UDC 001.81: 621.397: 681.3

A Prototype Multimedia Database System

●Hiroshi Ishikawa

(Manuscript received May 29,1997)

Emerging multimedia applications such as digital libraries and document management require new databases. Next-generation database systems must enable users to efficiently and flexibly develop and execute such advanced multimedia applications. In this paper, we focus on the development of a database system which enables flexible and efficient acquisition, storage, access, retrieval, distribution, and presentation of large amounts of heterogeneous media data. We take the approach of extending an object-oriented database, which is more suitable for describing media structures and operations than a traditional relational database. In this paper, we describe a multimedia data model and its efficient implementation.

1. Introduction

New multimedia applications emerging on top of information infrastructures include on-demand services (e.g., videos, news, and sports), digital libraries and museums, online shopping, and document management. We need a next-generation industry database system which enables users to efficiently and flexibly develop and execute such advanced multimedia applications.[1] Moreover, since in some cases of application development there is no existing database while in other cases there are databases or files to be integrated, we need to enable both top-down and bottom-up database development. (This is because top down design is unsuitable when there is an existing database.)

To meet these requirements, we focus on developing a database system which enables flexible and efficient acquisition, storage, access, retrieval, distribution, and presentation of large amounts of heterogeneous media data. We take a realistic approach based on a Fujitsu object-oriented database (OODB) product called SymfoWARE/ODB. We decided to use an OODB because OODBs are more suitable for descriptions of media structures and operations than traditional relational databases (RDBs). The main features of our system are outlined below.

1) Multimedia data model

We propose a multimedia data model that is an integration of structural, temporal, spatial, and control functionality. Our extended data model based on agents provides uniform interfaces to heterogeneous media in addition to defining structures and operations specific to such media. The model enables the representation of temporal and spatial relationships and of temporal synchronization and QOS (quality of service) control by extending a scripting language suitable for multimedia application development. That is, we take an object-oriented database approach suitable for description of media structures and operations and extend the object-oriented approach by providing temporal and spatial operators and control of distributed computing and QOS.

2) Flexible acquisition

Our database system allows users to acquire newly produced media data via distributed networks including ATM LANs and the Internet. Moreover, we provide multidatabase functionality by managing metadata (e.g., a data dictionary) of existing data files or heterogeneous databases to give users interoperable access to such files and databases. Our technology includes schema translation between OODBs and RDBs, uniform WWW (World Wide Web) gateways to databases, HTML

(Hyper Text Markup Language) page management by databases, and WWW directory management by databases.

3) Efficient storage

We provide media data management mechanisms which enable efficient storage and access of large amounts of media data. These mechanisms enable users to customize media-specific storage in areas such as indexing, clustering, and buffering. We take an object-oriented approach to resolve heterogeneity in data formats, compressors/decompressors (CODECs), and physical media used for implementation of logical media.

4) Efficient retrieval

To facilitate interactive retrieval of multimedia data, we enable users to flexibly and efficiently access partial data such as sub-streams of videos using temporal information (e.g., temporal intervals), keywords, and other related information. This technique of subsetting a large amount of media data is analogous to RDB views. Efficient processing of partial accesses is facilitated by combining software techniques such as access methods, clustering, and exploiting available hardware such as parallel machines. Also, we provide a light-weight technique based on color information to segment scenes and recognize objects for content retrieval of stream data.

5) Flexible distribution and presentation

We provide the means for flexible distribution and presentation of retrieved multimedia data over distributed networks by executing QOS control and scripts. To this end, we use techniques such as prefetching, caching, synchronization, and distributed processing. In this paper, we focus on a multimedia data model and its implementation.

2. Multimedia data model

2.1 Related work

We propose a multimedia data model which integrates structural, temporal, spatial, and control functionality. (Some individual functionality has been researched in previous work, and we have used the results of that work to assess the individual functionality of our model.)

We take the object-oriented model of a given OODB2)-4) as the basis of our multimedia data model. Structures and operations of each media data are directly defined as properties and methods of objects. The set of media objects is managed by the OODB and retrieved and manipulated by a set-oriented query language. Hypermedia functionality corresponds to navigation of media objects through links. Hypermedia links, static or dynamic, can be implemented as properties or methods of media objects.5) In the Internet hypermedia protocol, called HTTP,6) URLs (Uniform Resource Locators) and operational codes (i.e., methods) are suited to object-orientation. Our model can integrate HTTP servers and OODBs, which enables users to access OODBs with WWW browsers.

There are models7)-10) which support temporal descriptions. However, most of them focus only on temporal functionality and pay little attention to the other functionalities. There is almost no work that provides a substantial spatial description functionality. Our model provides set-oriented operators for associative (or partial) access to an ordered set of internal frames constituting stream media such as video and audio. These set-oriented operators are analogous to relational algebra11) for associative access to a set of records constituting tables. In general, to develop multimedia applications, we need control structures which support features such as concurrency. In our model, control is described as annotations to other functionalities, i.e., structural, temporal, and spatial. Annotated concurrency is advocated by Lohr.12)

Relational database vendors, such as Oracle and Sybase, have announced that they will support multimedia extensions by providing universal database servers. However, we cannot compare our work with such products because detailed specifications of the products are not yet available.

In addition to concurrency control, QOS parameters such as latency and bit rates are specifiable in concurrency annotations to allow real-time execution of stream media. We extend the annotated approach to concurrency to allow for QOS control options. QOS is controlled either by executing methods or by retrieving stored data. An Event-Condition-Action (ECA) paradigm of active databases[13] is applicable; however, it needs to be extended for real-time use. We use techniques such as prefetching and caching. In general, event specification facilitates relative invocation of synchronization or serialization of media objects. In particular, time events enable absolute invocation of media objects at specified times. Temporal relationships such as "before" and "after" are directly described by structural operators of the OODB and hypermedia.

In summary, our multimedia data model is unique in that it allows concurrent object-oriented computing and QOS control for the development of distributed and real-time multimedia applications in addition to set-oriented temporal and spatial operators for associative access. In the rest of this section, we describe our multimedia data model, which is clearly different from other work in that it can subsume and integrate previous work.

2.2 Our data model

1) Multimedia

We think that multimedia data are not just static data, but rather compositions of several types of media data and operations on them. We therefore provide structural, temporal, spatial, and control operations as media composition operators, as described later. In other words, our model has multiple facets and subsumes existing models such as object, temporal, spatial, and agent models. Individual operations are orthogonal to one another. Our model is integrated seamlessly with existing technologies.

Multimedia systems consist of multimedia databases and applications. Multimedia databases consist of a set of media data. Media types include text, graphics, images, and streams. Stream types include audio, video, streamed texts, streamed graphics, and streamed images. Multimedia applications consist of a set of scripts. Basically, a script has an identifier (ID) and temporal and spatial operations on a set of streams with QOS options. A stream has an ID and temporal and spatial operations on a set of frames. A frame has an ID and temporal and spatial operations on frame data.

QOS options are parameters given to the QOS controller. QOS types include latency, jitter, various bit rates and frame rates, resolution, colors, and fonts. QOS is controlled either by executing specified QOS functions or by retrieving stored QOS data. The QOS function takes a stream and a time and gives frame IDs. The QOS data consisting of the time and a frame ID are stored in advance by obtaining them from rehearsal.

To concretely explain the features of our data model, we consider the following multimedia application or script, called Script1, under the assumption that there are multimedia databases containing multiple video streams of the same object.

Script1:

a) retrieves all video streams of the prime minister taken on January 17th, 1995,
b) selects only parts of the retrieved video streams temporally overlapping each other,
c) arranges the selected parts on the same presentation space (i.e., window), and
d) plays the parts in temporal synchronization.

2) Structural Operations

We describe an object-oriented model of an OODB using Jasmine,[2] which is a well-published research prototype version of SymfoWARE/ODB. Note that throughout this paper, Jasmine is not a product name but the code name of a research prototype of Fujitsu Laboratories, Ltd. Objects are a collection of attributes, which are categorized into properties (enumerated attributes) and methods (procedural attributes). Properties are object structures and methods are operations on those objects. Objects are categorized into instances and classes. Instances denote individual data, while classes denote types (i.e., structures) and operations applicable to instances of the class. Instances consist of a collection of attribute names and values. Classes consist of attribute names, definitions, and associated information such as demons. Objects are identified by values of the system-defined attribute object identifier (OID). Therefore, objects with the same object identifier in a consistent database have the same values. On the other hand, while values such as numbers and character strings have no OIDs, they do have system-defined classes. Objects with OIDs are called reference objects, and values with no OIDs are called immediate objects. Objects can include other objects (i.e., OIDs) as attribute values. A property containing an OID is a relationship or link implementing a hypermedia link. This enables the user to directly define complex objects (composite objects).[14] Objects can have a set of objects or just a single object as an attribute value. The former are called multiple-valued attributes, and the latter are called singleton-valued attributes.

Classes are organized into a hierarchy (more strictly, a lattice) by generalization relationships. This hierarchy is called a class hierarchy. A superclass in a class hierarchy is denoted by the Super system-defined attribute. Classes (i.e., subclasses) can inherit attribute definitions from their superclasses. Unlike the features provided for in Smalltalk-80,[15] the user can make instances (i.e., instantiate) from any class in a class hierarchy. Such instances are called intrinsic instances of the class. If a property type is a superclass, the property can contain objects of the subclasses.

For example, media objects such as streams and frames are defined (see **Fig. 1**). In addition to specifying how object types and methods are defined, classes are also interpreted as sets of instances. That is, the instances of a class are the unions of all the intrinsic instances of the class and all its subclasses. This differentiates our OODB from other OODBs such as GemStone,[16] where the user must define separate classes for both types and sets.

```
STREAM                                FRAME
  Super   MEDIA                         Super MEDIA-Data Property
Property                                TIME           Time
  TIME            Internal Time         FRAME-Data   Data
  TIME            Real Time
  SPACE           Internal Space
  SPACE           Real Space
  Set FRAME       Frame
  STRING          Topic

MPEG
  Super   STREAM
Property
  Set MPEG-Frame Frame
Method
  MPEG            AddFrame()
  MPEG            RemoveFrame()
  VOID            Forward()
  VOID            Backward()
```

Fig.1– Definition of media objects.

Property inheritance, method inheritance, and set inclusion are facilitated through super relationships.

Specialized functions, called demons, can be attached to attributes to enable the user to flexibly implement active databases. Hypermedia links written in HTML[6] of the WWW can be interpreted so that access events on links invoke methods (viewers) on data (retrieved hypertexts).

In an OODB, the user manipulates objects by sending messages to objects just as in object-oriented programming languages; this process is called singleton access. The user can assign values to attributes and reference attribute values. An OODB allows set-oriented access in addition to singleton access. Set-oriented access is done through object queries. The basic unit of an object query is the object expression, which is a class name followed by a series of attribute names delimited with periods. Object expressions eliminate most of the need for the equijoin predicates used in an RDB. The user can also specify methods in object expressions. An object query consists of a target and a condition. The target part is a list of object expressions. The condition part is a logical combination of simple conditions that

compares object expressions using comparison operators. For example, the following query retrieves streams of the prime minister taken on January 17th, 1995, which realizes Script1 a):
STREAM.Frame from STREAM where STREAM.RealTime = "01171995" and STREAM.Topic = "the prime minister"

To make application programs, users can combine singleton-access and set-oriented access. An element of a set of objects is assigned to an object variable and is manipulated by sending messages to the object variables. The introduction of object variables reduces impedance mismatch between a programming language and a database language.16) Since users can specify object queries as well as simple manipulation of attributes in methods, views of objects corresponding to relational views can be defined using methods. Since a query on a class returns all the instances of the class and its subclasses, a single OODB query can retrieve what would otherwise take multiple RDB queries. A method is either a program or a query. The result of a method is a value or OID or a set of values or OIDs. Methods can also have side effects. Methods returning OIDs also implement hypermedia links.

By specifying methods in a query, users can retrieve and manipulate objects in a set-oriented manner. If a superclass is specified with a method in a query, methods dedicated to instances of the class and its subclasses can be invoked simultaneously. This facilitates polymorphism17) in a set-oriented manner. Thus, polymorphism, which can invoke dedicated methods by specifying generic messages, is usually applied to a single object at a time. But our language enables simultaneous application of polymorphism to a set of objects. A query can also make new instances from more than one class, like joins in an RDB.

Application programs written for the OODB are precompiled into C programs. During this process, references to attributes are statically resolved to reduce the burden of a dynamic search, allowing the C programs to execute efficiently. A set-oriented query can also be interpreted interactively. Objects are basically persistent since they exist after program execution; however, as in conventional programming languages, users can make temporary objects which exist only during program execution. OODBs usually have many classes, and users can access several databases concurrently or switch between them. Also, OODBs provide basic database facilities such as transaction management and buffer management.

3) Temporal and Spatial Operations

Temporal and spatial data are viewed as universal keys common to any stream media data. Such temporal and spatial relationships structure multimedia data implicitly in contrast to explicit links. We define set-oriented temporal and spatial operators that specify such relationships, which are analogous to relational algebra.11)

Although time is one-dimensional and space is three-dimensional, they have similar characteristics. Real time is the time that passes when streams are recorded in the real world. Internal time is the time required for normal playback of streams. External time is the time needed to play a stream using scripts. Usually, real time is equal to internal time. In the case of high-speed video, real time is shorter than internal time. External time is specified by providing a magnification level for internal time. The default magnification is X1; that is, external time is equal to internal time. In the case of slow play of streams, external time is longer than internal time; in the case of fast play, external time is shorter than internal time. Assuming that S1 and S2 are streams and P is a predicate on frames of a stream, temporal composition of streams is achieved by temporal operators as follows:

a) Tintersection (S1, S2) returns parts of S1 and S2 which temporally intersect.
b) Tdifference (S1, S2) returns a part of S1 which does not temporally intersect with S2.
c) Tunion (S1, S2) returns S1 and S2 ordered in time with possible overlaps.
d) Tselect(S1, P) returns a part of S1 which satisfies P.

e) Tjoin (S1, S2, P) = Tselect(Tunion(S1, S2), P).

f) Tproject(S1, Func) returns the result of Func on S1,

where Func is an operation on frames which may include the spatial operators described below.

Note that internal time of a composite stream is the union of the external time of its component streams. Real time of a composite stream is the union of real time of its component streams. For example, we assume that the query result of Script1 a) is scanned and is individually set to Stream1 and Stream2. To select only parts of Stream1 and Stream2 which temporally overlap one another, which realizes Script1 b), we only need to execute expression Tintersection (Stream1, Stream2) based on the internal time. Here we name the selected parts Stream1' and Stream2' for Stream1 and Stream2, respectively. The schematic explanation of the effect of the expression is presented in **Fig. 2**.

Similarly, space is divided into real space, internal space, and external space. Real space is space occupied by streams in the real world. Internal space is space typically occupied by streams in a presentation. External space is space occupied by streams during actual presentation of scripts. External space is specified by providing a magnification of internal space. By default, external space is equal to internal space. Assuming that F1 and F2 are frames and P is a predicate on pixels of a frame, spatial composition of streams is accomplished by spatial operators as follows:

a) Sintersection (F1, F2) returns parts of F1 and F2 which intersect in space.

b) Sdifference (F1, F2) returns a part of F1 which does not intersect in space with F2.

c) Sunion (F1, F2) returns F1 and F2 merged in space.

d) Sselect(F1, P) returns a part of F1 which satisfies P.

e) Sjoin (F1, F2, P) = Sselect (Sunion (F1, F2), P).

f) Sproject(F1, Func) returns the result of Func on F1, where Func is an operation on pixels.

Note that internal space of a composite stream is the union of the external space of its component streams. Real space of a composite stream is the union of real space of its component streams. For example, to arrange Frame1 of Stream1' and Frame2 of Stream2' on the same window, which realizes Script1 c), we evaluate expression Sunion (Frame1, Frame2) based on the external space, the effect of which is schematically explained in **Fig. 3**.

4) Control Operators

Processes, called agents, represent control structures. Processes consist of events, conditions, and actions. Event specification allows for serial, parallel, and alternative occurrences of component

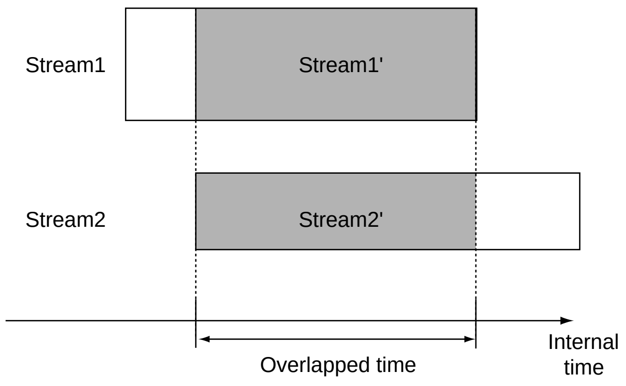

Fig.2– Schematic explanation of expression Tintersection (Stream1, Stream2).

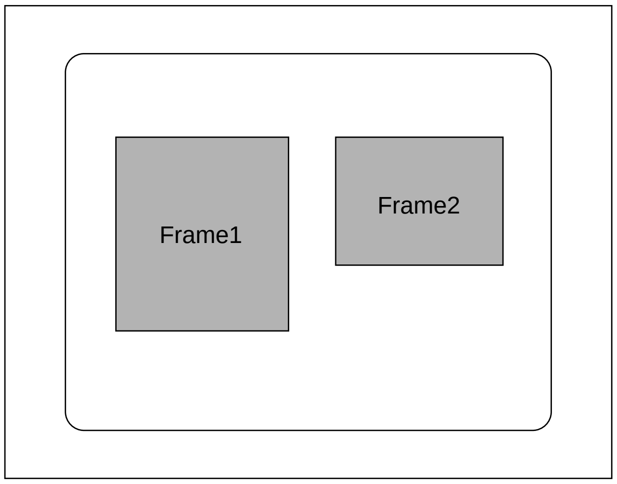

Fig.3– Schematic explanation of expression Sunion (Frame1, Frame2).

events. Time is specifiable in events with keywords such as “before”, “after”, “between”, “at”, and “periodically”. Events are invoked by actions within other processes. Conditions can monitor database states, process states, and QOS states. Actions specify control of processes such as parallel, serial, alternative, and other model operators such as structural, temporal, and spatial. Concurrency is represented as annotations together with QOS options, which reduces to other model operators in the case of serial compilers. Serial specification is simply object-oriented programming and query languages. Merging of processes is specified by the conjunction of events. QOS information is given as parameters to the process construct.

For example, the following shows the specification based on external time for parallel execution of Stream1' and Stream2' while taking two QOS parameters, latency and bit rate, into consideration:

```
QOS (Latency, Bit Rate) ;
Set1= (Stream1'  Stream2' );
Set1.parallel.play;
```

This realizes Script d). The effect is schematically shown in **Fig. 4**. We describe how to satisfy the QOS parameters in Section 3.2.

3. Implementation

3.1 System architecture based on OODB

We now describe the implementation of a multimedia database system. As shown in **Fig. 5**, the system architecture consists of agent management, media management, object management, data management, and multidatabase management layers on top of the OS and network protocol management layers. The agent management layer enables users to flexibly describe multimedia applications. The media management layer provides interfaces to individual media data. The object and data management layers, which provide basic database management facilities, are OODB-based.[2)] The multidatabase management layer enables integrated accesses to traditional media, for example, numbers and character strings, in an RDB and to new media, for example, audio and video streams, in an OODB.

In this section, we describe an OODB which interacts with the OS and network protocol management layers as the kernel of a multimedia database system. An OODB provides an object-oriented data model, as described in Chapter 2. The object management layer enables users to flexibly define and manipulate objects. Object manipulation is facilitated by combining object-oriented programming and query languages. The data management layer can efficiently store a large amount of objects and perform object manipulation.

The OODB system has a layered architecture consisting of object management and data management layers. The object management layer allows modeling and manipulation of objects. In particular, this layer has object buffers that efficiently manage objects in main memory. The data management layer provides support for transaction management and page buffer management as database functions.

As the storage manager of our OODB, the data management layer extends relational databases. This enables the user to define and access nested relations as well as flat relations. In addi-

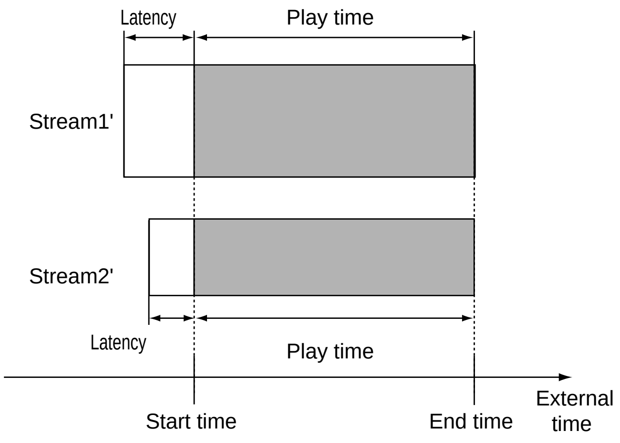

Fig.4– Schematic explanation of QOS-constrained concurrent playback of two streams.

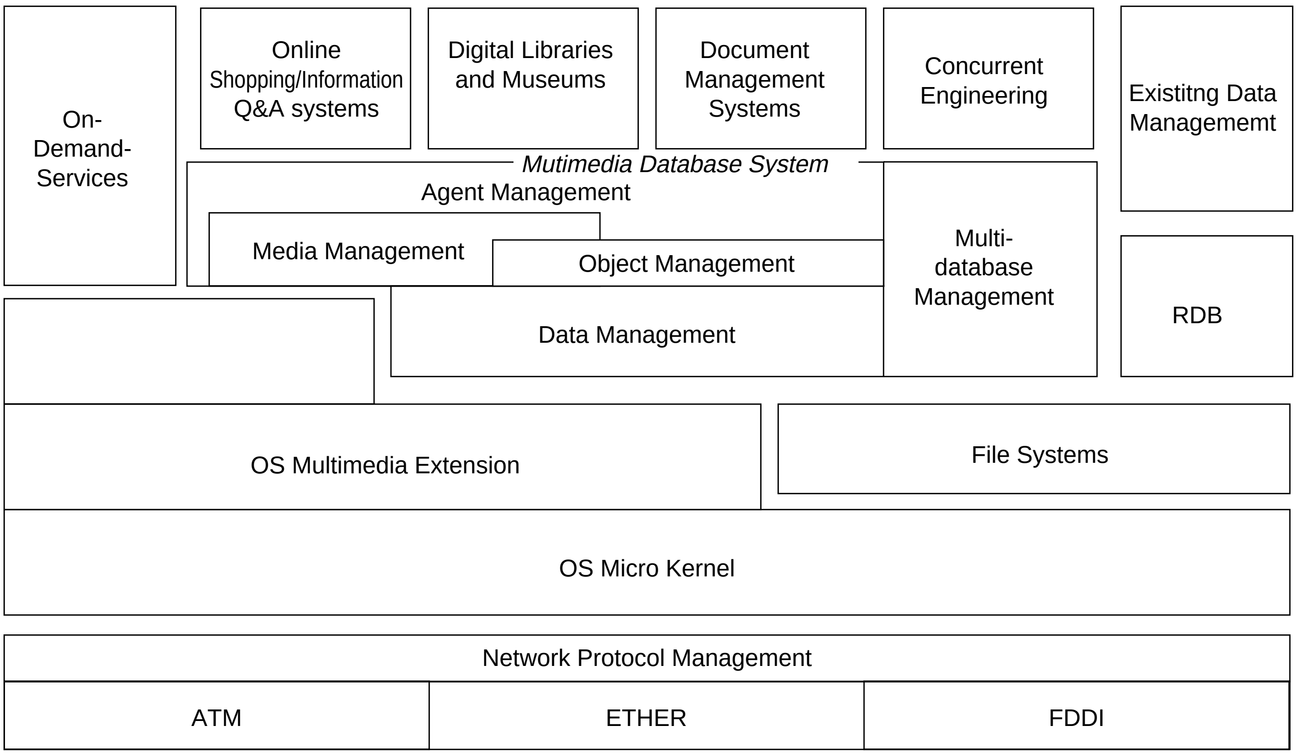
On-Demand-Services
Online Shopping/Information Q&A systems
Digital Libraries and Museums
Document Management Systems
Concurrent Engineering
Existitng Data Managememt
Mutimedia Database System
Agent Management
Media Management
Object Management
Multi-database Management
RDB
Data Management
OS Multimedia Extension
File Systems
OS Micro Kernel
Network Protocol Management
ATM
ETHER
FDDI

Fig.5– System architecture.

tion to reference and update operations, this layer provides nest and unnest operations for relations. The system provides sequential, B-tree-based, and hash-based access to flat and nested relations. A clustered index can be implemented by storing whole tuples into B-tree relations. A nonclustered index can be implemented by storing only keys and tuple identifiers into B-tree relations.

Objects are mapped into relations as follows. All intrinsic instances of a class are stored in a relation by making attributes correspond to fields. Intrinsic instances include inherited and non-inherited attributes. Multiple values are stored in multiple-valued fields, which are the simplest form of nested relations. Classes are stored in nested relations because they have nested structures.

The user can specify logical page sizes for each relation. Each class has its own page size. A class normally inherits the page size of its superclass. However, if necessary the page size can be enlarged. There is no limit on the number and length of tuples and fields, although whole tuples must be contained in one page. This enables the user to optimally store and access large-scale data such as images and text. Operations and tests on fields of relations, called manipulation and predicate functions, are treated as user-defined functions in the data management layer and are compiled into operations on data in page buffers.

Object queries are translated into relational operations such as selection and join. During this process, they are optimized. Object expressions may generate several joins whose execution order is determined dynamically. Joins are usually processed based on hashing. If an index is attached to fields, it is used for selection and join.

Page buffers are appropriate for access to homogeneous data but inappropriate for access to related heterogeneous data such as complex objects. Therefore, the object management layer provides object buffers. Objects, when accessed for the first time, are fetched from databases in secondary memory to page buffers in the data management layer. Only the required data is transferred to the object buffers from the page buffers. Object identifiers are represented as a triplet of database, class, and instance numbers. The identifiers of objects fetched into object buffers are translated into addresses in main memory. This eliminates the need for joins of relations and enables direct access of complex objects. The objects in object buffers also have tuple identifiers. If there are any updated objects in the object buffers, they are written back to the page buffers using the tuple identifiers at the end of the transaction.

Before a set-oriented query is evaluated, any updated objects associated with the query that are in the object buffers are moved to the page buffers. The query is then evaluated against the page buffers. Unlike our approach, Orion[14)] evaluates the same query for both object buffers and page buffers and integrates the results. Because our approach needs only one evaluation scheme, our system is more compact.

A query on nonleaf classes in a class hierarchy is translated into multiple queries on relations. Simple methods specified in a query, such as manipulation of attributes, are transformed into operations on fields of relations. These can be executed more efficiently on page buffers because unnecessary data transfer between page and object buffers is reduced. On the other hand, more complex methods, such as manipulation of heterogeneous objects of complex objects, are more efficiently evaluated in object buffers. Methods appearing in the condition part are similarly processed. Unlike other OODBs, our approach efficiently executes methods by combining object and page buffers. Hereafter we focus on the implementation of our current prototype.

3.2 Agent management

We will now describe the management of agents in our current prototype. Agents provide a distributed, object-oriented computing mechanism. Users describe multimedia applications in the form of scripts by using agents. In distributed computing, process migration from one server to another can also be done using agents. For example, in online shopping and VOD (Video-On-Demand) services, after a bunch of processes have been initiated and partially executed by one server, the rest of the processes are executed by another server at a client site. Agents also facilitate distributed hypermedia by retrieving and viewing hypermedia in a way that can be processed in a distributed manner. Scripts not only provide interfaces to the users but also provide the system with hints on QOS control.

First, information needed for QOS control is obtained by rehearsing scripts. Through rehearsal, QOS-related data are stored as properties of media objects. Next, a script execution plan is made based on the obtained data and then declared to the OS to reserve resources for QOS control. The agent management layer negotiates with the OS to satisfy QOS constraints. If necessary, plans, including QOS parameters, are changed. This phase corresponds to static optimization of script execution. In our current prototype we do not use a multimedia OS which can reserve demanded resources because, to our knowledge, there is currently no product that is sufficiently stable. Instead we do resource management at a user level.

Agents are basically objects, although they can be executed by migrating them to other processors available over a network if necessary. Scripts including agents are translated into objects and operations on them. Therefore, agents have storage structures identical to ordinary objects, although they may be moved to other servers. Individual structures of media types such as streams and texts are described later.

In general, there may be deviations from the reserved QOS parameters in real executions of scripts. Therefore, the agent management layer monitors environment information and checks the deviation from the expected values. If there is any deviation, the agent management layer dynamically changes the QOS parameters to satisfy the constraints. We use prefetching and caching techniques to satisfy QOS constraints such as latency.

3.3 Media management

This section describes the management of media data in our current prototype. Media management provides a versatile storage system which can be customized for various types of media. Users can specify interfaces to store and access individual media types by defining media storage objects. Moreover, the users can resolve heterogeneity in multimedia interfaces and physical media interfaces (i.e., data formats, CODECs, and devices) by using polymorphism of object methods.

We will illustrate the media management by using streams and text as examples.

1) Stream media data such as video and audio are usually unstructured and large-scale (i.e., large binary objects), compressed, and sequentially ordered and clustered. They are wholly or partially accessed, and more often referenced than updated. They are real-time accessed with QOS constraints. Access to whole streams, partial streams, and individual frames is allowed. Partial and individual access is required for frame-based image processing. A stream can be played in its entirety, or a section or single frame of a stream can be played. Streams can be played forward or backward at fast, normal, or slow speed. QOS parameters include latency, jitter, bit rate, frame rate, resolution, and color. An index can be created using the correspondence between frame numbers, time, and offsets. View streams can be derived from base streams stored in databases primarily by using an index on the time intervals. Indexes on keywords or other related objects can also be defined. View streams can be accessed and played with QOS constraints, like base streams. Composite view streams can be recursively defined using existing view streams.

In fact, a base stream is a logical stream which can have multiple physical streams of different qualities. A view stream is defined as a subset of a logical stream by specifying time intervals. Either the entire physical stream is stored in databases or only meta data such as size, load factors, and file descriptors are stored in databases. In the latter case, bulk data are managed by ordinary files or special stream servers. In both cases, to satisfy the QOS constraints, during script translation the system chooses the appropriate quality of physical streams that can be provided by the available computing resources.

The stream classes are defined (see **Fig. 6**). Streams can be accessed with uniform interfaces independent of CODECs such as MPEG and JPEG. Usually, stream data are sequentially clustered in favor of forward play, although they may be declustered for striping to allow parallel access.

To facilitate interactive retrieval of multimedia, we enable users to flexibly and efficiently access partial data such as sub-streams of videos using temporal information (e.g., temporal intervals), keywords, and other related information. This technique of subsetting a large amount of media data is analogous to RDB views. For example, **Fig. 7** shows the relationships among views, logical contents, and physical streams. A view selects a subset of logical contents by specifying a time interval. Keywords are attached to views for retrieval. Characteristic data such as

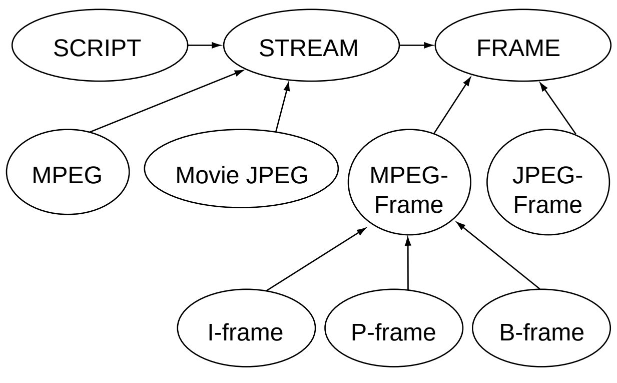

Fig.6– Class hierarchy of media objects.

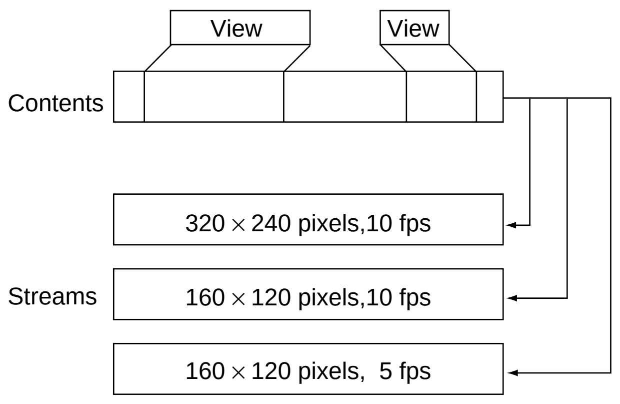

Fig.7– Views, contents, and streams.

figures, colors, and motion directions are attached to frames of streams corresponding to views for content-based retrieval. Logical contents can have several physical streams of different quality.

Unlike other approaches, we use a light-weight technique to segment scenes and recognize objects for content-based retrieval of stream data.[18] First, the system detects scene cuts by using differences in successive frames, such as motion vectors of MPEG macro blocks and colors. Then the user can define views of streams (i.e., sub-streams) by attaching keywords to such scenes. Further, the user can define new views recursively by combining existing stream views. The system also chooses representative frames, abstracts characteristic data, and then stores the frames and abstracted data into databases.

The system detects moving objects by using MPEG motion vectors and decreases the number of colors to more accurately recognize moving objects. In addition to figures and colors associated with moving objects the system also stores motion directions. The user can retrieve sub-streams corresponding to views with specified keywords. The user can also retrieve sub-streams containing samples of specified colors, figures, and motion directions. Content-based retrieval is used by end users and by content providers.

The following is an example of a script for content-based retrieval:

```
Script2:
    Set1 = VIEW from VIEW where
               VIEW.like (Sample1);
    On Event Selection by User;
    Set2 = Set1 from Set1 selected by User;
    Set2.parallel.play;
```

The user specifies a sample (e.g., Sample1) through a GUI as shown in **Fig. 8**. A sample figure consists of several parts like a human body. The system uses the largest part (i.e., the trunk) as a search key to a multi-dimensional index such as an R-tree. The system evaluates the other parts (e.g., the head) as additional conditions of a query. A content-based query evaluates to many views (e.g., Set1). Then the user chooses several views (e.g., Set2) for simultaneous playback.

We provide the means for flexible distribution and presentation of retrieved multimedia data over distributed networks by executing QOS control and scripts. To this end, we use techniques such as prefetching, caching, synchronization, and distributed processing.

In particular, unlike other approaches, our script scheduler detects overlaps of intervals of view streams appearing in users' scripts and selects appropriate physical streams by using views to enforce QOS control. For example, if the user chooses three streams for parallel playback in

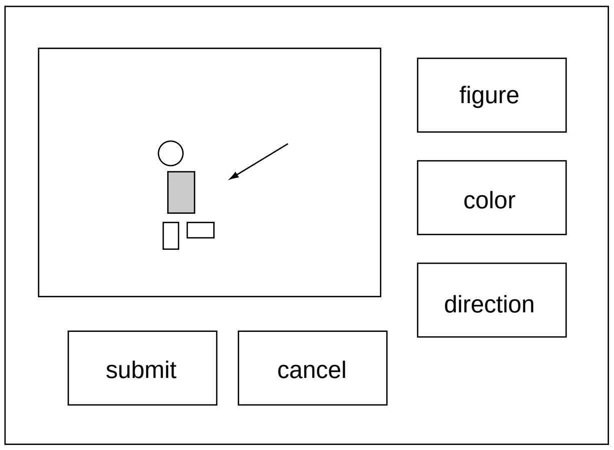

Fig.8– GUI for content-based retrieval.

Script2, the total playback time is divided into three intervals as shown in **Fig. 9**. The scheduler chooses physical streams of appropriate quality which can be played with the available CPU resources within each interval. In other words, physical streams of different quality may be chosen for the same view, such as a stream for View3.

2) Text data are usually structured, medium-scale, uncompressed, and hierarchically ordered and clustered. They are wholly or partially accessed and reference-oriented or update-oriented. Also, they are non-real-time accessed with few QOS constraints. Structural access via component relationships is allowed. Hypermedia links can be stored within texts. An index on keywords or links can be made. Format-independent interfaces such as SGML and ODA are provided using polymorphism. One possible QOS parameter is a font parameter.

The page manager of our OODB is being extended to accommodate different kinds of data (i.e., complex objects) within a single page. SGML texts are clustered in such pages along their hierarchical structures. Individual components of SGML texts, such as sections and paragraphs, are stored as objects. HTML texts are usually clustered according to media types such as text and graphics. Mutual translation among SGML, HTML, and plain texts is facilitated.

We provide media data management mechanisms which enable efficient storage and access

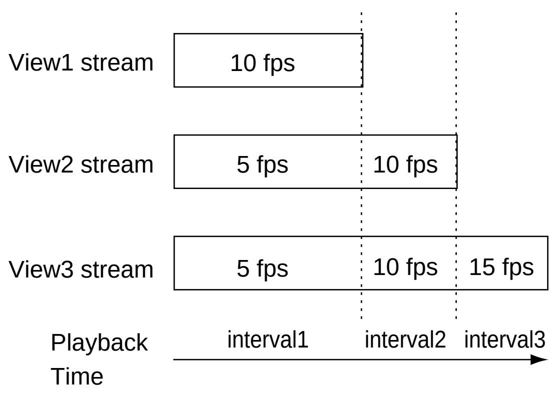

Fig.9– Script schedule example.

of large amounts of media data. They enable users to customize media-specific storage in areas such as indexing, clustering, and buffering. We take an object-oriented approach to resolving heterogeneity in data formats, compressors/decompressors (CODECs), and physical media used for implementation of logical media.

Structured texts such as SGML texts are often accessed according to component links. The system must cluster relevant texts in the same or neighboring pages so they can be retrieved efficiently. We assume that the user chooses to cluster texts. Thus, the user specifies how data are clustered. Then the system clusters data according to the user's specification. We are planning to provide a facility to monitor hot spots of access patterns. Either the system or the user clusters data based on the results of monitored accesses. We allow the user to recluster data after heavy updates.

In addition to heterogeneous clustering, we allow homogeneous clustering, for example, clustering of all instances of the same class. We allow subtrees of a whole component tree to be flexibly clustered according to the user's specification by combining homogeneous and heterogeneous clustering.

3.4 Multidatabase management

We will now describe the management of multidatabases in our current prototype. We provide multidatabase management to establish interoperability between existing media in an RDB and new media in an OODB. Heterogeneous database systems, data models, and schemas are integrated by model primitives, viewports, and views. Translation between different models can be done by enabling users to describe their models through model primitives. If there is no direct mapping between two different models, the users can specify rules to translate one schema at one site into a combination of schemas at another site.[19),20)] Using HTTP on the Internet, we provide an integrated interface to files, RDBs, and OODBs. Also, we manage HTML documents using an OODB because OODBs have common hyperlink structures, and we represent a hierarchically-structured directory of WWW servers by taking advantage of the generalization hierarchy of OODBs.

The framework we provide consists of model primitives, viewports, and views. Users describe their relational or object-oriented data models and schemas at local sites using the model primitives. Data model description using such primitives constitutes viewports, whose role it is to resolve heterogeneity in data models and database systems at local sites. At relational viewports, both relational and object-oriented schemas defined at other sites are viewed as relational schemas. Similarly, at object-oriented viewports, any schema defined at another site is viewed as an object-oriented schema. Within local sites, relational[11] and object-oriented[21] views are used to resolve such semantic heterogeneity. In other words, as a first step, viewports generated by system and model descriptions translate database schemas represented by a data model of a database system at one site into those represented by a model of a system at another site. As a second step, view mechanisms transform the schemas into ones which users at the site would like to view. Thus, we take a step-wise approach to resolving the three types of heterogeneity; that is, database system, data model, and semantic. Our approach enables us to concentrate on resolving one type of heterogeneity at a time and to incorporate existing technologies such as SQL and even new technologies such as object-oriented views into the framework.

Ease of data acquisition through the WWW, however, makes the size of collected data unmanageable for the user. Keyword-based retrieval alone is not sufficient. The system automatically abstracts keywords from collected HTML or SGML texts. Then, the system chooses the 100 most frequent keywords contained by a set of texts and places each text in an information space of 100 axes ranging from those that have the corresponding keyword to those that do not. The system uses a Self-Organizing Map (SOM)[22] technique to cluster a set of texts into the given number of groups in the above information space. The system displays the structured map by using a 3-D graphics technique such as VRML. The user can retrieve texts by navigating a 3-D user interface. The important point here is that the users cluster collected texts for their own use. Of course, content providers can use this technique when they cluster their own texts.

To allow for access control, we must maintain the integrity of programs for database access. Further, we must maintain program components for reuse and divide the processing between clients and servers. In general, the result of a database retrieval is rather large, so we must adopt protocols other than HTTP, which is insufficient for transfer of bulk data.[23]

Program components are managed by database servers. The system maintains information on, for example, program configurations. When the user requires programs, the system compares the user's configuration and the server's configuration by looking up information such as the configuration version and user level stored in databases. Then, the system sends only the differences it detected.

The system processes programs by distributing processes between clients and servers specified by scripts. Client processes take care of the graphical user interface, and server processes take care of database access. The query result is sent back to the client by the database system's protocols.

If a user query for program components is not satisfied by one server, the query is federated to other servers. We take an agent-based approach to such federation. The search agent also uses the SOM technique to construct an inter-server map for navigating the query through relevant servers, which is a new approach. The query result is sent to the mailbox specified by the client, then the client obtains the program components from the mailbox.

4. Conclusion

In this paper, we proposed a multimedia database system for multimedia applications that is based on an OODB model. We have developed a prototype multimedia database system to evaluate the proposed approach. The prototype supports scripts, keyword-based and content-based view retrieval with QOS control, SOM-based clustering, and WWW integration. We plan to enhance the functionality and performance by applying our technology to the promising application of document warehousing, in which corporate documents are managed on intranets.

References

1) Ishikawa, H. et al.: A Next-Generation Industrial Multimedia Database System. Proc. IEEE 20th Intl. Conf. Data Engineering, pp.364-371, 1996.

2) Ishikawa, H. et al.: The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System. ACM Trans. Database Syst., **18**, 1, pp.1-50 (1993).

3) Ishikawa, H.: Object-Oriented Database System. Springer-Verlag, 1993.

4) Ishikawa, H. et al.: An Object-Oriented Database System Jasmine: Implementation, Application, and Extension. IEEE Trans. Knowledge and Data Engineering, **8**, 2, pp.285-304(1996).

5) Ishikawa, H.: An Object-Oriented Knowledge Base Approach to a Next Generation of Hypermedia System. Proc. IEEE COMPCON Spring 90 Conference, pp. 520-527 (1990).

6) Berners-Lee, T. et al.: The World-Wide Web. CACM, **37**, 8, pp.76-82 (1994).

7) Gibbs, S., Breiteneder, C., and Tsichritzis, D.: Audio/Video Databases: An Object-Oriented Approach. Proc.9th Intl. Conf. on Data Engineering, 1993, pp.381-390.

8) Gibbs, S., Breiteneder, C., and Tsichritzis, D.: Data Modeling of Time-Based Media. Proc. ACM Sigmod Conference, May 1994, pp.91-101.

9) Hamakawa, R., Rekimoto, J.: Object composition and playback models for handling multimedia data. ACM Multimedia Systems, 2, 1994, pp.26-35.

10) Prabhakaran, B., Raghavan, S.V.: Synchronization Models For Multimedia Presentation With User Participation. Proc. ACM Multimedia 93, 1993, pp.157-166.

11) Date, J.C.: An Introduction to Database Systems. 1, Addison-Wesley, 1990.

12) Lohr, K.-P.: Concurrency Annotations for Reusable Software. CACM, **36**, 9, pp.81-89 (Sept.1993).

13) Ishikawa, H. et al.: An Active Object-Oriented Database: A Multi-Paradigm Approach to Constraint Management. Proc. 19th VLDB Conference, 1993, pp.467-478.

14) Kim, W. et al.: Architecture of the ORION Next-Generation Database. IEEE Trans. Knowledge and Data Engineering, **2**, 1, pp. 109-124 (1990).

15) Goldberg, A. et al.: Smalltalk-80: The Language and Its Implementation. Addison-Wesley, Reading, MA., 1983.

16) Maier, D. et al.: Development of an object-oriented DBMS. Proc. 1st OOPSLA Conference, 1986, pp. 472-482.

17) Stefik, M. et al: Object-Oriented Programming: Themes and Variations. AI MAGAZINE, **6**, 4, pp.40-62 (winter 1986).

18) Kato, K. and Ishikawa, H.: Content-based Retrieval System for Video Data. Proc. Third Intl. Conf. Computer Science & Informatics, pp.195-198, 1997.

19) Ishikawa, H. et al.: The design and Implementation of an Interoperable Database System based on Scripts and Active Objects. IEICE Trans. Inf. & Syst., **E78-D**, 11, pp.1396-1406(1995).

20) Kubota, K. and Ishikawa, H.: Structural Schema Translation in Multidatabase System:Jasmine/M. (in Japanese), Proc. IPSJ Advanced Database Symposium, 1994.

21) Ishikawa, H. et al.: An Object-Oriented Da-

tabase System and its View Mechanism for Schema Integration. Proc. Second Far-East Workshop on Future Database Systems (Kyoto, Japan), 1992, pp. 194-200.
22) Kohonen, T.: Self-Organizing Maps. Springer-Verlag ,1995.
23) Kubota, K., Takarabe, Y., and Ishikawa, H.: A Proposal of Agent System for Database Access. (in Japanese), Proc. IPSJ National Conf., 1996.

Hiroshi Ishikawa received the B.S. and Ph.D degrees in Computer Science from the University of Tokyo, Tokyo, Japan in 1979 and 1992, respectively. He joined Fujitsu Laboratories Ltd., in 1979. He is the chief architect of an object-oriented database system Jasmine. His research interests include object-oriented databases, multimedia databases, distributed databases, and active databases. He has published actively in international, refereed journals and conferences, such as ACM TODS, IEEE TKDE, VLDB, IEEE ICDE. He has authored a book ***Object-Oriented Database System*** (Springer-Verlag). He received the Sakai Memorial Distinguished Award from IPSJ in 1994 and the Minister of State Award from Science and Technology Agency in 1997. He is a member of IEEE, ACM, IPSJ, and IEICE.

E-mail : hiro@flab.fujitsu.co.jp