Volker Turau, Thomas C. Rakow

A Schema Partition for Multimedia Database Management Systems

Address of authors:

Prof. Dr. Volker Turau
Fachhochschule Gießen, Fachbereich MND
Wilhelm–Leuschnerstraße 13
D W-6360 Friedberg, Germany
E–mail: turau@prfhfb.fh–friedberg.de

Thank: This work was partially done while the author visited the database group of HP Labs in Palo Alto.

Thomas C. Rakow
GMD-IPSI
Integrated Publication and Information Systems Institute
Dolivostraße 15
D W-6100 Darmstadt, Germany
E-mail: rakow@darmstadt.gmd.de

In the interest of a subsequent final publication this manuscript should not be copied. Critical comments would be appreciated by the authors.

المنارة للاستشارات

www.manaraa.com

A Schema Partition for Multimedia Database Management Systems

Volker Turau [*]

Fachhochschule Giessen
Fachbereich MND
Wilhelm-Leuschnerstr. 13
W6360 Friedberg, Germany
e-mail: turau@prfhfb.fh-friedberg.de

Thomas C. Rakow

GMD - Integrated Publication and
Information Systems Institute (IPSI)
Dolivostr.15
W6100 Darmstadt, Germany
e-mail: rakow@darmstadt.gmd.de

Abstract

We propose a partition of the conceptual schema of multimedia databases according to the different aspects of multimedia data. (1) In a multimedia type schema the specific characteristics of multimedia data are modelled but a homogeneous interface utilizing polymorphism is used. (2) A device schema makes applications independent from low level aspects of multimedia presentation devices and import or export from and to external environments. (3) An interaction schema allows interactive manipulation of presented multimedia data. The requirements of managing multimedia data in a database system like storage of high volume data and time-dependent presentation lead to support specific built-in functionality. This is also reflected in a new architecture currently under development using the object-oriented database management system VODAK designed and implemented at GMD-IPSI. The proposed partition provides a high degree of flexibility for extending the database system and designing multimedia applications. By making types describing devices and interaction tools part of the database schema, interactive queries can be supported by the database system.

Keywords: multimedia database management system, schema modeling, interactive query language, object-oriented.

[*]This work was partially done while the author visited the database group of HP Labs in Palo Alto.

1 Introduction

Many types of information which were previously represented in analog form are now also available in digitized form. The most important examples are audio and video and samples of signal data such as ECG. This has several advantages such as lossless storage and transmission of the data. Furthermore it offers the possibility that all information can be processed by a computer, thereby the variety of ways data can be utilized and manipulated is enormously increased. Combining these new multimedia datatypes with data types already in use such as text and numbers can be achieved, resulting in a much more flexible management of the data.

Since multimedia systems collect, maintain, manipulate and present on request large volumes of persistent data, which often are changed by many users in parallel, a database management system is an essential ingredient of multimedia systems. A multimedia database management system (MM-DBMS) should have the capability of storing, managing and retrieving information on individual media, managing interrelationships between the information represented by different media and it should be able to exploit these media for presentation purposes. The advantages a database management system offers in comparison to file systems are well known: physical aspects of storage are transparent for applications, protection of data through defined access methods, query facility, multi-user access through transaction mechanisms and reliability through recovery concepts. Object-oriented database systems additionally provide the encapsulation of data and behavior and offer extensibil-

المنارة للاستشارات
www.manaraa.com

ity [3, 4]. In order to preserve these advantages, it is necessary to integrate multimedia data into database management systems.

So far the focus of research on multimedia systems has been on conceptual modeling and single user systems. Most of the work favors the object-oriented approach and there seems to be a consent which constructs should be supported. But many proposals resemble more a programming environment than a multimedia database system. Very often database system issues such as query processing, user interaction or architectural implications remain unconsidered.

In this paper we propose a partition of the conceptual model according to the different aspects of management of multimedia data:

- *a multimedia type schema* to model the specific characteristics of multimedia data;
- *a device schema* to make applications independent from low level aspects of presentation and import or export from and to external environments;
- *an interaction schema* to allow interactive manipulation of presented information.

Our goal is to provide a high degree of flexibility for designing applications and on the other hand recognizing database specific requirements. For reasons of efficiency it is sometimes not feasible to realize some types as suggested from a modeling point of view. This can be resolved by providing built-in multimedia types with interfaces that conform to the logical view and have an efficient implementation. For example from the model point of view a video is a sequence of frames with a corresponding audio track. But if each frame is treated as an individual object, continuous playing is not possible. Thus a video can statically be modeled as a list of frames but dynamically it behaves differently. This way the enormous sizes of the data and the real time aspects of their presentation can be handled. By making types describing devices and interaction tools part of the database schema, interactive queries are possible (e.g. queries where the input is provided in a nontextual form by devices).

The partition of the DBMS schema has several advantages. Individual parts can be compiled separately and the information is stored in the data dictionary of the database. Other schemas can include those parts with an include mechanism similar to that found in programming languages. This way users can share schemas. This will ease the maintenance of interfaces, facilitate the development of new applications and reduce the compile time.

VODAK is a prototype of an object-oriented and distributed DBMS developed at GMD-IPSI [8, 7]. One of the applications of VODAK was a video database. The gained experience led to us to the proposed architecture and currently this architecture is realized.

The main contribution of this paper is a strong consideration of database specific issues in the design of a multimedia system. The requirement for interactive queries as well as the recognition of the difference between the modeling functionality and the built-in functionality lead to a partition of the database schema. This is also reflected in a new architecture for a multimedia database.

After reviewing the existing work and some characteristics of multimedia data types an architecture for a multimedia database is presented in section 3. In section 4 the schema partition is introduced and the individual parts are described. In section 5 it is demonstrated how the different parts work together.

1.1 Existing Work

Comprehensive multimedia processing, which has not been widely put to practical use to date, becomes possible; here multimedia processing encompasses the integrated generation, manipulation, presentation and storage of machine processable information expressed by different media. This is already reflected in a variety of new products such as Intel's DVI technology (digital video interactive) and CD-I (compact disc-interactive) which is a system specification for interactive audio, video, and computer system based on CD as the storage medium. The spectrum of multimedia applications is wide and ranges from education over mailing to publishing systems.

Multimedia database systems are a relatively new field due to the fact that the necessary hardware became available only recently and is still developing. The first approaches were database systems for specialized data such as spatial databases [10], [9] and pictorial databases [13]. But in some cases a DBMS was only used as a somewhat complex file system. Spatial databases were attractive because the semantics of the objects and operations were clearly defined and their properties can be derived

www.manaraa.com

from geometry.

One of the first multimedia effort in managing multimedia data was the Multimedia Information Manager in the ORION object-oriented database system, developed at MCC [15, 16] and now available as the product ITASCA. The integration of the new data types is accomplished through a set of definitions of class hierarchies and a message passing protocol not only for the multimedia capture, storage, and presentation devices, but also for the captured and stored multimedia objects. This way a high degree of flexibility is achieved since new storage or presentation devices are easily included by providing the corresponding types as subtypes of the existing types. This approach is very promising, but it remains more or less a collection of classes. Specific database issues such as query processing, user interaction and architectural implications are not considered. Furthermore, it can be questioned whether the modeling of devices down to the level of methods such as *get-next-block* lead to efficient realizations. In the case of continuous media such as video or audio this is just not feasible.

Other projects are MINOS, the multimedia object presentation manager developed at the University of Crete [2] and DOM dealing with distributed multimedia object management [6].

Synchronization mechanisms for distributed multimedia systems are addressed in [12], an approach for modeling time-dependenies of multimedia data can be found in [1], and the design and implementation of a continuous media player is described in [11].

2 MM-DBMS vs DBMS

Some of the characteristics of multimedia datatypes make MM-DBMS different from traditional DBMS. These differences will be discussed in the following.

Multimedia data differs from ordinary data in that respect, that the presentation of the data is not canonical for traditional computer systems. The types of conventional data usually comprise the types known from programming languages (e.g. character, integer, real or records of these types) and their representation is inherent to a computer system. Multimedia data is not directly supported by programming languages and its presentation depends on special devices and additional information such as image format, compression techniques and layout description is necessary.

One of the key obstacles for many multimedia applications is the vast amount of data involved. The use of digital images often is not viable due to high storage or transmission costs, even when image capture and presentation devices are affordable. Modern image compression technology can compress typical images from 1/10 to 1/50 of their uncompressed size without visibly affecting image quality. But the storage requirements for one object still exceeds the size of an average object handled by conventional database systems. Storage media such as optical disks have to be considered. Also the available techniques for buffer management, indexing, recovery etc. are not suitable for these sizes of data, for example objects do not fit into main memory in their entity.

Traditional database applications use data of fixed size, but the size of multimedia data such as bitmaps, voice or video is not fixed and can vary dynamically. This poses new problems for the management of the storage layer. New storage structures for continuous data are needed. A frame in a video can be regarded as a single object and treated accordingly in the database, but this way continuous presentation is not feasible. Furthermore, many traditional databases do not support unformatted data. So far all unformatted data (mainly text) has been handled in database systems through long fields or BLOB's (binary large object), but these usually support only a few operations such as reading or writing them in a single operation as a unit and they are excluded from queries. It is necessary to access or update parts of long fields and it must be possible to impose some structure to accommodate for example a frame structure like in a video sequence.

In audio, video, and voice the information itself is expressed as a function of time. Synchronization assures a temporal order of events. Examples of synchronization of multimedia data are a movie and its soundtrack or the synchronization of two stereo channels. Conventional databases do not provide mechanisms for expressing these synchronization conditions nor for controlling them.

In contrast to conventional data the production, manipulation, and presentation of multimedia data is performed with special devices or tools which are external to the database. Hence these processes cannot be done inside the database unless these tools are integrated into the MM-DBMS. To provide the flexibility needed, generic interfaces must

www.manaraa.com

be provided to integrate new devices. The access to these devices by concurrent presentation processes must be controlled and the interaction with those must be handled (i.e. interrupting the presentation of a continuous multimedia object).

The integration of presentation devices is very appealing from another point of view. They enhance the possibility for user interaction with the database, for example interaction supported queries (e.g. based on pointing device) become possible. This can only partially be achieved in case these devices are included in the application.

2.1 Multimedia Information

Multimedia information refers to a combination of text, still images, video and sound. Multimedia documents are examples which comprise the above information types. The new datatypes can be categorized as discrete or continuous data, stressing the time dependency. In the following the characteristics of these information types are presented.

2.1.1 Discrete Data

Still Images (Bitmaps)
The arrangement of adjacent pixels and lines in a contiguous region of memory representing an image is called a bitmap. There are several different formats for storing colored images available such as GIF, Portable Bit Map or JFIF etc. The crucial value of a bitmap is the number of bits used to represent the color value of each pixel in a digitized image. Real color pictures supporting 16.7 million colors require at least 24 bits per pixel. A high resolution VCR has a resolution of 1152 x 900 pixel. Under these assumption one color picture requires about 3 MBytes of storage. Modern image compression techniques can compress typical images from 1/10 to 1/50 of their uncompressed size without visibly affecting image quality. A general-purpose standard image compression technique for continuous-tone still images has been established by the Joint Photographic Experts Group (JPEG) [14]. This technique is applicable to practically any kind of continuous-tone digital source image and has tractable computational complexity. The JPEG proposal also contains an interchange format syntax which ensures that a JPEG-compressed image can be exchanged successfully between different application environments.

Graphics
Structured graphics are widely used on computers. In the vector format, a graphical image is stored as a group of shapes at specified positions (points, lines, rectangles, circles etc.). The shapes are hierarchical composites of more primitive onces. A complex group of shapes may be as large as 100K bytes. There are some proposals to standardize the definition of these shapes, such as GDI in Microsoft Windows and Apple Macintosh's Quickdraw or CGM in ODA. These standards are intended to allow interchange of graphics data between different applications on different machines. It is possible to translate an image in a vector format into a bitmap format but not vice versa applicable to a broader range of images.

2.1.2 Continuous Data

Video
Video data consists of motion picture and audio. The amounts of storage for digital video are enormous. For a small display window of size 182x137 at a frame/second rate of 30 using 16 bit/pixel a 40 MByte disk could store only 4.5 minutes of video (not taking into account the audio). Even this is only feasible using modern compression techniques. This implies that different storage devices are needed such as CD-ROM, WORM or MO. The aim of the Moving Picture Expert Group (MPEG) is to develop a standardization of video compression techniques [5]. MPEG is aiming for applications on different storage media and communication channels such as CD-ROM, DAT, Winchester Disk, writeable optical disks, ISDN, LAN, etc. The compression algorithm achieves a high compression associated with interframe coding, while not compromising random access for those applications that demand it.

Audio
The storage requirements for audio are not as high as for video. A Compact Disc - Digital Audio (CD-DA) plays stereo audio for up to 74 minutes, which is equivalent to 764 megabytes of digitized audio data. Unfortunately there is no natural structure to impose on audio like frames for video. Obviously there is no equivalent to a still picture in audio.

www.manaraa.com

3 An Architecture for a MM-DBMS

Traditionally queries are submitted to the DBMS in a textual form and the system returnes the result again in textual form. Multimedia data requires an extension of this concept. Results are no longer presented solely in textual form, therefore the corresponding display forms must be part of the user interface. This opens new possibilities for the formulation of queries, because queries can reference displayed objects (e.g. images), parts of those (e.g. features) with pointing devices or audio just played. Thus, queries are no longer restricted to textual form. They can include non-textual parts, such as locations of points in a displayed map or frames of a presented video provided by special devices. Similar arguments show that the manner data is supplied to the system exceeds traditional forms. Therefore, capture methods must also be provided by the user interface.

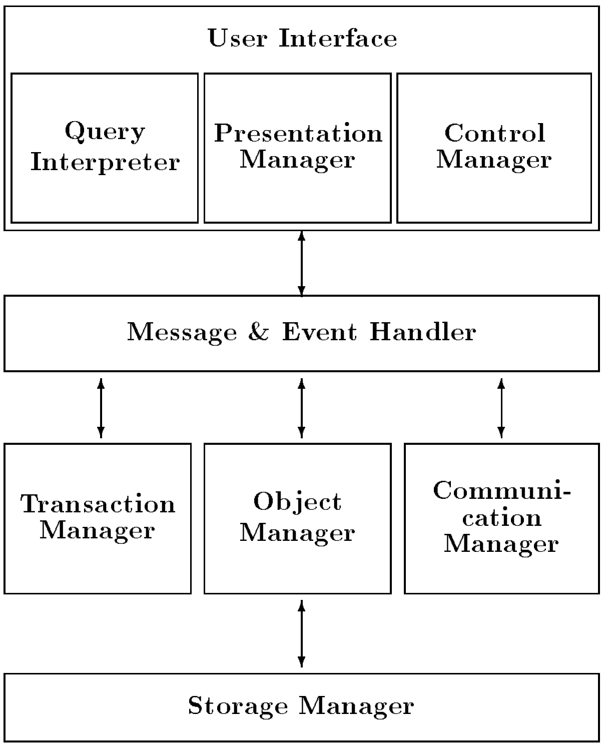

Figure 1: Architecture of the VODAK MM-DBMS.

These requirements are reflected by the architecture of the VODAK DBMS [8, 7]. VODAK is a prototype of an object-oriented and distributed DBMS developed at GMD-IPSI. Currently, existing components of it are changed and new ones are added for enhancement in direction to an MM-DBMS. The architecture of VODAK is shown in figure 1. Every application has its own modules which are shown in the figure but data structures like object buffer and lock tables are shared.

The user interacts with a user interface that is conceptually divided into three parts: query interpreter, presentation manger and control manager. These modules are described in detail because they support the above mentioned requirements of querying the database and interaction with the user.

All query input is handled by the *Query Interpreter*. The query is assembled from the query expression and additional information provided by the Presentation Manager (e.g. location in a window supplied by a mouse). Analogously the result of a query is supplied to the interpreter which in turn utilizes the Presentation Manger to present the results accordingly.

The *Presentation Manager* separates all other modules from the task of presenting database entities. For example information about locations of display windows and availability of devices is administered there. Presentation combines operations to display, to animate, and to let the user hear something. The Presentation Manager reliefs applications of the burden of knowing the means of presentation. An application should not have to specify how and on which type of device a result of a query is to be presented, i.e. whether a piece of text will be displayed or presented acoustically or both. This doesn't prevent an application to choose among the available forms of presentation. Each new presentation device or presentation method (e.g. a browser) can be incorporated into the Presentation Manager.

The *Control Manager* controls the presentation of multimedia objects and all interactive user input. The user can change the presentation style or interrupt the presentation through the Control Manager. Moreover, the capture of objects is also controlled by this module. If a presentation is stopped by a user event this request is notified to the Message & Event Handler.

A short overview of the other modules completes the description of the architecture. The *Message & Event Handler* is responsible for exchanging messages between objects. The message handler determines the receiver object of a message and the method to execute. The module super-

www.manaraa.com

vises time constraints and handles synchronization events. Furthermore, the Message Handler supports interruption events by pausing or terminating currently executed presentation operations. The *Transaction Manager* performs the same task as in traditional DBMS. The problems occurring here are the same as in conversational transactions (i.e. a roll back is not possible for already presented data). The *Object Manager* is responsible for the handling of the data. Traditionally the objects had all the same size, this is fundamentally different when objects with continuous data are handled. Here the sizes of the objects can vary drastically and some objects don't fit into the database buffer in their entirety. The *Communication Manager* provides reliable and efficient access to remote and heterogeneous databases. In addition, it supports a client server architecture. The *storage manager* separates storage devices from the other modules. New storage devices can be integrated at this level, thereby their characteristics are hidden.

The advantage of this architecture is that it provides a high degree of flexibility and extensibility. By providing a generic interface, the details of the different devices are concealed.

4 A Schema Partition for Multimedia Data

The proposed architecture requires that the information about the database entities must be organized in such a way, that the different modules are supported accordingly. This includes the above mentioned data that characterizes the interactive presentation of the data, which is beyond the information the user is interested in. This data was traditionally coded into the system. To support new evolving applications it is necessary that this information can be altered and extended. Therefore, it should be treated similar to the logical and content information and be part of the schema. This way it can be shared by multiple users and can be extended.

The interaction manager requires information about how to present objects of different types. This is independent of the logical organization of the objects with respect to the application. Furthermore, the information needed to capture objects is independent from the presentation information. Therefore, we suggest to partition the database schema into different categories. Each category is dedicated to a different aspect: representation, devices, and interaction. They can be regarded as orthogonal. This partition facilitates the development of new applications, since many parts can be reused. The necessity of such a partition can also be motivated from this different point of view.

The primary goal of a MM-DBMS is to realize the basic functions such as retrieving, searching and updating multimedia data. The next step is to utilize the rich semantics carried by these data. This can only be achieved by content search, which is a very difficult problem as can be seen from the area of information retrieval, which is still a very active area of research. The problems of content based search lie outside the scope of database research and are difficult to solve. The task of a database management system is to provide the framework for doing content search. The organization of the data must be organized, such that these operations can be added easily. A clear separation of different categories of data is necessary: some data are useful for content queries, other data are only needed for internal use such as presentation. This implies the need for a meta organization of the data.

Traditionally information is classified according to whether it describes logical or physical aspects. This distinction is reflected in the existence of a conceptional and a physical schema. We suggest to enhance this separation by providing three new categories:

- raw data
- device data
- presentation and control data

This can be achieved by defining different schemas for the different aspects of multimedia data.

- a *multimedia type schema* providing types that can be used to model multimedia data in applications, thereby separating the representation of the data and the supported operations from aspects specific to applications. Typical examples are types for arbitrary long bytestrings, bitmaps, audio or video;
- a *device schema* providing types constituting an abstraction for different system configurations in terms of storage devices, auxiliary devices like compression/decompression hardware, and display hardware;

المنارة للاستشارات
www.manaraa.com

- an *interaction schema* providing types for controlling and accessing the presented multimedia objects on different devices such as windows and loudspeakers. These types are the basis to perform interactive queries.

The *application schema* provides classes for representing objects of the application and their semantic relationships. This corresponds to the standard conceptional schema of a database. The objects of relevance for the application are modeled using the types defined in the other schemas. This way the designer is freed from considering the characteristics of devices and interaction. This eases the maintenance of applications.

The advantage of this partition is a clear separation based on the semantics of the data from the viewpoint of the database system. This way a high degree of extensibility is achieved. The separation of presentation data from the application schema makes changes of presentational aspects of the data independent of the application schema. To render interactive queries it is indispensable to have access to the presented data (e.g. image in a window) or to the actual status of a presentation (e.g. slider representing the time code of audio or video data). The interaction schema provides the necessary abstractions. Multimedia data is intrinsically tied to special devices. Making information about devices a part of the database and separating this information makes the system very flexible.

The different schemas are compiled separately and the information is stored in the data dictionary of the database system. Application schemas can include these and utilize the types defined there. This way application schemas are smaller and easier to compile. Furthermore the maintenance of the schemas becomes an easy task, since many schemas use the same device and multimedia type schemas. Thus, it suffice to change only those, as long as the interfaces of the types don't change.

In the following the individual schemas are explained in detail and their use is illustrated with examples.

4.1 Multimedia Type Schema

This part describes the different types that can be used to model multi-media data in applications. The purpose is to separate the representation of the data and the supported operations from aspects specific to applications. For example the concepts of a photograph and a bitmap must be distinguished. The former is clearly a notion used in an application. A photograph has attributes such as photographer, date, location etc., whereas the notion of a bitmap refers to the representation and provides data such as colors, size etc. The multimedia type schema contains the types that deal with the representation and handling of multimedia objects. They form the basic building blocks for applications.

There are two aspects here to consider, the logical structure of the objects and the representation of the primitive constituents. For bitmaps the logical structure consists of width, height, pixeldepth etc. whereas the pixel matrix itself contains the bulk of data. For the former the object-oriented data model with its complex types provides suitable means. The fact that there are different representation schemes for bitmaps can be modeled by providing a type BitMap and appropriate subtypes. The general type BitMap has attributes such as width and height specifying the dimension of the matrix of pixels. Furthermore, the methods for manipulating BitMap objects such as rotate, zoom and crop are defined here and can be redefined accordingly.

To represent the primitive constituents i.e. the raw data of multi media a built-in type ByteString is introduced. A ByteString can store arbitrary long sequences of bytes. It can be used to store the raw data of images, audio and video. It provides methods for accessing the content in various ways or as a whole. Moreover, appending, replacing and cutting of chunks of bytes is supported. Since efficient compression algorithms are media dependent, they are defined in the corresponding types. Figure 2 shows the definition of the type ByteString.

```
Built-in Type ByteString
get(offset : Unsigned, length : Unsigned) : ByteString;
append(offset : Unsigned, aByteString : ByteString);
insert(offset : Unsigned, aByteString : ByteString);
delete(offset : Unsigned, aByteString : ByteString);
importFromFile(name : String) : ByteString;
exportToFile(name : String);
```

Figure 2: The methods of built-in type ByteString.

The type ByteString is used to represent several multimedia types. Figure 3 shows the definition of the type BitMap. The attribute rawData of type ByteString stores the pixel matrix

www.manaraa.com

of the bitmap. The other attributes width, height and depth hold the relevant information about a bitmap. A few methods for manipulating bitmaps are also included. Specific implementations of bytestring can be used to store the raw data of audio and video. The corresponding built-in types guarantee that the units of the raw data are clustered.

```
Type BitMap
width : Integer;
height : Integer;
depth : Integer;
presentation: {Devices}
rawData : ByteString;
rotate(degree : Integer);
zoom(factor : Integer);
crop(x,y : Integer);
```

Figure 3: The multimedia type BitMap.

The designer of an application schema can use these multimedia types and is therefore not concerned with the formats of the images. The multimedia type schema contains types for all media types video, audio, images and graphics. The types will contain references to types which describe how to present the objects of those types. These are collected in the device schema.

4.2 Device Schema

The types in the device schema provide an abstraction for interfacing to specific devices such as input/output devices (loudspeakers, scanners etc) or auxiliary devices (compression/decompression). They conceal aspects of lower levels of the system architecture and encapsulate the corresponding behavior. Metainformation about the devices is stored and an interface is provided.

The implementation of the methods of these types is not given in the modeling language. This would decrease the performance considerably and in cases such as playing a video this is just not possible due to the real time aspects. Instead the methods are realized as foreign functions. From a conceptual point of view these types are treated as all other types. They do not substitute low level real-time device drivers, they merely constitute the logical abstractions and provide hooks allowing the development of entire multi-media applications. Furthermore, they foster easy adaptation of local devices or environments to existing application schemas.

The different media need different presentation devices and for each type there may be different devices available. Each type in the multimedia type schema has an attribute specifying a list of devices on which the instances can be presented. Furthermore, a present method is also provided which by default utilizes the first device in this list for the presentation of instances. The device receiving such a message activates is own present method. Alternatively, the present message can be sent to any one of the appropriate devices.

Figure 4 shows the definition of the type AudioDevice. The parameter of the method play is an instance of the AudioClass which collects all instances of the datatype Audio.

```
Type AudioDevice
type : String;
play(data : AudioClass);
setSpeed(speed : Real);
setVolume(level : Integer);
setPitch(level : Integer);
setSampleRate(rate : Integer);
```

Figure 4: The device type AudioDevice.

4.3 Interaction Schema

The purpose of an interactive query language in a MMDBS is the same as in a conventional DBMS: retrieve data and present the results. Existing query languages have emphasized the first aspect, the retrieval of the data in a standardized language. The second aspect becomes more important in the framework of multimedia data compared to data in the form of numbers and strings. The main focus is:

- complex structures compared to traditional table based data;
- the visual presentation of query results;
- the interaction between users and the presentation of multimedia data;
- using results as input for subsequent queries.

The first aspect must be addressed by any query language for an object-oriented datamodel. Traditionally query languages provide a single, predefined format for presenting query results. By using a

المنارة للاستشارات
www.manaraa.com

device schema it is possible to specify device dependent presentation methods. But it is still necessary to specify the format of the presentation of the result of a query returning multiple objects (browsers, different windows etc.).

The manipulation of data through the query language in conventional DBMS can be reduced to operations on simple data types (e.g. integer, string). Communicating with the presentation of multimedia data at this level of abstraction provides significant difficulties, because users would be required to deal with a low level of abstraction. Instead, the interaction must be based on operations defined with the corresponding types. The interaction between a user and the presentation of multimedia data should be centered toward using the representation available to the user as a reference in upcoming queries. Typical query languages are command languages with all input from the keyboard. Extending the user dialog by using pointing devices for selection by pointing in a window promotes the usage of query results as a reference in subsequent queries. The new concepts such as windows, sliders or pointing devices must be made part of the query language, such that their semantics is inherent to overall model.

```
Type Slider
type : String;
cursor : Bitmap;
window : Device;
getPos() : Integer;
markPos();
changedPos() : Bool;
```

Figure 5: The interaction type Slider.

The types in the interaction schema provide the methods to perform interactive queries. Figure 5 shows the definition of the type Slider. The method getPos provide the value of the slider position. It can be used inside a query expression.

4.4 An Example Type Hierarchy

Figure 6 depicts a type hierarchy for devices. The intention is not specify the ultimate device schema, rather it is supposed to illustrate the general idea. Generally, devices are distinguished according to acoustical and visual presentation. The device *file* allows to import and to export multimedia data from and to the filesystem. Types representing acoustic devices to record or to play audio data such as microphones or loudspeakers respectively are subtypes of *AudioDevice*. The device *Window* presents images, graphics as well as video data. The specialization of Window named *CaptureWindow* allows to import visual data from the screen.

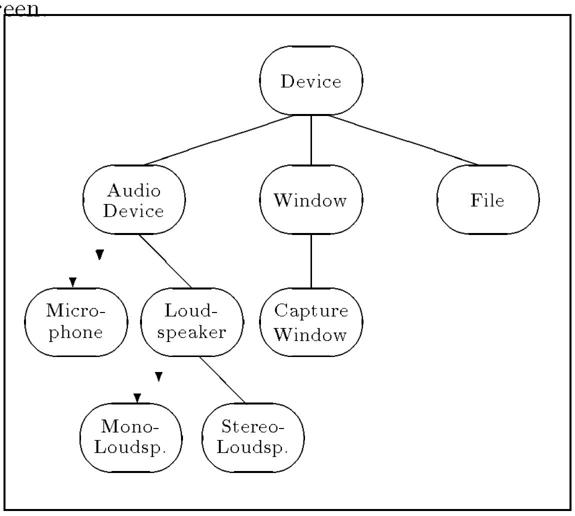

Figure 6: Example of a Device Type Hierarchy.

5 Doing the Job

This section describes how the different schemas work together. It should be clear from our description of the architecture in section 3 which and how the modules are involved.

There exists a generic interface for the multimedia types defined in the Multimedia Type Schema. The operations allow to import and to export objects from or to an external environment and to present objects to the user. To utilize type dependent semantics polymorphism is used for redefinition of these methods in each type. The operations for encoding and decoding to compress/decompress the data are encapsulated in the import/export methods. A general purpose *looseless* compression method can be used in case no specific method is available. Sources and sinks of the interface operations are objects of the Device Schema. Thus, an easy to use access is offered to heterogeneous operations like capture multimedia data shown on a screen, digitize analog signals, and read or write data to or from the filesystem. The generic interface is extended for continuous multimedia types to stop, to pause or to continue a presentation. Figure

www.manaraa.com

7 shows the generic interface and some examples.

```
<MM_Object>→import(<Device>);
<MM_Object>→export(<Device>);
<MM_Object>→present(<Device>);
<MM_Continuous_Object>→stop;
<MM_Continuous_Object>→pause;
<MM_Continuous_Object>→continue;

aBitMap→import(aCaptureWindow);
aBitMap→present(aWindow);
aBitMap→export(aFile);
aVideo→import(anAnalogVideoDevice);
aVideo→present(aWindow);
aVideo→stop;
```

Figure 7: The generic interface for multimedia types and some examples.

The interaction between users and the presentation of multimedia objects is implemented by simply sending methods to instances of the interaction types. The method results can be used as parameters of other methods. For example, the currently shown picture of a stopped video can be grapped by sending a message to a corresponding slider instance: `aBitMap.rawData:= (aVideo→getFrameAt(aSlider →getPos).`

6 Conclusion

In this paper we have described a system for partitioning the schema of a MM-DBMS. This was motivated by the need for flexibility and extensibility in the design of multimedia applications and by the special dynamic requirements of these data types. Due to the enormous sizes of the data and the real time aspects of the their presentation continuous data can not be handled as traditional data. Therefore, we provided built-in types such as ByteString for efficiently handling this data. Built-in types are a necessary supplement to the data types provided by database systems to handle discrete data. This way, a designer is only concerned with modeling the semantic relationships of the objects and not with the problem of efficiently handling them. Only the interfaces of the types are specified in the modeling language, their implementation is hidden and can be realized in a different language. For example from the point of view of the datamodel a video clip is a list of frames i.e. individual objects, but it is not implemented this way.

By making types describing devices and interaction tools part of the database schema, a high degree of flexibility is achieved. Furthermore, making the interfaces of devices part of the schema enables interactive queries, i.e. queries where input and output is provided in a nontextual form by devices.

References

[1] Aberer K., Klas W. *The Impact of Multimedia Data on Database Mangement Systems* ICSI, TR-92-065, Berkeley, Ca., September 1992.

[2] Christodoulakis S., Ho F. and Theodoridou M. *The multimedia Object Presentation Manager of MINOS: A Symmetric Approach* Proc. Int. Conf. on Management of Data, Washington, 1986, 295 - 310.

[3] Fishman D.H. et al. *Iris: An object-oriented database management system* ACM Trans. Office Inform. Syst., Vol. 5, No. 1, Jan. 1987.

[4] Klas W., Neuhold E.J. and M. Schrefel. *Using an Object-Oriented Approach to Model Multimedia Data* Computer Communications, Special Issue on Multimedia Systems, Vol. 13, No. 4, 1990, 204 - 216.

[5] LeGall D. *MPEG: A Video Compression Standard for Multimedia Applications.* Communications of the ACM, April 1991, Vol. 34, No. 4, 46 - 58.

[6] Manola F., Hornick M.F. and Buchmann A.P. *Object Data Model Facilities for Multimedia Data Types.* TM-0332-11-90-165, GTE Laboratories Incorporated, Waltham MA, 1991.

[7] E. J. Neuhold, V. Turau (Eds.) *Database Research at IPSI.* SIGMOD RECORD Vol. 21, No. 1, March 1992, 133-138.

[8] P. Muth, T. C. Rakow, W. Klas, E. J. Neuhold *A Transaction Model for an Open Publication Environment.* Ahmed K. Elmagarmid (Ed.): Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers, San Mateo, Ca., 1992.

[9] Stonebraker, M., Rowe, L.: *The Design of POSTGRES.* Proc. ACM SIGMOD 1986.

المنارة للاستشارات
www.manaraa.com

[10] Orenstein, J., Manola, F.: *PROBE Spatial Data Modeling and Queryprocessing in an Image Database Application.* IEEE Trans. Software Eng. 14, No. 5 (1988).

[11] Rowe, L., Smith B.: *A Continous Media Player.* Proc. of Workshop on OS/Networks of AV data, Nov. 1992.

[12] Steinmetz, R.: *Synchronization Properties in Multimedia Systems.* IEEE Journal on Selected Areas in Communications Vol.8 No. 3 April 1990.

[13] Tamura, H. and Yokoya, N.: *Image Database Systems: A Survey.* Pattern Recognition Vol. 17 No. 1 1984.

[14] Wallace G.K. *The JPEG Still Picture Compression Standard.* Communications of the ACM, April 1991, Vol. 34, No. 4, 30 -44.

[15] Woelk, D., Kim, W. Luther, W.: *Multimedia Applications and Database Requirements* Proc. IEEE Computer Society Symposium on Office Automation, April 1987.

[16] Woelk, D., Kim, W.: *Multimedia Information Management in an Object-Oriented Database System* Proceedings of the 13th VLDB Conference, Brighton 1987.

المنارة للاستشارات

www.manaraa.com