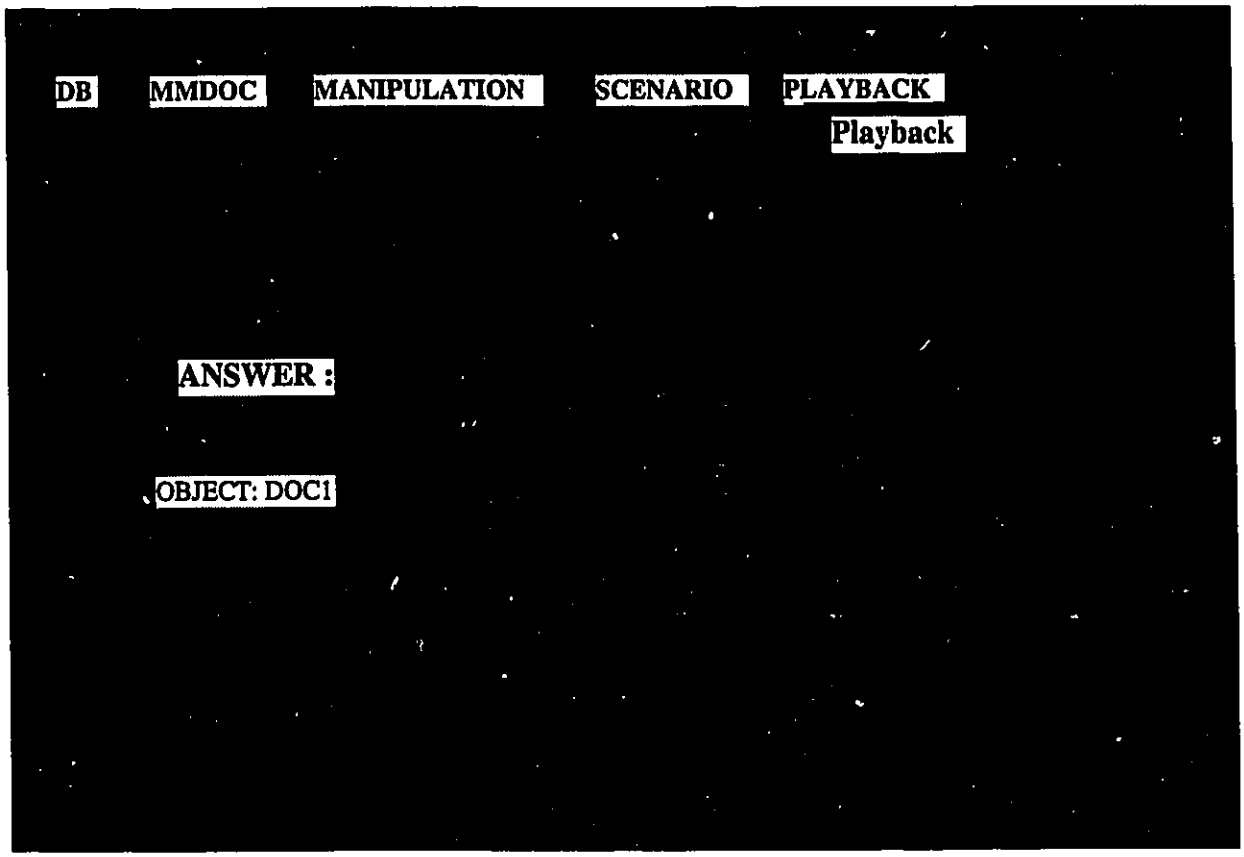

Figure 5.18 UI for Playback

Through the introduction of the user interface and the functions for the management of the system, we can see that the system solves most of the problems while complying with our initial goals, which were to make a multimedia object-oriented storage system for retrieving multimedia document from database in synchronized mode, and providing database features for storage, and manipulations of synchronized objects within the multimedia documents.

5.8 Summary

In this chapter, the design and implementation of the MEDIASTORE is presented. We have first introduced MEDIADOC which is the multimedia document architecture partially supported in MEDIASTORE. Secondly, we presented our object-oriented multimedia database system MEDIASTORE and its multimedia database schema for constructing and managing a multimedia document. The database schema is composed of five level classes (i.e., MMDOC, composite, temporal_stream, media_stream, and media_element classes). We explicitly represent the synchronization relationship between the different media streams by two types of objects (i.e., concurrent or sequential objects) in a multimedia document. A multimedia document is a vehicle to achieve the synchronization control between its component multimedia objects. Then the most important part of our work, the synchronized retrieval algorithm of multimedia document for playback is presented. The retrieval algorithm supports continuity control and multiple synchronization control methods. The data sharing, playback and layout, and queries aspects have also been discussed. Finally, the design and implement of the graphical user interface for MEDIASTORE have been described.

Although MEDIASTORE does not have *all* the features which we would like it to have, in order for it to be a true multimedia object-oriented database system, it does provide

a common multimedia database framework for the management of real-time multimedia information.

Chapter 6

Conclusions

In this thesis, we have investigated the areas of multimedia and object-oriented databases in order to make use of the object-oriented database technology for real time multimedia applications. MEDIASTORE is developed as part of an object-oriented database capable of handling real time multimedia applications. MEDIASTORE offers many features to store and manipulate multimedia data in a multimedia document. A document architecture is used to describe and model complex objects such as audio, video, image, and text. The temporal and spatial relationships between component objects are also described in the document. A synchronized retrieval algorithm for playback of multimedia document is developed over the document architecture. It allows users to playback several related data streams which are put into one document in synchronized mode automatically.

The first part of this thesis covers database concepts, object-oriented database technology, and multimedia requirements. The analysis shows that the relational database systems have many significant shortcomings for multimedia applications, although they are quite successful in managing traditional business applications upon alphanumeric information organized as records. On the other hand, object-oriented database systems are

more suitable for managing multimedia data. It is easier to model real-world entities (in our case, to model document hierarchy) with object-oriented database systems. As well, it greatly enhances the extensibility of the system by reusing the properties of existing classes. We have reviewed the current trends of the research in multimedia databases. There are two directions: extending the relational model, and developing new databases from scratch based on the object-oriented model.

As the first step in designing our own system, the requirements of multimedia applications were analyzed. Multimedia information, as its name says, includes various media types produced from different devices. Each medium has diverse formats since vendors of these devices use different standards for their products. Multimedia data can be quite complex, such as two- and three-dimensional graphical images. Complex relationships between different media may exist. This requires the system to provide the capability to model diverse complex data types and to represent the relationships between them. The system to be built must have extensibility to include new media. The object-oriented model is a perfect tool to meet these requirements. Moreover, since multimedia data are also time-based data, temporal and spatial relationships between different media streams may exist. Thus, capabilities to represent the temporal relationships and manage synchronization between different media streams to be played back simultaneously are required. However, general object-oriented databases do not have the features to deal with these problems. Therefore, a multimedia database system for real-time multimedia applications is needed. This is the motivation for the development of MEDIASTORE.

The design of the MEDIASTORE system focused on the temporal relationship representation and synchronization management for real-time multimedia applications. Our strategy is to use the document architecture as our database schema to organize and manage the related multimedia information which is integrated into one single entity called the

multimedia document. It allows us to describe, organize and structure multimedia objects, and represent their temporal relationships in an integrated and homogeneous method. Based on the architecture, we designed the algorithm to retrieve the document in the synchronized mode. Once the user finds the document and invokes the retrieval method, the whole document would be retrieved automatically in a synchronized manner. Therefore, the document is a vehicle to achieve both integration and synchronization. Moreover, a number of features were developed to create and manipulate a document, as well as to manage the scenario of the document for the playback.

The MEDIASTORE is implemented using C++ and the ObjectStore object-oriented database management system (OODBMS). It gives us full benefit of the features of the underlying OODBMS. The object-oriented data model is perfect to construct a document hierarchy. In the meantime, we do not have to write codes that translate data between the disk resident and the in-memory representations used during execution. This is referred to as single-level storage. It is one of the significant advantages of OODBMS over RDBMS. For example, to store a C++ object in a relational database, the programmer would have to construct a mapping between the two, and write a code that picks fields out of tuples and copies them into data members of objects. This is part of the problem that has been called the "impedance mismatch" between programming languages and database access languages. In addition, the extensibility in object-oriented paradigm makes it an easy task to add a new medium in the system.

At present, our prototype system MEDIASTORE provides basic features for the management of a multimedia document and its components. To refine the system, more features can be added in the future, such as:

- A graphical interface for the scenario (timing schedule) management is necessary. It must be able to handle relationships between related streams such as BEFORE, SAME, and AFTER. Further, it must provide users with the dynamic adjustment capacity that allows them to modify the scenario during the playback.

- The use of C++ and ObjectStore prevents us from adding new classes at run-time, or modifying the classes. For instance, to add a new media, one must change the source code of the system and recompile the source files. The limitation is due to the lack of high-level query language of ObjectStore to define and manipulate classes.

- The capability to control hardware devices for capture, display and process of multimedia data should be provided. Furthermore, the system should offer the facility to identify various formats of multimedia data automatically. For instance, it should be able to identify automatically the possible formats of video data such as JPEG, MPEG, and DVI.

In summary, we have developed a multimedia logical storage system MEDIASTORE for the management of multimedia documents. The design of the multimedia document architecture focused on synchronization management and retrieval. The implementation of this system fulfilled most of the requirements of this study. This is a good basis for further research in the area of multimedia databases.

Appendix

A.1 The Examples of the MMDOC Classes

```
class MMDOC: public Object {
        /* Attributes: */
                        int             MMDOC_ID;
                        Object          *MMDOC_root;
                        os_Set<Media*> &temporal_stream_list;
                        segment         *MMDOC_segment;
                        ...             ...
        /* Methods: */
        /* Document:*/
                        void            remove_document ();
        /*Component:*/
                        Object*         retrieve_component(char*,char*);
        /*Scenario:*/
                        void            scenario_setup ();
                        void            scenario_propagation ();
        /*Synchronized Retrieval*/
```

```
                    void            synchronized_retrieval();
        /*Others:*/
                    int             get_MMDOC_ID();
                                    MMDOC ();
                                    ~MMDOC ();
                }

class Object {
                    int     global_ID;
                    char    *name;
                    char    *object_type;
                    . . .
                    Object  *parent ;
                    List<Object*>           *children;
                    . . .
                    Object* navigative_retrieval (char*,char*);
                    void    insert_child(Object*,int);
                    void    remove();
                    int     get_global_ID();
                            Object(char*);
                            ~Object();
                }

class Temporal_Object : public Object{
                    global_time     start_time;
                    global_time     duration;
                    global_time     stop_time;
```

```
        global_time     intermission_delay;
        global_time     concurrent_global_time;
        real_time       concurrent_real_time;
        . . .
        void    synchronized_retrieval();
        void    init_synchronized_retrieval();
        void    scenario_propagation ();
                Temporal_Object(char*);
                ~Temporal_Object();
        . . .
}
```

B.1 MMDOC Creation Algorithm

Note that the function defined in the classes is called method, otherwise, called function. A temporal object is an object which belongs to a specific document (e.g., MMDOC, composite, concurrent, sequential, or temporal_stream object). A media object is one which is not document specific object (e.g., media_stream, video, audio, text, graphics).

1. Initialize the algorithm.

   open the database with the specified name;

   start a transaction;

   find the roots of the class extents from the database.

   (i.e., *MMDOC_extent, Object_extent, Video_extent, Audio_extent, Text_extent, and Graphics_extent*)

2. Create a MMDOC object on top of the document hierarchy.

   /* call function create_MMDOC(); */

generate a segment to cluster the document hierarchy pointers.

create a MMDOC object, set *name*, *MMDOC_segment* of the MMDOC.

generate *MMDOC_ID*, call method get_MMDOC_ID ().

insert the MMDOC object into the *MMDOC_collection*.

return the MMDOC object pointer *a_MMDOC*.

3. Create a temporal object as *a_temporal_object*.

The temporal object may be a single object or composite object with a hierarchy.

3.1 Create temporal object.

/* call function create_temporal_object (*a_MMDOC*); */

Case(sequential_object): /* call function create_sequential (*a_MMDOC*); */

create a Sequential object as *a_sequential*;

cluster the object to *MMDOC_segment;*

set *name* of the Sequential object;

generate *global_ID*, call method get_global_ID ();

create children objects;

/* call function create_children (*a_MMDOC, a_sequential*); */

return the Sequential object pointer *a_sequential*.

Case(concurrent_object): /* call function create_concurrent (*a_MMDOC*); */

create a Concurrent object as *a_concurrent*);

cluster the object to *MMDOC_segment;*

set *name* of the Concurrent object;

generate *global_ID*, call method get_global_ID ();

create children objects;

/* call function create_children (*a_MMDOC, a_concurrent*); */

return the Concurrent object pointer *a_concurrent*.

Case(temporal_stream): /* call function create_ temporal_stream(*a_MMDOC*); */

create a temporal_stream object as *a_temporal_stream*;

cluster the object to *MMDOC_segment;*

set *name* of the temporal_stream object;

set *start_time, stop_time, intermission_delay* of the object;

generate *global_ID*, call method get_global_ID ();

insert the temporal_stream object into

a_MMDOC->temporal_stream_list;

query original media stream object;

if not found then

create original media stream object;

/* call function create_media_stream (*a_temporal_stream*); */

set *a_temporal_stream-> original_stream* =

create_media_stream (*a_temporal_stream*);

derive its children objects;

/* call derivation functions: Copy_Of(), Part_Of(), or Scale_Of(). */

return the temporal_stream object pointer

a_temporal_stream.;

Case(quit): return 0;

3.2 Create children /* call function create_children (*a_MMDOC, a_composite*)*/

The function is to create child objects and link the children to the father. As the result of the recursive invocation of function create_temporal_object () and create_children (), the document hierarchy is built. Note that only the composite object (e.g., the sequential or the concurrent object) has children.

Case(more_children): create a temporal object *a_temporal_object,* call function

a_temporal_object = create_temporal_object(*a_MMDOC*);

insert the object into the father's children list,

a_composite ->children.insert (*a_temporal_object*);

Case(no_more_children): return 0;

3.3 Create original media stream object

/* call function create_media_stream (*a_temporal_stream*); */

Case(video_object): /* call function create_video (*a_temporal_stream*); */

create a Video object as *a_video*;

set *name* of the Video object;

set *filename* of the Video data;

set *width, height, num_frames* of the Video object;

generate *video_segment* to cluster the video_frame objects;

set *duration, rate* of the Video object;

generate *global_ID*, call method get_global_ID ();

generate *video_ID*, call method get_video_ID ();

generate *video_segment* to cluster the video data;

insert the Video object into the *Video_collection*;

create video_frame objects as its children;

/* call function create_video_frame (*a_video*); */

return the Video object pointer *a_video*.

Case(audio_object): /* call function create_audio (*a_temporal_stream*); */

create a Audio object as *a_audio*;

set *name* of the Audio object;

set *filename* of the Audio data;

set *audio_size, audio_sample_rate,*

audio_sample_resolution of the Audio data file;

set *video_rate* of the corresponding Video object;

calculate *audio_segment_size, segment_num* of the Audio;

generate *audio_segment* to cluster the audio_segments;

set *duration* of the Audio object;

generate *global_ID*, call method get_global_ID ();

generate *audio_ID*, call method get_audio_ID ();

insert the Audio object into the *Audio_collection*;

divide the Audio into audio_segments as its children;

/* call function create_audio_segment (*a_audio*); */

return the Audio object pointer *a_audio*.

Case(text_object): /* call function create_text (*a_MMDOC*); */

create a Text object as *a_text*;

set *name* of the Text object;

set *filename* of the Text data;

set *text_size, text_segment_size* of the Text data file;

calculate *segment_num* of the Text;

generate *text_segment_size* to cluster the text_segments ;

set *duration* of the Text object;

generate *global_ID*, call method get_global_ID ();

generate *text_ID*, call method get_text_ID ();

insert the Text object into the *Text_collection*;

divide the Text into text_segments as its children;

/* call function create_Text_Segment (*a_text*); */

return the Text object pointer *a_text*.

Case(graphics_object): /* call function create_graphics (*a_MMDOC*); */

create a Graphics object as *a_text*;

set *name* of the Graphics object;

set *filename* of the Graphics data;

set *width, height* of the Graphics object;

set *duration* of the Graphics object;

generate *global_ID*, call method get_global_ID ();

generate *graphics_ID*, call method get_graphics_ID ();

store the Graphics *data*,

call method getimage (*filename, data, width, height*);

insert the Graphics object into the *Graphics_collection*;

return the Graphics object pointer *a_graphics*.

3.4 Create video frames /* call function create_video_frame (*a_video*); */

The function is to create a sequence of video frames in a loop.

generate the filename of the Video Frame data (e.g., *filename1);*

create a video frame object as *a_video_frame;*

set *name* of the Video Frame object;

generate *global_ID*, call method get_global_ID ();

allocate disk space for the Video Frame *data* in the *a_video->video_segment*;

store the Video Frame *data*, call method getimage (*filename, data, width, height*);

insert the Video Frame object into the Video's children list,

a_video->children.insert(a_video_frame);

repeat the above steps for all the video frames of this Video stream.

3.5 Create audio segments /* call function create_audio_segment

(*a_audio*); */

The function is to create a sequence of audio segments in a loop.

create an Audio Segment object as *a_audio_segment;*

set *name* of the Audio Segment object;

set *seq_no* of the Audio Segment object;

generate *global_ID*, call method get_global_ID ();

allocate disk space (i.e., the *audio_segment_size*)

for the Audio Segment *data* in the *a_audio->audio_segment*;

store the Audio Segment *data*,

call method get_data (*filename, data, audio_size, audio_segment_size, seq_no)*;

insert the Audio Segment object into the Audio's children list,

a_audio->children.insert(a_audio_segment);

repeat the above steps for all the audio segments of this Audio stream.

3.6 Create text segments /* call function create_text_segment (*a_text*); */

The function is to create a sequence of text segments in a loop.

create an Text Segment object as *a_text_segment;*

set *name* of the Text Segment object;

set *seq_no* of the Text Segment object;

generate *global_ID*, call method get_global_ID ();

allocate disk space (i.e., the *text_segment_size*) for the Text Segment *data*

in the *a_text ->text_segment*;

store the Text Segment *data*,

call method get_data (*filename, data, text_size, text_segment_size, seq_no)*;

insert the Text Segment object into the Text 's children list,

a_text->children.insert(a_text_segment);

repeat the above steps for all the text segments of this Text stream.

3. Link the Object to the MMDOC root, *a_MMDOC->MMDOC_root = a_temporal_object.*

4. End the transaction the close the datebase.

B.2 Navigative Query Algorithm

The input data include the document pointer (e.g., DOC1, ... DOC5), the attribute name of the navigative query (e.g., Name, Global_ID), the value of the attribute (e.g., concurrent1) and the result pointer name (e.g., OBJ1, ... OBJ10).

1. Invoke the navigative query function from the User Interface.

```
/* call function navigative_query (Doc_Pointer, Attribute, Attribute_Value,Result_Pointer);*/
        open the database with the specified name;
        start a transaction;
        set a_MMDOC= Doc_Pointer;
        call method retrieve_component () from the document object,
                Result_Pointer=a_MMDOC->retrieve_component (Attribute, Attribute_Value);
        end the transaction;
        close the database;
```

2. Verify query condition

```
        /*call method retrieval_component (Attribute, Attribute_Value)
                                        from a_MMDOC object. */
        if (Attribute=="Name" or Attribute=="Global_ID")
        then    {
                        return  MMDOC_root->navigative_retrieval (Attribute, Attribute_Value);
                }
```

```
else    {
        printf ("The Attribute %s not found. \n", Attribute);
        }
```

3. Navigative Retrieval Algorithm:

Every component object has one recursive method defined in its class for the navigative retrieval: navigative_retrieval (Attribute, Attribute_Value).

```
set not_found=1;
if (Attribute=="Global_ID")
then    {       convert Attribute_Value from char* to int;      }
if ((Attribute=="Name" and name == Attribute_Value) or
        (Attribute=="Global_ID" and global_ID== Attribute_Value_int))
then    { return this;  }
else    {
        for (child=children.first(); child and not_found; child=children.next())
                {
                result_pointer = child->navigative_retrieval (Attribute, Attribute_Value);
                if (found the object, i.e., result_pointer <>nil)
                then    {
                        set not_found=0;
                        }
                }
        return result_pointer;
        }
```

B.3 Scenario Propagation Algorithm

Assume the start_time and stop_time of every temporal stream object in the document has been set.

1. Invoke the method of scenario propagation from the document object.

```
/* call function scenario_propagation (Doc_Pointer);*/
open the database with the specified name;
start a transaction;
set a_MMDOC= Doc_Pointer;
call method scenario_propagation () from the document object,
        a_MMDOC->MMDOC_root->scenario_propagation ();
end the transaction;
close the database;
```

2. The scenario propagation algorithm:

Every component object has one recursive method defined in its class for the scenario propagation: scenario_propagation ().

```
Case(sequential):   /* call method scenario_propagation () from sequential object */
                    for (child= children.first(); child; child=children.next())
                    {
                    child->scenario_propagation ();
                    if (the current child is the last child, i.e., !children.next())
                    then    {       set stop_time = child->stop_time;       }
                    if (the current child is the first child, i.e., child== children.first())
                    then    {
                            set start_time = child->start_time;
                            set start_time1=start_time ;
```

```
                set stop_time1=child->stop_time;
                if (start_time1> stop_time1)
                then    {
                  printf ("object %s scenario error 1\n", child->name);
                        }
                }
        else    {
                set start_time2=child->start_time;
                set stop_time2=child->stop_time;
                if (start_time2> stop_time2)
                then    {
                  printf ("object %s scenario error 1\n", child->name);
                        }
                if (stop_time1> start_time2)
                then    {
                  printf ("object %s scenario error 2\n", child->name);
                        }
                set start_time1=start_time2;
                set stop_time1=stop_time2;
                }
        }

Case(concurrent):   /* call method scenario_propagation () from concurrent object */
                    for (child= children.first(); child; child=children.next())
                    {
                    child->scenario_propagation ();
                    if (the current child is the first child, i.e., child== children.first())
```

```
then    {
        set start_time1 = child->start_time;
        set stop_time1=child->stop_time;
        if (start_time1> stop_time1)
        then    {
          printf ("object %s scenario error 1\n", child->name);
                }
        }
else    {
        set start_time2=child->start_time;
        set stop_time2=child->stop_time;
        if (start_time2> stop_time2)
        then    {
          printf ("object %s scenario error 1\n", child->name);
                }
        if (start_time1 > start_time2)
        then    {
                set start_time1=start_time2;
                }
        if (stop_time1 < stop_time2)
        then    {
                set stop_time1=stop_time2;
                }
        }
}
set start_time=start_time1;
set stop_time=stop_time1;
```

References

[ANS86] American National Standards Institute, "The Database Language SQL", Technical Report, ANSI, 1986, Document ANSI X3.135.

[BER90] P. B. Berra et al., "Architecture for Distributed Multimedia Database Systems", *Computer Communications*, Vol. 13, No. 4, May 1990, pp. 217-232.

[BOW94] C. Bowman, "Why We Need Object-Oriented Systems", *Database Programming & Design*, February 1994, pp. 27-30.

[BRE92] C. Breiteneder, S. Gibbs, and D. Tsichritzis, "Modeling of Audio/Video Data", *Object Frameworks, Proc. 11th International Conference on the Entity-Relationship Approach*, October 1992, pp. 322-339.

[COD70] E. Codd, "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.

[COD72] E. Codd, "Further Normalization of the Data Base Relational Model", *Data Base Systems,* Vol. 6 of Current Computer Science Symposia Series, Prentice-Hall, Englewood Cliffs, N. J., 1972.

[COD74] E. Codd, "A Relational Model of Data for Large Shared Data Banks", *Proceedings of the IFIP Congress,* August 1974.

[COE92] A. Coen-Porisini et al., "Recent Investigations into Relational Data Base Systems", *Proc. 2nd Far-East Workshop on Future Database Systems,* 1992, pp. 28-37.

[CON92] M. Conway, "SUIT Reference Manual", University of Virginia, 1992.

[DAD89] P. Dadam, and V. Linnemann, "Advanced Information Management (AIM): Advanced database technology for integrated applications", *IBM Systems Journal,* Vol. 28, No. 4, 1989, pp. 661-681.

[EHL94] L. Ehley et al., "Evaluation of Multimedia Synchronization Techniques", *Proc. of the International Conference on Multimedia Computing and Systems,* May 1994, pp. 514-519.

[EME93a] J. Emery, and A. Karmouch, "A Multimedia Document Architecture and Rendering Synchronization Scheme", *Proc. of 2nd International Conference on Broadband,* Athens, June 1993, pp. 59-67.

[EME93b] J. Emery, and A. Karmouch, "A Document Model for Continuous Media", *Proc. Canadian Conference on Electrical and Computer Engineering,* Vancouver, September 1993, pp. 640-643.

[EME93c] J. Emery, and A. Karmouch, "A Document Model for Multimedia Information", *Proc. of the IBM 1993 CAS Conference,* Toronto, October 1993, pp. 715-728.

[FIS86] D. Fishman et al., "IRIS: An Object-Oriented DBMS", *ACM Transactions on Office Information Systems,* Vol. 4, No. 2, April 1986.

[GIB91a] S. Gibbs, "Composite Multimedia and Active Objects", *Proc. OOPSLA'91,* 1991, pp. 97-112.

[GIB91b] S. Gibbs, L. Dami, D. Tsichritzis, "An Object-Oriented Framework for Multimedia Composition and Synchronisation", *Multimedia - Principles, Systems and Applications*, ed. L. Kjelldahl, Springer, 1991.

[GIB93] S. Gibbs, C. Breiteneder, and D. Tsichritzis, "Audio/Video Databases: An Object-oriented Approach", *Proc. 9th International Conference on Data Engineering*, April 1993, pp. 381-390.

[GRE92] J. Green, "The Evolution of DVI System Software", *Commun. of the ACM*, Vol. 35, No. 1, Jan. 1992, pp. 53-67.

[ISO88] ISO, Information Processing - Text and Office Systems. Office Document Architecture (ODA) and Interchange Format (ODIF), Parts 1-8, International Standard 8613, March 1988.

[KAR90] A. Karmouch et al., "A Multimedia Medical Communications System", *IEEE JSAC*, Vol. 8, No. 3, April 1990, pp. 325-339.

[KAR93a] A. Karmouch, "Multimedia Distributed Cooperative System", *Computer Communication*, special issue on Group Communications, Vol. 16, No. 9, September 1993, pp. 568-580.

[KAR93b] A. Karmouch, "A Multimedia Information and Communications System: MEDIABASE", *Proc. ICCC Multimedia Communications '93 Conference*, Alberta, April 1993, pp. 287-296.

[KAR93c] A. Karmouch, "Multimedia Databases and Distributed Systems", *Multimedia Communications Research Laboratory Report of Activities: 1992-1993*, Dept. EE, University of Ottawa, pp. 27-33.

[KIM90] W. Kim, "Introduction to Object-oriented Databases", MIT, 1990.

[KLA90] W. Klas, E. J. Neuhild, and M. Schrefl, "Using an Object-oriented Approach to Model Multimedia Data", *Computer Communications*, Vol. 13, No. 4, May 1990, pp. 204-216.

[LEG91] D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications", *Commun. of the ACM*, Vol. 34, No. 4, April. 1991, pp. 46-58.

[LIL92] L. Li, A. Karmouch, and N. D. Georganas, "Synchronization in Real-time Multimedia Data Delivery", *Proc. IEEE ICC'92*, Chicago, pp. 587-591.

[LIL93a] L. Li, A. Karmouch, and N. D. Georganas, "Real-time Synchronization Control in Multimedia Distributed Systems", *ACM Computer Communication Review*, Vol. 22, No. 3, 1993, pp. 79-86.

[LIL93b] L. Li, A. Karmouch, and N. D. Georganas, "Performance Modeling of Distributed Data Integration in Real-time Multimedia Systems". *Proc. ICCC Multimedia Communications'93*, Banff, Canada, April, 1993.

[LIL93c] L. Li, L. Lamont, A. Karmouch, and N. D. Georganas, "A Distributed Synchronization Control Scheme in a Group-oriented Conferencing System", *Proc. of 2nd International Conference on Broadband Islands*, Athens, Greece, June 1993.

[LIT90a] T. Little, and A. Ghafoor, "Multimedia Object Models for Synchronization and Databases ", *Proc. IEEE Int. Conf. on Data Engineering*, 1990, pp. 20-27.

[LIT90b] T. Little, and A. Ghafoor, "Synchronization and Storage Models for Multimedia Object", *IEEE JSAC*, Vol. 8, No. 3, April 1990, pp. 413-427.

[LIT92] T. Little, and A. Ghafoor, "Scheduling of Bandwidth-Constrained Multimedia Traffic", *Computer Communications*, Vol. 15, July/August 1992, pp. 381-387.

[LIT93a] T. Little et al., "A Digital On-Demand Video Service Supporting Content-Based Queries", *ACM Multimedia*, June 1993, pp. 427-436.

[LIT93b] T. Little, and A. Ghafoor, "Interval-Based Conceptual Models for Time-Dependent Multimedia Data", *IEEE Transactions on Knowledge and Data Engineering* , Vol. 5, No. 4, August 1993, pp. 551-562.

[LUT91] A. Luther, *Digital Video in the PC Environment*, McGraw-Hill, New York, 1991.

[MAS87] Y. Masunage, "Multimedia Databases: A Formal Framework", *Proc. IEEE Computer Society Office Automation Sympo.*, 1987, pp. 36-45.

[MAS89] Y. Masunage, "An Object-Oriented Approach to Multimedia Database Organization and Management", *Proc. Int. Sympo. on Database Systems for Advanced Applications (DASFAA)*, 1989, pp. 190-200.

[MAY93] M. Maybury, "Introduction", *Intelligent Multimedia Interfaces*, Edited by M. Maybury, MIT, 1993.

[MEG91] C. Meghini, "Conceptual Modeling of Multimedia Documents", *Computer* , October 1990, pp. 23-28.

[MEY92] V. de Mey, C. Breiteneder, S. Gibbs, and D. Tsichritzis, "Visual Composition and Multimedia", ed. D. Tsichritzis, Centre Universitaire d'Informatique, Universite de Geneve, July 1992, pp. 243-257.

[MEY93] V. de Mey, and S. Gibbs, "A Multimedia Component Kit", *ACM Multimedia 93*, June 1993, pp. 291-300.

[NAF90] N. Naffah, "Multimedia Applications", *Computer Communications*, Vol. 13, No. 4, May 1990, pp. 243-249.

[OBJ93a] "ObjectStore Technical Overview: Release 2.0.", 1993, pp. 3-27.

[OBJ93b] "ObjectStore User Guide: Release 2.0." 1993, pp. 1-18.

[OOM93] E. Oomoto, and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, August 1993, pp. 619-628.

[RAM89] C. V. Ramamoorthy et al., "Object-Oriented Approach: Systems, Programming, Languages, and Applications", *IEEE Expert*, Vol. 3, No. 3, 1988, pp. 9-15.

[RAM93] K. Ramamritham, "Real-Time Databases", Distributed and Parallel Databases, Vol. 1, 1993, pp. 199-226.

[ROT93] S. Roth, and W. Hefley, "Intelligent Multimedia Presentation Systems: Research and Principles", *Intelligent Multimedia Interfaces*, Edited by M. Maybury, MIT, 1993.

[SHE90] T. Shetler, "Birth of the BLOB", *BYTE*, February 1990, pp. 221-226.

[STE93] D. R. Stephen, "C++ Programmer's Companion: Designing, Testing, and Debugging", Addison-Weslay Publishing Company, 1993, pp. 8-10.

[STO86] M. Stonebraker, editor, *The INGRES Papers*, Addison-Weslay, 1986.

[WAL91] G. Wallace, "The JPEG Still Picture Compression Standard", *Commun. of the ACM*, Vol. 34, No. 4, April. 1991, pp. 30-44.

[WAN93] R. Wang, and A. Karmouch, "A Multimedia File Structure for Continuous and Discrete Media", *1993 Canadian Conf. on Electrical and Computer Engineering*, Vancouver, pp. 644-647.

[WIE88] R. Wiener, "An Introduction to Object-Oriented Programming and C++", Addison-Weslay Publishing Company, 1988.

[WIL94] N. Williams, and G. S. Blair, "Distributed Multimedia Applications: A Review", *Computer Communications*, Vol. 17, No. 2, February 1994, pp. 119-132.

[WIN90] A. Winblad, S. Edwards, and D. King, "Object-oriented Software", Addison-Wesley, 1990, pp. 36.

[WOE86] D. Woelk, W. Kim, and W. Luther, "An Object-Oriented Approach to Multimedia Databases", *Proc. of the ACM-SIGMOD 1986 Conf. on Management of Data*, Washington, D. C., May 1986, ed. C. Zaniolo, SIGMOD Record, Vol. 15, No. 2, pp. 311-325.

[WOE87a] D. Woelk, and W. Kim, "Multimedia Information Management in an Object-Oriented Database System", *Proc. 13th Int. Conf. on VLDB*, Brighton, England, September 1987, eds. P. M. Stocker and W. Kent, Morgan Kaufmann Publishers, Los Altos, CA, 1987, pp. 319-329.

[WOE87b] D. Woelk, W. Luther, and W. Kim, "Multimedia Applications and Database Requirements", *Proc. IEEE Computer Society Office Automation Sympo.*, 1987, pp. 180-189.

[YOO87] B. Yoon, F. Suzuki, H. Ishikawa, and A. Makinouch, "Experimental Multimedia DBMS Using an Object-Oriented Approach", *Proc. IEEE COMPSAC*, 1987, pp. 632-641.