INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center Street • Suite 600 • Berkeley, California 94704 • 1-510-642-4274 • FAX 1-510-643-7684

The Impact of Multimedia Data on Database Management Systems

Karl Aberer and Wolfgang Klas†

TR–92–065

September 1992

Abstract

This paper analyzes the impact of multimedia data on database management systems and proposes some solutions which allow for a high degree of integrated handling of multimedia data by a multimedia database system. We first give a characterization of multimedia data with respect to issues like time dependency and amount of data. Then we derive major requirements which need to be satisfied in order to provide the integration. These requirements include e.g., dynamic data management, non-transparent parallelism, scheduling, several kinds of abstractions, resource distribution transparency, and advanced interaction models satisfying real time constraints. We show how some of the requirements can be met by exploiting concepts from the object-oriented paradigm and database systems. Then we discuss extensions needed with respect to data integration, scheduling, parallelism, and real time streams.

† On leave from GMD-IPSI, Dolivostr. 15, D–6100 Darmstadt, Germany; e-mail: klas@darmstadt.gmd.de

1 Introduction

Multimedia data (often also called continuous media) extends the set of alphanumeric datatypes available in conventional computer systems in two ways. First, by a new dimension of complexity, stepping from simple symbolic data, to complex symbolic and "real" data. This includes graphical, pictorial, and textual data. Second, by a new temporal dimension correlated with audio and video data. These two dimensions draw serious consequences on the integration of multimedia data with conventional data. The problems to be solved include issues like the degree of integration of multimedia data into the DBMS, the internal architecture of a multimedia DBMS (storage subsystem for continuous data, interfaces to special devices like magneto optical discs and compact discs, compression techniques, etc.), data model and data manipulation language extensions (multimedia data types, formats, presentation, etc.), indexing techniques for continuous media, and the handling of digital formats as well as analog sources. Current approaches in the field of database systems, database integration, and database interoperability provide only limited solutions for integrating continuous media. But as solutions are becoming available with respect to the limited availablility of appropriate digital video/audio hardware which can be integrated into conventional computing environments based on workstations, of standards for compression techniques, of some basic support at the levels of operating systems, networking, window systems, and application toolkits, one can approach an integrated solution for multimedia database systems which should incorporate the services provided by the other system levels. Such an approach should be based on appropriate architectures and should not be limited to add-on solutions as those may only partially satisfy the requirements of applications.

In this paper we analyze the "nature" of multimedia data, give a characterization and classification, and derive and discuss a set of essential requirements imposed on database management systems by multimedia data. We propose solutions for particular design issues of a database management system which are able to meet these requirements by focusing on data modelling, database functionality, and some additional concepts needed. Although many of the problems are related to other system levels like operating systems and networking too, we do not discuss the impact of integrating the handling of multimedia data into database management systems on those levels.

Solutions for multimedia database systems range from quite simple ones which *do not* address a *real integration* of the handling of multimedia data into the services of a database management system over those providing a kind of *pseudo-* or *semi-integration* to those which address a *real integration.*

In the first case the user of a (database) programming language or database model constructs a multimedia information system by writing an application on top of the database system. By doing so he does not get any support from the database management system for handling multimedia data. That is, he may just store file names or other annotational information in a database which refers to the multimedia data stored and handled outside the control of the database system. This is for example always the case when a database management system does not provide for arbitrary

user defined functions and when external equipment like video recorders, cameras, laser discs, etc. has to be controlled in order to handle analog video or audio.

In the second case the database system provides for a pseudo- or semi-integration of multimedia data by means of some trivial built-in data type like *binaryLargObject* which allows to store multimedia data in some general digital form, but which still leaves all the responsibilities of how to interpret, manipulate, and present the data with the application programmer.

In the third case in which the database system is able to handle multimedia data in an integrated manner with respect to its characteristic the user gets all the support he needs to store, manipulate, and present multimedia data by using a database system. Note that such a solution does not exclude application development as required in the previous two cases, but it provides the optimum support for dealing with multimedia data as application design should not be forced to deal with the implementation of basic multimedia concepts and modeling primitives. Hence it provides relief for application implementation, but it also supports standardized semantics of basic concepts of multimedia data. This case is comparable to the step from using a relational database system to using an object-oriented database system: a (truly) object-oriented database model allows to capture more semantics of data and operations in a database than the relational model.

The proposed solutions presented in this paper are intended to eventually lead to a multimedia database system which provides integrated management of multimedia data. It is based on a careful analysis of the requirements of multimedia data. It turns out that the central problems are related to (a) time dependency, and (b) the huge amounts of multimedia data. First we discuss the use of already available concepts and techniques. Our approach exploits the object-oriented paradigm as it provides proper semantic modelling concepts needed to deal with many requirements which are consequences of (b). We illustrate this by using the open object-oriented database model VML [15] of VODAK. We show that utilizing database management functionality (persistent, consistent management of data for multiple users) can cover some of the requirements derived from (b). Second, we focus on extensions of the data model and the database functionality. These include mechanisms within the data model of VODAK and extensions of it to integrate multimedia data types up to different degrees. To meet requirements derived from (a) we introduce mechanisms for scheduling, show up some prospects to introduce parallelism, and analyze ways of how to integrate the abstraction of real time streams.

There is several related work which addresses basic solutions in this particular field: Recent results in coding and compression techniques [24] as well as the establishment of standards like JPEG [28] and MPEG [19], now lead to more appropriate digital video and audio technology and, hence, will pave the way for advanced multimedia applications. Managing continuous media requires also appropriate support at several system levels: Operating systems should provide appropriate basic concepts and abstractions for multimedia file management [23], for parallelism [22], for window management [4], synchronization [5]. Further aspects are real-time requirements (e.g., transmission and de/compression of data [26]), scheduling of resources [2], and architectural issues [14]. For an overview of related work in the field of operating systems and net-

working see [10]. Additional aspects include programming and query language support (e.g., [8]). Furthermore, experiences made with hypermedia systems like *NoteCards* [12], [13], *Neptune* [7], *Intermedia* [11],[21], and *KMS* [1] as well as first versions of prototypes of multimedia database systems like ORION[27] and VODAK [16] may also influence the overall design of multimedia database management systems with respect to their functionality.

The paper is organized as follows: Section 2 characterizes and classifies multimedia data and gives an analysis of requirements with respect to multimedia database systems. In section 3 we analyze and show how to meet these requirements with respect to database mechanisms, object oriented programming, and multimedia extensions.

2 Analysis of Multimedia Requirements

In this chapter we first give a characterization of multimedia data. Then we derive and discuss requirements for multimedia computing focusing on the impact of multimedia data on database management systems. We do not claim that the list of requirements is complete and all the requirements are equally important. However, we believe that we can cover the most central points and we understand this list as a starting point for conceptually well founded development of multimedia database management systems.

2.1 Characterization of Multimedia Data

Very often systems are defined as *multimedia* systems by enumerating in lists data like motion video, audio, still images, graphics, animation, text, etc. they are able to deal with. This typical list of so called multimedia data reflects the fact that very little attention is given to the characteristics behind these kinds of data. By looking closer to the different kinds of multimedia data one can differentiate and classify them according to specific criteria.

The most significant features of multimedia data come from the observation that its representation can be much closer to the physical or a virtual physical reality than the usual alphanumeric data which in general is used to represent symbolic information. For example, a video is a recording of the visual information[1] over a period of time at a point in space.

Due to the unique role of time in physics, the most striking classification of multimedia data can be made as either *time dependent* data like audio, video, and animation or *time-independent* data which includes data types like text, still images and alphanumeric data types. Time dependent data has only a meaningful interpretation with respect to a constantly progressing time scale. A time scale is needed to associate with a time dependent data its correct interpretation at each point of time expressed by the atomic constituents of the data. We call this kind of data *dynamic*, more precisely *dynamic in time*. We further can distinguish between data with an *absolute time scale*

1. Namely a photon count over a discrete pattern in space, time, and energy.

which have a canonical correspondence to real world time and data with a *relative time scale* which is to ensure the correct chronological sequence of the atomic constituents, but leaves the rate of progression open. In contrast we will call time independent data *static*, more precisely *static in time*. The atomic constituent of the data is the data itself.

Examples: A video recorded with a camera has a canonical mapping to real world time. The atomic constituents are called *frames* which correspond to intervals of equal length on the time scale of the video. The length of the interval is determined by the recording speed, e.g., 30 frames per second. An animation also consists of frames, but since there is no canonical mapping to the real world the time scale may be deformed, for example to speed up or slow down the animation without affecting the natural meaning of the animation.

Note: The relationship between the atomic constituents of a dynamic data may be more intricate than it may appear at first glance. One has also to think of compression techniques for digital video or audio which use interpolation methods such that the representation of the data goes beyond a sequence of independent frames or samples.

To distinguish from this is now whether data is *static* or *dynamic in size*, i.e., whether one atomic constituent of the data requires a constant or variable amount of storage. Although data dynamic in size plays already a role in e.g., commercial data base systems, e.g., lists, text, they are much more common among multimedia data.

Examples: An integer is static in time and size. A text field is static in time and dynamic in size. Uncompressed digital video is dynamic in time and static in size, while compressed digital video may be dynamic in time and size depending on the compression technique.

Due to the closeness to physical reality a further characteristic of multimedia data is its high density of information in one datum. This holds already for data like pictures and text, but even more for data dynamic in time. Therefore, we want informally distinguish between data with potentially low and high information content. Potential, because data with high information content in general is also highly redundant.[2]

2.2 Requirements

This subsection presents the major requirements which need to be satisfied by the kind of multimedia database management system we proposed earlier. A summary of all requirements is given in the appendix.

2.2.1 Amount of Data

Due to high density of information in multimedia data the amounts of data can be huge. As long as Data is static in time and size like symbols, pictures, or images no serious problems in terms of

2. Note, that there are many aspects of redundancy of course, e.g., with respect to the users model or the internal representation.

processing speed are imposed on networks, on storage devices, and on main memories of current computer technology. Also data dynamic in size can be handled efficiently by using the abstraction of files as provided by operating systems. Serious problems with such data occur only in connection with applications where extreme high numbers of data elements are involved, e.g., processing satellite images for weather forecast [25]. We do not address these problems herein.

On the contrary, data dynamic in time inherently leads to huge amounts of data for single data elements. For examples see the following table:

Voice quality audio	64 Kb/s
CD quality audio	1.4 Mb/s
true color	24 bits/pixel
motion video	30 frames/s
NTSC quality video (512 x 480, 8 bpp)	1.92 Mb/frame
HDTV quality video (1024 x 2000, 24 bpp)	48 Mb/frame

Additionally to the analogous problems mentioned above now we have to deal with this huge amount of data under real time constraints. This has on one hand serious consequences to the design of hardware, operating systems, and networks, but it also must be taken into account when designing a multimedia database system.

When dealing with this data it may be convenient or even necessary to perform the processing not on the data values themselves but on the references to the values. A good example for this is video script editing.

Note: Since these references are not necessarily provided a priori they may have to be created by the user or by the system. We address this problem later in subsection 2.2.5.

Certain applications of dynamic data may need operations which cannot be performed over references, e.g., copying, but also cannot be executed in the standard way as for alphanumeric data because it exceeds the physical resources. In this case some form of dynamic data management has to be provided which a kind of spreads the process over time such that at each distinct moment only a limited amount of physical resources are needed. Since these kind of dynamic operations heavily affect the behavior of a system they must not be transparent. For example, they last for a considerable amount of time or block certain resources.

The requirements determined in this subsection can be summarized as follows:

R1 *appropriate referencing mechanism*

R2 *dynamic data management of very large objects*

2.2.2 Temporal Aspects

When processing dynamic data typically parallel tasks occur. This comes from the nature of this data since in contrast to processing static data operations take non negligible periods of time. Also it often is necessary to process data in parallel.

Example: An application plays back a video on a screen. Simultaneously it gets the audio from a different device and allows for user interaction to control presentation, e.g., to associate annotations at certain points to the video without interrupting the video presentation. This includes parallel tasks for playing back the video and audio, for the user interaction, and for processing user input. The application programmer must be able to control the synchronization of these tasks.

Although parallel execution of applications is supported by database management systems, it is considered to be transparent to users. Hence, database management systems do not explicitly and efficiently provide concepts for the control of parallel tasks by the user[3]. Some tools have to be provided to the user which allow him to explicitly control the parallel execution of different tasks.

There are basically the following alternatives to control parallelism of tasks: establishing relationships between tasks (relative scheduling), e.g., two tasks have to be executed simultaneously or a task can trigger another one, or placing events on a time scale (absolute scheduling), or combining both, e.g., start a task at the next full hour when another task has finished. Both alternatives require appropriate concepts which allow a programmer to express such schedules.

We summarize the requirements determined in this subsection as follows:

R3 *non-transparent parallelism*

R4 *scheduling mechanisms*

2.2.3 Resources

Many different physical devices are involved in processing multimedia data because of the fact that one standard device cannot handle all kind of multimedia data. There are several reasons for this: special purpose hardware (e.g., compression chips, equipment for analog or digital video/audio, presentation devices like loudspeakers, monitors, and windows), efficiency (although it is possible to store digital videos on hard discs, it may be much more efficient for retrieval to store them on laser discs), space requirements (a few minutes of digital minutes easily fill up any standard hard disc).

These devices can range from physical devices with their corresponding device drivers over devices which come with all kinds of special software to devices hidden by other database systems. Although these different types of devices and their behavior should be made transparent as far as possible to the application developer, some of their characteristics should be made visible as far as necessary. Therefore an abstraction mechanism is needed for *device transparency.*

Some of these devices may often be used only by a limited number of applications at the same time. Therefore, appropriate mechanisms to *share* these *resources* upon applications have to be provided.

---

3. It is of course possible to coordinate tasks by using the database as a coordinating medium.

By *classifying* them into appropriate hierarchies and groups one can reduce redundancy and modelling is such more efficient. Since the available devices evolve continuously it must be possible to integrate them in a simple and efficient way without affecting existing systems. This can be achieved by employing the well-known *modularization principle*.

Since there may be so many devices involved in a multimedia application the same data can reside on different devices. This should be made transparent for the application programmer, therefore an mechanism for *data distribution transparency* is needed.

Multimedia applications have to interact with these devices simultaneously, maybe over long lasting periods of time, or on the basis of interrupts. We will cover the requirements derived from this fact in 2.2.7 because these aspects are closely related to user interaction.

We summarize these requirements as follows:

R5 *device transparency*

R6 *resource sharing*

R7 *modular device classification and modelling*

R8 *data distribution transparency*

2.2.4 Data Representation

Any data stored in a computer is a representation of some "reality", e.g. a physical reality like visual information, virtual (physical) reality, mathematical reality like a geometric model, organizational structures like account information at banks. As explained earlier multimedia data primarily emerges from representations which are close to physical realities. The representation by alphanumerical data is straightforward, and formatting problems are already mostly settled for this kind of representation. Also the operating systems guide mostly the way by providing some standard set of datatypes. This is, at least today and in the near future, not the case for multimedia data.

The basic datatypes like the alphanumeric ones are not appropriate to reflect the strucuture of multimedia data. New built-in datatypes like *bitmap* or *audiosample* have to be provided. Furthermore, type constructors taking the temporal nature of multimedia data into account will be needed in some form. Additionally, appropriate support for processing these data types have to be provided. Similarly to the standard operations associated with alphanumeric data (e.g., add integers, concatenate strings) operations like interactive editing videos, playing back and synchronizing videos and audios.

But there is another aspect which has not been considered yet: while it does not make sense to use many different formats for the same alphanumeric datatypes like integer, float (on the contrary it is confusing and dangerous, as one can experience from *C*) this is crucial for multimedia data for the following reasons:

(1) Different compression techniques may be appropriate for different applications. Each format can in principle converted to another. Different resources may need different formats. Due to the high degree of hardware dependency of multimedia data proprietary standards are more likely to emerge. One has to take this into account as de-facto standards and has to provide for the proper openness of a system. The system must provide for a modular and efficient representation of these different formats and standards but it must also be able to make this transparent to the user.

Example: Encoding of still or moving images is different in nature. For videos with little dynamics differential compression may be appropriate.

(2) The internal representation may not be appropriate to be presented to the user (in contrast for alphanumeric data the representation to the user is close to the internal). So special representations for different users to provide different views of the same data may be needed. These may be generated on the fly or be stored persistently (e.g. as a result of a query).

From the above we can derive the following requirements:

R9 new built-in data types & operations

R10 modular and efficient representation of different formats

R11 transparency of data representation

R12 different views for the same data

2.2.5 Modelling

Representation of multimedia data encodes the physical reality (as explained in section 2.1) and hence is a very low level representation formalism. This leads to the problem of huge amounts of data as discussed in 2.2.1. As mentioned there references to the data which also can be understood as abstractions of the data are needed to efficiently process it by avoiding copies. More complex abstractions like references enriched with more information than just used for identification can be used to index data to provide for fast access.

Another reason for introducing such abstractions is to allow the user to refer to the data in terms of abstractions which make up his model of the application domain. These abstractions may be provided by the user or by the system based on the contents of the multimedia data. It can be very reasonable to store these derived abstractions since their computations may be very expensive. For the retrieval and organization of the multimedia data it should be possible to provide several layers of abstractions.

Examples: Assume we have a database of videos. A first possible layer of abstractions would be to identify single scenes in videos such that several abstractions may be provided for a single video. Another layer could be used to identify geometric objects in these scenes, and in a further layer the geometric objects could be related to real world entities in the scenes. A user may search for such

entities based on attribute values stored elsewhere in the database, and so, may access a video in which this entity occurs using indices at each layer.

In order to have these features available in a multimedia database system appropriate solutions have to be found for

R13 indexing mechanisms

R14 semantic and consistent modelling of abstractions.

2.2.6 Database Management Functionality

Many of the needs discussed so far apply for multimedia *programming languages* as well as for multimedia *database manipulation languages*. The need for database management functionality for multimedia data comes besides from the usual reasons from the nature of multimedia data. Multimedia applications deal with persistently stored data because of the huge amounts of data already for single objects, and the processing of the multimedia data very likely requires secondary storage. Few exceptions can be found in real-time applications like video-phones or video-conferences.

Very often abstractions as discussed previously in subsection 2.2.5 are based on derived data and, therefore, database management functionality is needed to maintain consistency between original data and derived data.

Besides the usual reason for multi-user support, namely consistently sharing data among several users, there is the aspect of efficiency for sharing data which is significant for storing multimedia data since it makes no sense to make and to maintain copies of the same multimedia data.

Thus there is a need for the typical functionalities like transaction management, query languages, data dictionaries, etc., but due to the characteristic of multimedia data discussed so far these have to be adapted or new concepts have to be developed. The individual major requirements can be summarized as follows:

R15 persistent secondary storage management

R16 consistent management of derived data

R17 efficient sharing of multimedia data among applications

2.2.7 User Interaction

User interaction is much more complicated when multimedia data is involved. For example, input devices like microphones, cameras may be used additionally to keyboard and mouse for speech and gesture recognition or output devices like windows, monitors, loudspeakers, and VCRs could be involved. Thus the interaction takes place simultaneously over different media which needs (1) simultaneous control of different devices, (2) handling interrupts from users, and leads (3) to long

lasting interactions. In the presentation as well as retrieval of multimedia data different modes can be used to control the quality of output and input, e.g., different resolutions and speeds, image stabilization, browsing. While not all of these techniques and their support will be integral parts and central goals of multimedia database systems they have to support any developments which will take place in these directions.

The major requirements with respect to user interaction can be named as follows:

R18 appropriate simultaneous device interaction (see also subsection 2.2.3)

R19 efficient (real time) handling of user interaction

R20 appropriate model for long lasting interactions

R21 support for advanced user interfaces

3 Building Blocks for Multimedia Database Systems

In this section we analyze and show how we can meet the requirements presented in the previous section in order to get solutions for the design and implementation of a multimedia database management system. In general, we conclude from the topic *database management functionality* (see 2.2.6) that there is definitely a need for *multimedia database systems*. Topic *resources* (see 2.2.3) and topic *modelling* (see 2.2.5) suggest that the most promising starting point are object-oriented databases. Topics *amount of data* (see 2.2.1), *temporal aspects* (see 2.2.2), *data representation* (see 2.2.4), and *user interaction* (see 2.2.7) are those for which presently no adequate database systems solutions are available and where most additional research will be needed. Therefore, in our approach we will first exploit concepts and techniques from the fields of object oriented programming/modelling and database management systems, and, second, we will introduce multimedia extensions needed to meet requirements which are not covered so far.

3.1 Object-Oriented Paradigm

The object-oriented paradigm provides several useful concepts which we can exploit to meet particular requirements. This is also the reason why most approaches for multimedia database systems follow object-oriented principles (e.g., [8], [18], [27]).

Object identity leads to a referencing mechanism (R1) provided that object identifiers are made explicitly available in the data model.

Encapsulation, message passing: Encapsulation is one fundamental building block for (static) transparency of definition because it allows to hide details of implementation from the application programmer and provide uniform method interfaces. *Message passing* is one fundamental building block for (dynamic) transparency of execution because it allows for a separation between method interfaces and method implementations during runtime. Both mechanisms contribute to R5, R7, R8, R10 and R11.

EXAMPLE: In VODAK the concept of semantic relationships allows for delegation of messages to other classes. This can be used to implement data distribution transparency (R8) by assigning the same multimedia object to different storage devices. In runtime the system decides to which device to pass on a message call requesting a representation of this object.

Class taxonomy and inheritance: they allow for reusability of definitions and implementations by sharing common parts and are the tool to provide an ontological order of the application domain. This contributes to R7 and R10.

EXAMPLE: We give an example of how to realize resource integration by modelling multimedia devices in a class hierarchy. We give the object types defining the class interfaces.

```
OBJECTTYPE LaserDisk_Type
        INTERFACE play() Stop() position() //only signatures

OBJECTTYPE Sony–LaserDisks_Type SUPERTYPE LaserDisk_Type
        INTERFACE       repeat() //only signatures
        IMPLEMENTATION  play() {//use repeat}
                        position() {//use repeat}

OBJECTTYPE SonyLD–SVC77 SUPERTYPE Sony–LaserDisks_Type
        INTERFACE       index()
        IMPLEMENTATION  repeat(), index(),new()
```

This type hierarchy reflects several layers of abstraction with common interfaces. We assume to have a common interface for all Laser–Disk players. This may then be enriched for devices from a specific producer which share some common functionality and which is then refined to the type corresponding to a specific model. This is the common technique to ensure certain interfaces, organize code etc..

Views: To distinguish from a hierarchical organization of interfaces and code in order to ensure certain access to objects is the problem of making inhomogeneous interfaces transparent to users who are not interested in the details. An important mechanism to do this (which is still not well understood in the framework of object–oriented modelling) is the concept of views. Views are important to provide the right level of abstraction of the resources, data distribution and data representation (see R5, R8, R11). Views are also important to provide different views of same data (R12), as, e.g., to realize the often mentioned distinction between internal and external representation of data.

EXAMPLE: A user may be interested in a very abstract view of a device, like a laserdisc, while the application programmer may need some more details, like formats, and the person responsible for the integration of the device needs full access to all details.

After devices like laserdiscs and VCRs have been made available an application designer

wants to provide a posteriori a common view of these to a certain group of users.

```
VIEW VideoDevice
    CONTRIBUTORS INTERFACE play();
    INTERFACE forward(); reverse();
    IMPLEMENTATION   forward(); reverse(); // uses play()
    INIT: LaserDisc->participateInView(VideoDevice);
        VCR->participateInView(VideoDevice);

CLASS LaserDisk ...
CLASS VCR ...

CLASS CDI-Player
    INIT: participateInView(VideoDevice)
```

The interface defined in CONTRIBUTORS INTERFACE represents the minimal interface necessary for a class to contribute to this view. It should describe the minimal functionality needed for the implementation of the view. The clause INTERFACE gives the methods visible to a user of this view.

Note: In object–oriented systems, which do not provide a subset semantics of subclassing, the concept of views cannot be simply mapped to the concept of classes. This is because the extension of the superclass cannot contain objects which are instances of subclasses. In object–oriented systems which provide a subset semantics of subclassing, one can simulate views to some extent under the following serious limitation: the extension of the superclass is defined to be *exactly* the union of the extensions of the subclasses, and the resulting view is union and projection of the subclasses. But it is well recognized that views are much richer in their semantics as it is ,e.g., orthogonal to the notion of class taxonomy. In VODAK it is proposed to use the concept of semantic relationships and message inheritance via those to realize views.

The object-oriented paradigm is the basis for many systems which provide the appropriate and powerful mechanisms for semantic modelling (R14). In VODAK this mechanism is provided by the concept of semantic relationships. It has already proven its powerful modelling capabilities in many applications, e.g., [16], [17] [18].

3.2 Database Systems

Database management systems are a successful approach to work with huge amount of data on persistent media, in a shared environment (R15, R17). They provide services for, e.g., locking, recovery and concurrency. Many other functionalities of database management systems can be

used to meet several requirements mentioned in section 2, but also some extensions are needed. We want to go now into some non–standard applications of basic database services with respect to multimedia data and some extensions of database services needed .

Devices which can be accessed only by a limited number of users at a time are typical for multimedia applications (R6). By modelling devices as data in the data base, the transaction management can avoid conflicts occuring through simultaneous access to these devices, e.g., by providing appropriate locks.

Since transactions on multimedia objects may be time-consuming there is a need for cooperative and long-lasting transactions in order not to lock unreasonable large parts of the database for other users (R20). This is already the issue of an own area of research (for details see [9])

Another major issue in database management is to maintain the consistency of the data. This is especially important for multimedia databases, since for several reasons much derived data is present (R13, R16). Constraints can be used to maintain consistency between derived data and underlying raw data.

Query and retrieval mechanisms may be needed to be adapted according to new or modified features of a database system. New concepts like visual indexing and access methods based on abstract models of the data may be needed to be introduced.

Techniques from the field of distributed databases (e.g., partitioning of schemas, management of replicated data, query processing, data distribution transparency) may play a significant role. They may be adapted and extended (but we do not go into details).

Since the dispersed resources for multimedia applications may be considered as heterogeneous databases techniques from this area may be useful to be able to provide transparent access to different kinds of databases (different formats and devices). For example, a video database may contain analog, digital, compressed or uncompressed videos, and a user of such a database should not have to deal with the different formats. One and the same video is stored in different formats, e.g., in analog PAL and in NTSC digital. A user should not be forced to distinguish between these different formats, but the system should know about the availability of both formats and should choose the right one for specific processing.

3.3 Multimedia Extensions

In this section we will now focus on the extensions needed to provide appropriate support for multimedia data. These extensions will focus on the problem of integrating multimedia data types and on real time management.

3.3.1 Data Integration

In order to model multimedia data properly new static datatypes are needed (see R9). There are basically three different approaches to data integration, depending on how far the data is technically integrated into the computer system.

(1) *New built-in data types*:
In this case, specific data types like *Audio, Video, Animation* are predefined and built into the system. They are available as language constructs and the user can use them like other well-known built-in types. But this solution makes only sense when the platform supports explicitely standard formats as it is today the case for most of the alphanumeric data. That is, in the case of a database management system, the operating system should provide appropriate support for e.g., video and audio data. Then the data base model can provide built-in data types based on the functionality of these underlying data types.

(2) *Integration via data types or classes*:
As long as the platforms on which multimedia applications are running are not supporting some standard formats for multimedia data, as it is the case for alphanumeric data, some type of raw format like *binary large object* or *byte sequence* are appropriate. The first alternative we consider is to construct a multimedia datatype as an abstract data type describing the functionality of the type. The data itself is encapsulated in the raw format. This approach is useful if we want to manipulate the data as transient values.

EXAMPLE:

```
ABSTRACT DATATYPE bitmap_jpeg
    compressedData: ByteSequence
    OPERATIONS      decode(p: bitmap_jpeg) : X11WindowBitMap;
                    encode(p: KodakCDPicture) : bitmap_jpeg;
                    cut(p:bitmap_jpeg, x, y, l, w: INT) : bitmap_jpeg;
                    width(p:bitmap_jpeg) : INT;
                    length(p:bitmap_jpeg) : INT;
END
```

An alternative approach is to introduce such types as classes. Depending of whether abstract data types are available or not we have two possibilities to define these classes. The first using the previous definition of *bitmap_jpeg*

EXAMPLE:

```
CLASS SatellitePictures
    PROPERTIES:   picture: bitmap_jpeg;
                  recoredAt : DATE; ...
    INTERFACE:    decode() : X11WindowBitMap;
                  encode(p: KodakCDPicture);
    IMPLEMENTATION:
        decode() : X11WindowBitMap; { return decode(picture); };
        encode(p: KodakCDPicture); { picture := encode(p); }
END
```

One disadvantage of this approach is that we have to provide trivial implementations for the bitmap_jpeg interface functions based on the abstract data type bitmap_jpeg operations. We

can avoid this on the cost of loosing transient values for bitmap_jpeg by implementing bitmap_jpeg itself as a class and exploiting inheritance.

EXAMPLE:

```
CLASS bitmap_jpeg
    PROPERTIES: compressedData: ByteSequence
    INTERFACE:
            decode() : X11WindowBitMap;
            encode(p: KodakCDPicture);
            cut(x, y, l, w: INT) : bitmap_jpeg;
            width() : INT;
            length() : INT;
END

CLASS SatellitePictures SUBCLASS bitmap_jpeg
    PROPERTIES:   recoredAt : DATE; ...
    INTERFACE:    ...
END
```

Note: as stated in requirement R1 large data may be handled by references. Such a referencing mechanism is given as long as such data is modelled as classes since object identifiers are assigned to each instance of a class and this identifier is explicitly available in the language. If we introduce a referencing mechanism (pointers) on the level of abstract data types (as it is usual to handle large objects in programming languages like C/C++) and we use these pointers as properties of classes in the database, then the actual property values of the database objects are no longer under the control of the database management system as long as no extensions are provided for this situation.

(3) *Modelling as real world entities*:
However there are many cases where data integration is not possible, simply because the computer system has no direct control of the data. In this case there remains only the possibility to consider the data as real world entities which are modelled (and controlled) by the database systems, but for which consistency between the real world and the database world has to be maintained by the user.[4] For example consider a VCR with a collection of tapes stored in some cabinet maybe controlled by some mechanical device. But the data is never controlled by the database system.

---

4. This is very similar to the situation where in a standard database application like as address database, the correctness of the address has to be maintained constantly by updating the data (manually).

3.3.2 Extensions For Real Time Management

One of the most important gaps between todays programming paradigms used for database systems and the needs of multimedia data is that concepts related to real time behavior are not present. We showed the necessity of concepts related to real time behavior with requirements R2, R3, R4, R18, and R19.

In order to provide for scheduling of different tasks on multimedia data (R4) , appropriate simultaneous device interaction (R18), and efficient handling of user interaction (R19), we propose a built-in language construct for scheduling. We also discuss some of the consequences for the message handling in an object-oriented language.

Scheduling allows for the modelling of non-transparent parallelism (R3). So we have to discuss possible realizations of parallel events in a multimedia database system. Our intention is not to propose a parallel programming language, but to integrate parallelism in order to satisfy the requirements identified earlier.

An important abstraction which is useful for the dynamic management of dynamic objects (R2) is that of real time streams. We show how these can be modelled by taking advantage of the scheduling mechanisms introduced earlier.

(1) *Scheduling*

It is obvious that for the integration of the presentation aspects into the services of a database management system, some kind of scheduling mechanism will be needed. Less obvious, scheduling mechanisms are also needed for the internal manipulation or updates on the data, which in turn are intertwined with presentation aspects. This kind of scheduling has to be reflected in the database manipulation language and goes far beyond the traditional concepts like message passing for object oriented databases. Scheduling includes triggering events at certain points in time, upon other events, before deadlines and other similar operations. Without analyzing all possible kinds of features of a scheduling mechanisms we want to concentrate on the fundamental aspects.

Two problems can be identified when introducing a scheduling mechanism:

(1) conciliation with the object oriented message-passing paradigm, and

(2) conciliation with the transaction management.

We propose a short outline of such a scheduling mechanism by presenting the basic components which solve these two problems:

The Global Clock:
Since multimedia data are time-dependent our scheduling mechanism is based in a clock. This is the basis for *absolute scheduling* on the time scale. A tick of the clock is considered as an event. In order to be able to reconcile schedules in a distributed environment the clock should reflect a system wide world time. We propose that the clock is realized by a built-in function *currentTime()*.

The Event Language:
In order to express time relationships one has to introduce appropriate language constructs which form the basis for *relative scheduling* of events. This aspect has already been recognized and several temporal interval based schemes have been proposed e.g., [20]. One should introduce language constructs for building time expressions based on such schemes.

The Schedule:
A schedule consists of a set of messages (which are method calls or schedule calls) at prespecified points of time. Schedules are defined for classes like methods. They are part of the interface of a class. Schedules can be called by sending a message to an object like for methods. In contrast to methods a schedule call *never* returns any result value.
Example:

```
CLASS X
 INTERFACE
   METHODS  m1(...) : ...    // some methods definitions
   SCHEDULES
    m(a, b, c) {
       VAR x, y : T          // some variable declarations sequence
       x := ....             // some initial computations
       y := ....             // some initial computations
       v := x–>m2(a, y, ...) AT time-expression  // this is a message to be sent
                                                 // at time time-expression
       ... }
```

Like a method, a schedule may take arguments (a, b, c). The body of a schedule consists of an *init sequence* (variable declaration, initial computations) and a set of *scheduled messages* where the schedule time is expressed by the *AT* construct and the *time-expression* is formed by using constructs from the event language and the clock call.

The Message Handler (Semantics of calling a message):
Let us assume that the *Message Handler* is the module which executes messages sent to an object. It checks whether the selector *m* in a message *o–>m(arg1, arg2, ...)* identifies a schedule or a regular method. If *m* is a schedule (we call this a schedule call), the Message Handler sends a request to *register* the schedule *m* with its actual parameters to the *Scheduler* (a module we describe later). No value is returned from this message call which corresponds to executing a methods which does not define a return value. Otherwise the Message Handler executes the method *m* as usual [15].

The Scheduler:
The Scheduler maintains a list of registered messages, and a list of schedule scopes. A schedule scope has associated all its *scheduled messages* and all the variable bindings of the variables declared in the *init sequence* and of the parameter list of the schedule. The list of registered *scheduled messages* is based on a time scale over all registered schedules.

The functionality of the scheduler is conceptually separated into two phases: the registration phase and the realization phase.

The Register Phase:

When the Scheduler receives the request to register a schedule from the Message Handler, it builds the *schedule scope*. A schedule scope consists of all the *scheduled messages* and the (parameter) variable bindings. The actual arguments passed with the schedule call become part of the variable bindings. An *init sequence* of a schedule is executed after the parameters have been put into the variable binding. The execution of the *init sequence* can establish or change the variable bindings.

The Realization Phase:

The scheduler continuously chooses registered messages from its schedule list and requests the Message Handler for executing the message. Results of messages (which are in this case always method calls) will change the variable bindings of the corresponding schedule scope. In case the message is a schedule it is sent to the Scheduler as explained above.

A major advantage of this concept is that we do not need to implement a full parallel language. The scheduler can exploit parallelism if available, otherwise it is simple running out of resources. The approach fits nicely into the object-oriented paradigm, makes use of the object-oriented concepts, no fundamental changes have to be made to the message handler, and no changes are required for the transaction management.

Open issues are the design of an event language, the handling of several parallel running message handlers, and how far a static scheduling is applicable in this context.

(2) *Parallelism*

From our requirements and our proposal of a scheduling mechanisms we observe the following situations where parallelism occurs:

(a) The realization of a schedule involves the execution of parallel tasks in case two messages are scheduled at the same point in time.

(b) External tasks including simultaneous operated resources which may be devices as well as users. We require to model these tasks by internally parallel computations which are active for the same period of time as a resource is operating. This model is in contrast to an approach based on a *global* interrupt handler, since this contradicts the object-oriented paradigm.

(c) Long lasting interactions call for parallelism as tasks may overlap. Similar problems occur in the context of multi-user support.

By allowing for several message handlers which can run in parallel we can realize parallel execution of tasks in a schedule as in (a). Parallel external tasks can be modelled as parallel

scheduled messages. in general, there are two alternative to implement parallel running message handlers: either by processes with separate address spaces or by threads with a common address space. The only feasible solution appears to be the use of threads, since otherwise the overhead is much to high and complex (see also [22]). Specific approaches like check-in/ check-out paradigm may be employed to solve the problem of long lasting interactions (c).

(3) *Real Time Streams*

Streams, especially real time streams, correspond to dynamic management of dynamic data. This abstraction is very useful and will be soon provided as a primitive in operating systems. It should also be made visible to the database programmer. The situation is very similar to that of data integration. As long as there is no built-in support of this concept we do not want to make it a built-in construct of the data manipulation language (because there is no guarantee to get the appropriate operating system support for the language kernel). Therefore, at the present stage real time streams should be modelled by classes. This takes also into account that there are other similar yet in the details different models emerging (e.g.,[6]), and we want to be able to take advantage of these by reflecting them by appropriate classes.

(a) Basic building blocks for real time streams:
The most common abstractions used in the context of real time streams are *Ports, Filters, Sources* and *Sinks*. They can be used for forming complex *configurations* in the form of directed graphs which describe the topology of the stream flow.

(b) Global clock for synchronization:
All components in a stream configuration depend on their real time behavior from a global clock which is equivalent to associating a time-dependent functionality to the entire stream configuration.

(c) Realization of building blocks:
The following two variants of a mechanism which controls the behavior of a stream configuration are conceivable: the data driven approach where sources and sinks dominate, or the control driven approach where ports and filters dominate. Both can be implemented through classes by using the scheduling mechanism introduced earlier.

Note: Streams may not always be instantiated as data flows within a program. For example, if videos reside on an external laser disc (transparently controlled by a multimedia database management system) and the video signal is sent from the laser disc directly to the presentation device via a separate video network, the stream of video frames may never be accessible within the database application program. These cases, when streams are bypassing the database management system, are close to the situation of process control, which again can be readily be implemented using the scheduling mechanisms.

4 Conclusion

In this paper we presented a characterization and analysis of multimedia data which goes beyond the definition of multimedia data often given in literature. We did not just define multimedia data as being video and audio, or a combination of these with other types of data like animations, still images and conventional data types like text and basic alphanumeric data types. We presented a classification of all these data types according to the "nature" of the data. The main criteria were based on time issues and the amount of data involved. This characterization then led to a set of requirements which need to be satisfied in order to provide the kind of integration of multimedia data into a database system which we proposed as an appropriate solution for multimedia database systems. For a summary of these requirements see the appendix. We showed how the object-oriented paradigm and concepts from data base systems contribute to meeting these requirements. For those requirements which cannot be satisfied with these concepts we proposed solutions with respect to data integration, scheduling of (simultaneous) tasks on multimedia data involving devices, presentation, and user interaction, non-transparent parallelism, and real time streams. Open issues not addressed in this paper are the design of an event language, the handling of several parallel running message handlers, and how far a static scheduling is applicable in this context. These aspects are currently investigated and solutions will be implemented on the basis of VODAK.

5 Literature

[1] Akscyn, R., D.L. McCracken and E.Yoder, "A distributed hypertext for sharing knowledge in organizations", *Communications of the ACM* 31, 7 (July 1988), pp. 820–835.

[2] Anderson D.P.: Meta-Scheduling Fro Distributed Continuous Media. Technical Report No. UCB/CSD 90/599, University of California, Berkeley, October 1990.

[3] Anderson D.P., R.Govindan, G.Homsy, and R.Wahbe: Integrated Digital Continuous Media: a Framework Based on Mach, X11, and TCP/IP. Report UCB/CSD 90/566, March 1990.

[4] Anderson D.P., R.Govindan, G.Homsy: Abstractions for Continuous Media in a Network Window System. International Conference on Multimedia Information Systems, January 1991. (also: UCB Technical Report No. 90/596, University of California, EECS, Berkeley, USA, 1990.)

[5] Anderson D.P., G.Homsy: Synchronization Policies and Mechanisms in a Continuous Media I/O Server. Technical Report UCB/CSD 91/617, University of California, Berkeley, 1991.

[6] Cabrera L.-F., and D.D.E.Long: Exploiting Multiple I/O Streams to Provide High Data-Rates. Proceedings of the 1991 Summer USENIX Conference on Multimedia – For Now and The Future, Nashville, Tennessee, June 1991.

[7] Delisle, N. and M.Schwartz, "Neptune: A hypertext system for CAD applications", in: *Proceedings of ACM SIGMOD '86* (Washington D.C., May 28–30). ACM, New York, 1986, pp. 132–142.

[8] Dimitrova N., F.Golshani: EVA: A Query Language for Multimedia Information Systems. Proceedings of the Int. Workshop on Multimedia Information Systems, Tempe, Arizona, Feb. 1992.

[9] Elmagarmid A.K. (Ed.): Database Transaction Models For Advanced Applications. Morgan Kaufmann, 1992.

[10] First International Workshop on Network and Operating System Support for Digital Audio and Video. ICSI Technical Report TR–90–062, ICSI, Berkeley, November 1990.

[11] Garret, L.N., K.E.Smith, and N.Meyrowitz, "Intermedia: Issues, strategies, and tactics in the design of a hypermedia document system", in: *Proceedings of the Conference on Computer–Supported Cooperative Work* (Austin, Texas, Dec 3–5). 1986, pp. 163–174.

[12] Halasz, F.G., T.P.Moran, and R.H.Trigg, "NoteCards in a Nutshell", in: *Proceedings of the 1987 ACM Conference of Human Factors in Computer Systems* (CHI +GI '87), (Toronto, Ontario, Apr 5–9). 1987, pp. 45–52.

[13] Halasz, F.G., "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems", *Communications of the ACM*, Vol. 31, No. 7, July 1988.

[14] Homsy G., R.Govindan, D.P., Anderson: Implementation Issues for a Network Continuous-Media I/O Server. Technical report No. UCB/CSD 90/597, University of California, Berkeley, September 1990.

[15] Klas W. et al.: VML – The VODAK Model Language Version 2.2, Technical Report, GMD-IPSI, August 1992.

[16] Klas W., E.J. Neuhold, M. Schrefl: Using an Object-Oriented Approach to Model Multimedia Data. Computer Communications, Special Issue on Multimedia Systems, Vol. 13, No. 4, May 1990.

[17] Klas W., E.J. Neuhold: Designing Intelligent Hypertext Systems using an Open Object-Oriented Database Model. Technical Report GMD, No. 489, Birlinghoven, 1990.

[18] Klas W.: Tailoring an Object-Oriented Database System to Integrate External Multimedia Devices. International Workshop on Heterogeneous Databases and Semantic Interoperability, Boulder, February 1992.

[19] Le Gall D. : MPEG: A video compression standard for multimedia applications. Communications of the ACM, Vol 34, No 4, April 1991.

[20] Little, T.D.C, and A. Ghafoor: Conceptual Models for Time -Dependent Multimedia Data. Proceedings of the Int. Workshop on Multimedia Information Systems, Tempe, Arizona, Feb. 1992.

[21] Meyrowitz, N: "Intermedia: The Architecture and Construction of an Object-oriented Hypermedia System and Applications Framework", in: *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA'86)* (Portland, Oregon, Sept. 29 – Oct 2) ACM SIGPLAN Not. 21, 11 (1986).

[22] Nakajima L., M.Yazaki, H.Matsumoto: Multimedia/Realtime Extensions for the Mach Operating System. Proceedings of the 1991 Summer USENIX Conference on Multimedia – For Now and The Future, Nashville, Tennessee, June 1991.

[23] Polimenis Vassilios G.: The Design of a File System That Supports Multimedia, Technical Report TR–91–020, International Computer Science Institute, Berkeley, March, 1991.

[24] SPIE – Proceedings of the Conference on Visual Communications and Image Processing, 10–13 November 1991, Boston

[25] Stonebraker M., Jeff Dozier: SEQUOIA 2000 – Large Capacity Object Servers to Support Global Change Research, Sequoia Technical Report 91/1, University of California, College of Engineering, Electronics Research Laboratory, Berkeley, CA 94720, (1991)

[26] Umemura K. and A.Okazaki: Real-Time Transmission and Software Decompression of Digital Vudei in a Workstation. ICSI Technical report TR–91–004, ICSI, Berkeley, January 1991.

[27] Woelk, Darrel, W. Kim, and W.Luther: An Object-Oriented Approach to Multimedia Databases; *ACM SIGMOD Record 1986*, pp. 311 – 325, ACM, 1986.

[28] Wallace G.K.: The JPEG still picture compression standard. Communications of the ACM, Vol 34, No 4, April 1991.

6 Appendix: Summary of requirements

This appendix provides a summary of all requirements given in subsection 2.2.