Towards a Universal Media Server

Ulrich Marder
virtualmedia@ulrich-marder.de

Sonderforschungsbereich 501
Technical Report 03/2000

Databases and Information Systems Group

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

ABSTRACT

In this report, we give an overview of the SFB 501 approach on developing abstractions and concepts needed to achieve the long-term objective of realizing a genuine universal media server. Such a media server is supposed to supplement repositories of any kind (e. g., digital libraries) by providing a high-level interface that allows applications to access and process media data over a network (including the world-wide web).

We focus primarily on the problem of optimally processing raw media data at the server. The traditional data independence abstraction, which is to be realized by a universal media server, is known to be impeding the processing of media data with optimal performance and/or quality. However, instead of abandoning data independence (which is most usual for media servers today), we propose an enhancement that is particularly tailored to the characteristics of networked media data processing: transformation independence. Conceptually, transformation independence can be supported by introducing a virtual media object layer at the server.

The VirtualMedia concept accordingly presented in this paper takes advantage of the common filter graph model to introduce a logical representation of such virtual media objects. Principles and rules for deriving optimal plans to materialize virtual media objects are also considered. We compare our approach with other realizations of media-ADTs (particularly the E-ADT approach) and, finally, draw some conclusions regarding urging problems which have to be addressed in the future.

Keywords

Media Server, Media-specific Abstract Data Types, Data Independence

Table of Contents

List of Figures

1 Introduction

Traditional relational database management systems (RDBMS) are specialized on the management of formatted data. For that purpose, they provide both numerical and alphanumerical data types together with operations defined on these types. The management of media data like images or video, however, has never been a strength of such systems. While many of them offer a generic data type (often called *Binary Large Object* or BLOB, for short) to use for media data, media-specific operations (most preferably integrated into SQL) are generally missing. Hence, it is the applications' job to detect the data format of a media object stored in a BLOB and, subsequently, to decide whether it is possible to perform any necessary operations on the object at the client or not (because, e. g, the format is not known at all).

This resembles very much the situation when media objects are stored in files, and so, it is surely not astonishing that most today's multimedia information systems (MMIS) are rather built on top of file systems than on top of a DBMS. Generally, two different categories of media servers are used in such MMIS. First, generic file servers (like, e. g., ftp or http servers) which are able to deliver virtually any kind of media data, but without providing media-specific semantics, are a very common base for MMIS. Second, another popular class of servers is specialized on a certain media format (e. g. MPEG). This type of media server is somewhat more sophisticated than the first, because they usually exploit their knowledge of the media format to provide special operations for that data type (e. g., typical VCR functions). However, being bound to one special media format is a serious restriction with regard to application neutrality.

While the recent growth of the Internet has initiated the development of large distributed MMIS, application neutrality increasingly becomes an issue to be dealt with. This can be illustrated by looking at digital libraries or at teleteaching environments, which both characteristically have large numbers of users with significantly different profiles. That means, all users regardless of their various roles (manager, administrator, client, librarian, author, teacher, student, etc.) and their manifold hardware equipment (graphical workstation with high bandwidth network, PC with modem, set-top box, etc.) have to be served equally well by the media server(s) on which the MMIS relies. This leads to the idea of a "universal media server" capable of satisfying almost every requirement a multimedia client might pose. Obviously, such a goal would be hard to achieve, and if so, surely not free of costs. However, as multimedia applications grow and diversify, there is probably no alternative to developing more general solutions to a common problem like optimizing concurrent access to large global multimedia databases.

Since there is currently no such thing as a "universal media server" (at least in the sense described above) one could investigate many obvious questions like, e. g., which of today's most advanced DBMS technologies (if any) would serve best as a starting basis for developing the universal media server or which software architecture to choose for it. However, it is our conviction that the most urging issues are the data abstractions and general concepts characterizing the semantics, behavior, and power of a universal media server. In the following two sections, our ideas towards a solution to this problem are presented. Namely, section 2 is concerned with the necessary data abstractions while section 3 introduces the so-called VirtualMedia concept which can be seen as a general framework for realizing the powerful media processing capabilities a universal media server should provide. Section 4 and 5 present related work and conclusions, respectively.

2 Media Data Abstractions

Our first approach to media data abstractions was the so-called *Media-specific Abstract Data Type* (MADT) concept (see, e. g., [KMM94] and [MR97]). The goal of the MADT concept was to introduce new (DBMS-) data types for media objects that provide the same abstractions as traditional "built-in" data types. To achieve this, it would not suffice to merely encapsulate the data. Rather, it is necessary to superimpose the internal structure of the data by an adequate logical structure and then define the operations of the data type on that logical structure. The logical structure of a *Text* data type could be, for instance, a hierarchical structure consisting of words building lines, building blocks, etc. Considering an *Image* data type one could imagine a pixel matrix as logical structure, while a *Video* is usually seen as a sequence of frames which in turn are naturally modeled as pixel matrices, thus being of type Image.

One advantage of this concept is, that the semantics of the data types are explicitly and unambiguously determined by their logical structure (and the operations thereon). Without that, which is the normal case today, the structure and properties of media objects stored in a database system become somewhat non-deterministic, because they are driven by the applications that create and modify the objects, and, hence, determine their physical structure. Unfortunately, letting the DBMS constitute the physical data format (which, by the way, would make finding a mapping between logical and physical structure a trivial task) is also not practical, since the applications' requirements regarding the data format often differ in a wide range or even may be mutually exclusive.

Hence, the question arises whether it is generally feasible (i. e. not a fiction) to create media-specific abstract data types with rich semantics, while supporting yet contradictory requirements from applications. To go on to this problem, in the following subsections, the abstractions needed to find a (hopefully) positive solution are worked out in detail.

2.1 Location Transparence and Device Independence

Location transparence and device independence could be easily achieved by storing media data in databases that are exclusively controlled by a DBMS (e. g. storing media data in BLOBs). However, it is often preferable to use storage locations external to the DBMS, thus enabling:

- **Presentation support:** Presenting continuous media requires the server to perform certain operations in real-time, which cannot be accomplished by (or demanded from) universal DBMSs. Hence, a *Continuous Media Server* acting beside the DBMS must be able to locate the media data on a storage device and access it in bypass of the DBMS.
- **Device support:** There are some special storage devices that are frequently used for media data (e. g. read-only devices like laser disks or live media sources). Such devices are not supported by most DBMS (and will probably never get supported).

Most commonly, to allow storing the media data externally, the storage device and location are deliberately made visible to the client of the DBMS (see, e. g., the IDS/UDO[1] with its Video Foundation DataBlade [Inf97b]). Obviously, this approach takes the burden of being really concerned with media presentation or special device handling off from the DBMS. On the other hand, the applications are forced or at least induced to manipulate the media data directly and to make assumptions on the characteristics of storage devices.

[1] Informix Dynamic Server with Universal Data Option

Since the MADT concept only exposes a logical model of the media, it must not adopt the approach sketched above. Consequently, the MADT concept indeed has to be concerned with any variant of accessing or working with the media including presentations, which makes it hard to realize (but eventually the effort will pay off). As a by-product, the MADT concept also guarantees the stableness and integrity of the media object identifiers (though, this appears feasible, too, when adhering to the first approach [NMB96]).

Certainly, most multimedia applications won't be able to accomplish their tasks by sticking all the time to a merely logical view on the data. Eventually a point will be reached, when exchanging "real" data between the database and the application is required. Considering this situation, another abstraction, known as data independence, comes into play.

2.2 Data Independence

The advantages of data independence base on the distinction between the internal, external, and logical representation of a data type. Only the logical representation (which is used to specify the semantics of the operations on the data type) and the external representation (which is used to exchange instances of the data type between DBMS and application in a format that is well known to the application) are visible to DBMS clients and, therefore, should be stable and best adapted to the applications' needs. Analogously, the internal representation may be optimally designed for storing the data and performing operations within the DBMS. Hence, the internal representation of a data type can be changed or improved, respectively, without any negative effect (regarding performance and quality of service) on existing applications.

Since we have already stated that the MADT concept requires a logical model of the media, it should be clear now that it also demands carefully distinguishing between internal and external data formats. While this would probably be enough to say when talking about simple data types like, e. g., numbers, regarding media data types this matter is somewhat more complicated.

One cause for trouble and sometimes misunderstandings is that media data types usually carry "raw" data and meta-data together. The relation between these two is quite unbalanced, since the meta-data is always needed to interpret the raw data, whereas the meta-data alone by all means may be reasonably interpreted. For instance, the meta-data might absolutely allow answering many queries concerning the content of a media object without (the DBMS) ever examining the raw data. This observation easily leads to identifying the media object with its meta-data, which in turn may be the reason why sometimes people tend to claim media data types being data independent, while, in fact, only the meta-data portion is. Such data types provide data independence only as far as looking at the raw data is avoidable. Real applications will hardly ever be able to follow this strategy, and, hence, eventually get into the trouble of getting served a raw data format which they do not like or—much worse—which they do not even know.

Data independence with respect to the raw media data is usually called *physical data independence* or *format independence* (which we prefer). From the start, providing format independence has been a major concern of the MADT concept. Therefore, it should be stated clearly, what format independence means and how it can be realized.

Obviously, the key to format independence is reconciling the conflictive requirements regarding the internal and external raw data format. Since the internal format only depends on the DBMS, its properties should be optimal for storing and processing the data. This internal format, however, must also guarantee application neutrality, which means: It must not loose any bit of information

provided by the creator of a media object. Such information loss could occur, e. g., by using a DCT-based compression scheme with the internal format. Moreover, information loss occurs as a side effect of many operations on the media data, e. g. clipping, downscaling, and filtering, because there is no inverse operation. Hence, the internal format must be prepared for neutralizing the effects of otherwise irreversible operations.

The external format, on the other hand, ideally depends only on the application. Hence, there might be as many different external formats as there are different applications (though, this will rarely happen). Generally, specifying the external format of a media object not only means determining an encoding scheme (e. g. JPEG for an image object), but also determining a set of quality parameters (e. g. the image size). Thereby, the DBMS is enabled to reduce the data being sent to the application to the minimum required to satisfy the actual quality demands. By the way, this also saves the applications from having to perform trivial operations like downscaling.

To give a little example, consider an application presenting a set of images from which the user may select some for printing. Thus, for the preview on screen this application might choose to retrieve all the images with relatively low quality in GIF format, while in the case of being requested to print out a certain image on a laser printer, it would be retrieved again, but this time as high-resolution, grayscale, PostScript-encoded image.

Concluding these considerations on format independence, it should be mentioned that it undoubtedly demands strong "under-cover" format conversion capabilities and large storage capacities as well. While this appears to form an obstacle at first glance, we would like to subtend that today knowledge on various data formats is replicated in countless applications—just to make them cope with as many formats as possible, however, without adding any substantial semantics.

Having an MM-DBMS providing data independence as illustrated above would be fine. However, it would probably not be very well founded, because format independence unfortunately interferes with MADT operation semantics and optimization issues. To illustrate this problem recall the example above, where creating the external image format involves scaling and color transformations. From a client's perspective these two operations obviously must be only applied temporarily (i. e., without creating or modifying database contents). That's all, in fact—nothing about *where* to execute them (may be on the client machine), which *sequence* to choose (scaling first or color transformation first), or what to do with (intermediate) *results* (store or forget). No doubt, it is easier (and, hence, most common) to design media data types that force the application programmers to deal themselves with such optimization problems, thus demanding to think about not only *what* to do, but also *how* to do it. However, this turns out to be no option for realizing the MADT concept, because it would require the application programmers to specify operation sequences for generating the external data format, which, of course, would make format independence an absurdity. These considerations stimulated thinking about a somewhat broader abstraction leading the way to solving both the optimization problem and the format independence problem. This abstraction, called *transformation independence*, is introduced in the next section.

2.3 Transformation Independence

Transformation independence can be shortly characterized as a way of generally specifying the semantics of (arbitrarily complex) media transformations while abstracting from places of execution, execution sequences (of atomic operations), and persistence considerations (i. e., how, when, where, and how long to store media data in the database which can be generated by applying operations to other media data). Some more explanatory statements on these ideas are given below.

Part of the MADTs is a (not necessarily fixed) set of operations which modify media objects either-way. All these operations are considered showing the same general behavior, namely, taking one or more input streams, processing them, and generating again one or more output streams. Thus, MADT operations may be equally well characterized as a kind of generalized *filters*. These filters are combinable in many ways, though, not all of the combinations would be valid (e. g., it would make no sense applying an audio filter to an image data stream). Hence, there exists a true subset of valid filter combinations which are called *media transformations*. A media transformation may or may not produce a target media type different from the originating media object's type. For instance, it may apply some fancy effects to an image object, thus producing again an image object, while another transformation might create a textual transcript from a speech recording, thus changing the media type from audio to text. One can also imagine transformations taking several input objects amalgamating them into only one output object (or the other way around). More examples are given in section 3.

If a media server had to support such media transformations, two features should be guaranteed: First, applications must be strictly isolated from each other. That means, a transformation initiated by one application must never interfere with interests another application might have. This would eliminate the irreversibility problem mentioned earlier. Second, optimization of transformations must (almost) completely be handled by the server. While, at first glance, this appears being merely an option, it is, in fact, a must, because the application programmer would (and should) not be able to figure out the necessary filters for transforming the internal data format into the external data format (and back again).

Restating the latter from a different point of view, demanding format independence implies that a client's transformation specification can not be complete in principle, since it would not contain any instructions related to format conversion issues. Therefore, such an "incomplete" transformation specification henceforth is called a *transformation request*. Clearly, a transformation request must be semantically unambiguous to be complete from the client's point of view. But even this constraint leaves to the server some degrees of freedom beyond simply adding the required format conversion filters to the transformation request. Consider, e. g., a transformation request for an image object asking the image being both rotated and sharpened. Obviously, you would not really want to bother about the actual sequence of these two operations, since you are probably expecting that it makes no perceivable difference. However, you might truthfully suspect that there actually *is* a certain sequence to prefer due to conditions which only the server is able to detect. Consequently, stating a transformation request should not require specifying semantically irrelevant operation sequences. In principle, the server would then be able to choose any sequence, because each one is considered equally valid, but, naturally, the intention is to let the server determine the optimal sequence (in terms of both cost and quality). Thus, on receiving a transformation request the media server must autonomously compute a so-called *transformation script* specifying all the necessary filters and how to connect them.

There are probably very few reasons why a transformation request should prescribe where (i. e., on which machine) the filters eventually selected in the transformation script should be instantiated. Actually, we could not even find a demonstrative example, since it is our opinion that disabling or enabling the instantiation of certain filters on the client machine is better done by means of configuration. Hence, having computed the transformation script the server is now left to, again autonomously, decide where to instantiate the filters, finally obtaining a *transformation schedule*. To give an idea how optimization could work during this process, consider the following basic rules, which may be used to decide the instantiation problem for each filter independently:

- *Input-driven instantiation:* The filter is instantiated where its input is generated, which is often optimal if the filter produces substantially less data than it consumes. However, this rule might be ambiguous if there is more than one input stream.
- *Output-driven instantiation:* The filter is instantiated where its output should go to, which is often optimal if the filter produces substantially more data than it consumes. Again, this rule might be ambiguous if there is more than one output stream.
- *Operation-driven instantiation:* The filter is instantiated where it may optimally perform its operation, e. g. on a machine with special hardware equipment supporting this kind of operation. This is generally optimal if the filter's computational complexity or its resource demands are very high.

Note that these rules naturally include choosing the client machine as filter instantiation target, which then, of course, is required to provide some media server features.

Now, the final issue to clarify is, if the media server supports media transformations, what happens to the outcomes of such transformations. To better understand the problem consider the following observations:

1. Data independence implies that generally one can not simply overwrite the media data stored in the database (irreversibility problem).
2. If overwriting media data is not allowed, then transformations always lead to the same outcome when repeated.
3. Hence, if transformations are repeatable, then from the client's perspective there is no semantic difference between the following procedures:
   (a) Issue a transformation request, store its outcome in the database, and request the newly generated media object.
   (b) Issue the transformation request twice.

There *is*, however, a semantic difference between a media transformation and a media object explicitly stored in the database. This is due to the fact, that a transformation request is usually private to the application issuing it while media objects are usually not. Hence, if an application wishes a transformation's outcome being accessible by other applications, it should be able to instruct the server to create a new object from the transformation.

What follows from these considerations is that storing the media data is merely an optimization issue. To put it in other words, the existence of a unique identifier for a media object does not imply that this object is physically stored anywhere. The identifier might as well (transparently) point to a transformation script telling the media server how to create this object physically on the request of a client. Thus, providing media transformations and the opportunity of creating objects from such transformations completely disburdens the clients from even noticing the storage needs for media objects. (Note that the source object(s) of a media transformation can very well be located outside the database as long as the media server is able to access it.)

The additional degrees of freedom gained on the server's side can be exploited for various optimizations. Again, only some basic ideas are pointed out, since the actual realization is not essential for understanding the transformation independence concept.

- *Materializing transformations:* The server may decide to materialize any outcome of a media transformation—based on whatever optimization algorithm and without informing any client, including the one that originally invoked that transformation. Hence, the server may also retract this decision eventually, even if it is about to destroy the materialization of a media object.
- *Materializing intermediate objects:* Intermediate media objects come into (usually short) existence during the execution of a transformation schedule, however, they are never visible to the applications. The server might decide to materialize such an object if it detects that it is frequently created by transformations not resulting in the same final outcome.
- *Including the client:* The probably ultimate optimization strategy is materializing media objects on the client machines. This resembles traditional caching strategies. However, this would be no cache in its true sense, because it does not simply contain copies of media objects stored in the database—rather, most of the objects would be very individual items.

Finishing this section, the relation between the MADT concept and transformation independence has to be discussed. Notwithstanding the fact that the MADT concept has initiated the development of transformation independence it proves not being an adequate data model for realizing transformation independence. This is true for two reasons:

1. Since the MADT concept introduces traditional abstract data types for media objects, it does not provide means for modeling the process of dynamic refinement (at runtime) of media transformations.
2. The traditional method of specifying operations as functions or procedures makes it hard to specify filter operations taking several input streams and producing several output streams that can be combined together.

To prove the first statement, recall that the (M)ADT concept generally assumes the internal data format being rather deterministic. This is, however, not true with transformation independence, because it is not known at design time which physical media objects will be materialized by the DBMS. Hence, the media objects visible to client applications are really *virtual* media objects. Operations on such virtual media objects (VMO) have to be mapped to (semantically equivalent) operations on the internally materialized objects. That means, the operations on VMOs are virtual, too.

Therefore, the MADT concept needs to be enhanced to provide the means for describing how virtual operations are to be applied to virtual media objects (transformation request) and how this can be mapped to real operations on real media objects (transformation script). This new concept has been named *VirtualMedia* concept.

3 The VirtualMedia Concept

If transformation independence can be viewed as being the theoretic foundation of a universal media server, then VirtualMedia is the conceptual basis (though not considering all aspects characterizing the behavior of a universal media server, e. g. concurrency issues).

3.1 MADTs in Practice

To illustrate the major aspects of the VirtualMedia concept a running example is being used. This example is first introduced employing the ‘classic’ MADT concept in order to demonstrate the deficiencies stated above. Assume a video object being stored in the database and that this video shows a talk given by a famous scientist. A client of the database wants to hear this talk, but for whatever

reason she only wants to hear the voice without watching the video and, additionally, she would like to have a textual transcript of the talk displayed on her screen. To accomplish this task the client application could first issue the following command using object-oriented notation (and assuming that the variable `a_video` is already bound to the video object being the source of the presentation):

```
a_video.decompose( image_sequence, soundtrack )
```

where both parameters are output parameters. Note that, however, only the second parameter is really needed for further processing, thus indicating that this MADT design is not optimal under all circumstances. The MADT developer, however, must choose one specific design or—if he cannot make up his mind—introduce redundancy, e. g. by additionally providing a `create_from_video` function with the Audio MADT:

```
audio.create_from_video( a_video, soundtrack )
```

where `a_video` is an input parameter and `soundtrack` is an output parameter. Finally, the textual transcript must be created from the soundtrack and, once again, the MADT developer might ask himself to which media type this operation should belong: Audio or Text. Assume for this example he chooses the first possibility and, hence, the command would be:

```
soundtrack.transcript( a_text )
```

where `a_text` is the output parameter containing the transcript of the talk.

Although this example is far from being complete (e. g., specification of media quality has been omitted to keep things simple) it helps understanding that the MADT approach has some limitations regarding virtual operations: Since MADTs are completely designed before being used, the application developer, e. g., is not allowed to change the scope of an operation at runtime. If this were possible, she could find the following solution for our example, even if it had not been anticipated by the MADT designer:

```
a_video.transcript( a_text )
```

Assuming that the transcript operation had not been defined and implemented for the Video MADT, the system would then be forced to dynamically find an implementation (based on its knowledge on how to decompose videos and how to transcript soundtracks).

Our enhanced MADT concept *VirtualMedia* is designed to support such dynamic improvements of the MADT’s functionality. To achieve this, the operations must be decoupled from the data types such that any operation, in principle, is applicable to any media object regardless of its type. (The system, however, will not be able to guarantee that it always understands the client’s semantics. Thus, the client might sometimes be compelled to restate its request more precisely until getting the result wanted.)

Another weakness of the classic MADT concept is that the graph-like structure of media transformations—with (virtual) media objects being the edges and operations (filters) being the nodes of a directed acyclic graph (DAG)—is not reflected adequately. Consequently, this DAG structure has become the base of VirtualMedia.

3.2 The Filter Graph Media Processing Model

Modeling and realizing the processing (i. e. transformation) of media objects through filter graphs is a probably well-known principle (see, e. g., [CSV96] and [Din95]). However, to our knowledge it has never been applied to model the behavior of a media server and to build an abstract media transformation concept.

As already mentioned, a filter graph is basically a DAG. The start nodes of the graph are media sources (media objects stored in the database or anywhere else, maybe even live media sources) and the end nodes are media sinks (most often client applications or the database). The intermediate nodes are media filters, the basic operations forming a media transformation, while the edges of the graph represent media streams flowing from one filter (or media source) to another filter (or media sink).

To illustrate the concept of filter graphs, Figure 1 shows a sample filter graph derived from the running example. In all figures showing filter graphs the symbol denotes a media source, the symbol denotes a media sink, and the symbol denotes a filter. The filter graph in Figure 1 exactly represents the semantics of the example scenario without dynamic improvement (i. e. not applying the transcript filter directly to the video object), which means: It is a graph representation of the transformation request the application would send to the server. Since both media objects and operations are virtual to the application, we call this a *virtual filter graph*. Of course, a virtual filter graph is not instantiatable. Hence, Figure 2 shows another filter graph as the possible outcome of computing a transformation script from the virtual filter graph in Figure 1.

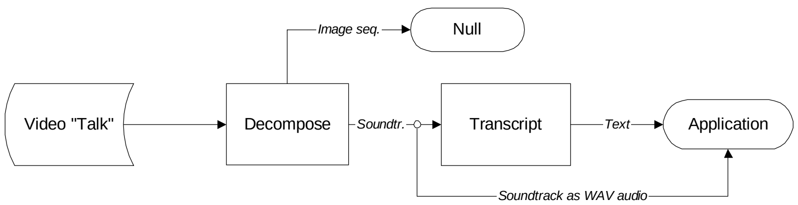

Figure 1: A Filter Graph for the Running Example.

In Figure 2 we assume that the server had to extend the transformation request graph by two additional filters (shaded). The first filter is needed because the real transcript filter available happens to being only able to process mono audio streams while decomposing the video is assumed to provide a stereo audio stream (there is actually no reason why the transcript filter itself should have the stereo to mono conversion capability, since this has nothing to do with speech transcription). The second filter converts the soundtrack being delivered by the decompose filter into a WAV-encoded audio stream, because WAV format has been requested by the application (cf. Figure 1). Note that equally named filters in Figure 1 and Figure 2 are generally assumed having very similar—if not identical—semantics (a more detailed discussion of semantics issues follows in section 3.3).

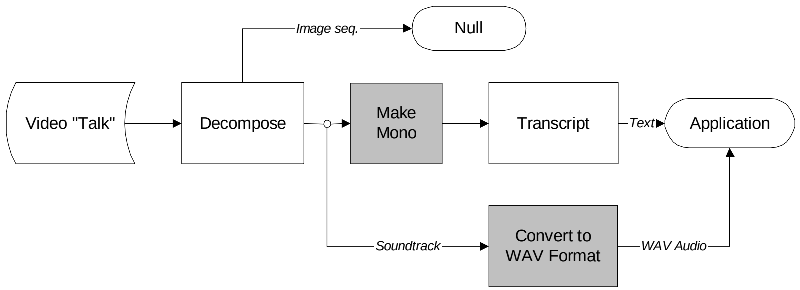

Figure 2: A Filter Graph Representing a Possible Transformation Script for the Example.

Considering the transition from the transformation request graph to the transformation script graph as demonstrated above, it can be concluded that this transition requires the server to carry out the following tasks:

1. Analyze the transformation request graph in order to detect all media type incompatibilities between subsequent nodes.
2. Resolve all these incompatibilities by extending the graph with additional filters, eventually yielding a correct transformation script graph.

Until now, only incompatibilities concerning certain attributes of a media stream (mono/stereo, encoding) have occurred, because the transformation request graph (Figure 1) has been directly derived from the first version of the example which apparently takes the internal ‘reality’ of the DBMS pretty well into consideration. Figure 2 has been intentionally designed to keep this semblance as yet—the proof that ‘reality’ might also be totally different is deferred to the next section (considering the “Talk” object being a virtual object, merely pretending to fulfill the semantics of a video object).

Tying up to the considerations of the previous section, with VirtualMedia we can easily build a virtual filter graph omitting the decompose filter from the transformation request even if the (real) transcript operation is documented as consuming an audio object and producing a text object. On choosing this option (shown in Figure 3) the client expresses its expectation that the system will find a suitable implementation for a (virtual) transcript operation consuming a video object.

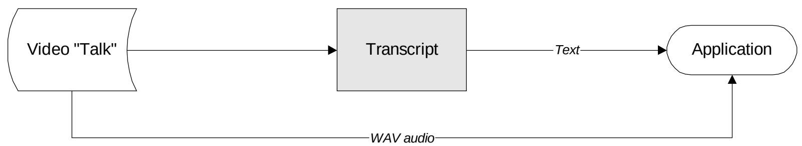

Figure 3: Simplified Transformation Request Graph for the Example.

Summarizing, the filter graph processing model is adequate for modeling both virtual operations on virtual media objects and real operations on real media objects. The transition from a virtual filter graph to a real filter graph is easily modeled by adding and removing nodes and edges. We can also interpret a virtual filter graph (transformation request) as a description of a set of *client virtual media objects* (CVMO). The actual number of CVMOs is determined by counting all the edges ending at an end node (media sink). Hence, the graph in Figure 3 describes two client virtual media objects: a text object and an audio object. The VirtualMedia concept therefore renames the transformation request as *virtual media descriptor* (VMD). A special *virtual media language* (VML) for writing and communicating VMDs is currently under development.

3.3 Semantics-based Optimization

Transformation independence opens powerful optimization options, some of which are sketched in section 2.3. In this section, we will reconsider those optimization issues related to the translation of transformation requests into transformation scripts. As stated in the previous section, with VirtualMedia this process is realized by adding or removing filters one after another to/from the virtual filter graph representing the transformation request until all virtual filters have been replaced by equivalent real filters. Hence, before considering possible optimization techniques, it is necessary to examine what rules must actually be followed to preserve the semantics of the transformation request throughout this process.

3.3.1 Semantics-preserving Filter Graph Manipulation

To begin with the bare truth: Perfect preserving of filter graph semantics is not achievable due to the fact that neither the client (application) has enough knowledge to be always able to specify its requests with sufficient accuracy nor the (DBMS) server is able to anticipate where an application's semantics deviates from that given by common domain-specific semantics. Providing the applications with the missing information would straightly destroy the abstraction introduced by transformation independence and, hence, is not considered further. On the other hand, while providing the server with any application-specific knowledge in advance is, of course, not possible, it is very well feasible to equip the server with domain-specific knowledge (i. e. common knowledge on media transformations).

Thus, we define that preserving semantics solely requires not violating such domain-specific knowledge. This knowledge is probably best described by introducing *equivalence classes* of filter graphs. We may specify these equivalence classes by rules stating, e. g., that certain filters are:

1. *Semantically inverse:* if two filters are determined to perform mutually inverse operations, then they can be safely eliminated from a filter graph in case they occur in a direct sequence.
2. *Semantically commutable:* two filters may be classified being commutable when occurring in a direct sequence.
3. *Semantically neutral:* such a filter can be safely added or removed without noteworthy semantic side effects (e. g., many format conversion filters are classifiable this way).
4. *Semantically assimilable:* two filters being equally named are semantically assimilable, if it is possible to unify their in- and outputs. This unification may be only accomplished by applying other rules (including this one) recursively. Example: inferring semantic assimilation of the virtual transcript filter (Figure 3) by the real transcript filter (Figure 2) is possible, if both the decompose-filter and the mono-filter are classified 'semantically neutral'.

While being the very basis for dynamic implementation of virtual operations, in a realistic implementation of VirtualMedia these rules certainly need quite a few refinements. Due to space limitations, however, we will not go into further detail here. Instead, we return to our running example showing how these rule-defined equivalence classes are exploited to manipulate and finally optimize filter graphs in order to obtain an instantiatable filter graph from a given virtual filter graph.

3.3.2 Eliminating Unnecessary Filters by Exploiting Semantic Knowledge

Recall that a media source object occurring in a transformation request is a virtual media object and, hence, does not necessarily exist as a physical object stored in the database. It might as well be represented internally by a transformation script (not known to the application), i. e. by a filter graph. Assuming that this is the case with our "Talk" video object, Figure 4 shows an imaginable transformation script for it.

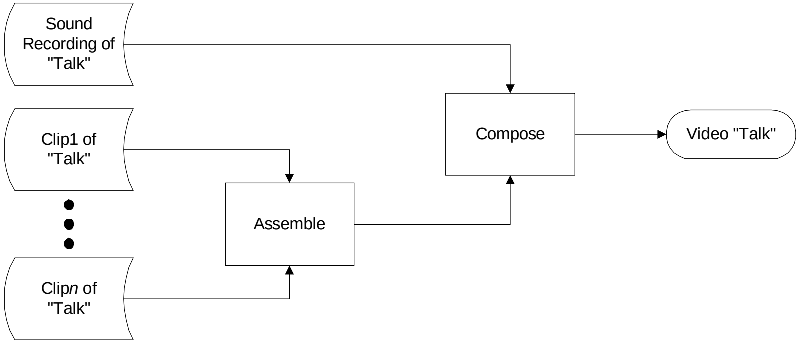

Figure 4: A Transformation Script Graph for Generating the Virtual "Talk" Video Object.

The transformation script graph in Figure 4 means that the "Talk" video is created by merging the sound recording of the talk with an appropriate video sequence which in turn is an assembly of several video clips (probably filmed with different cameras). On processing the original virtual filter graph (Figure 3) the server will eventually arrive at the graph of Figure 2 by applying rules *3* and *4*. Then, this graph must be expanded by substituting the "Talk" node with the graph in Figure 4. This combined filter graph is now optimizable by applying rule *1*, because a compose-filter is immedi-

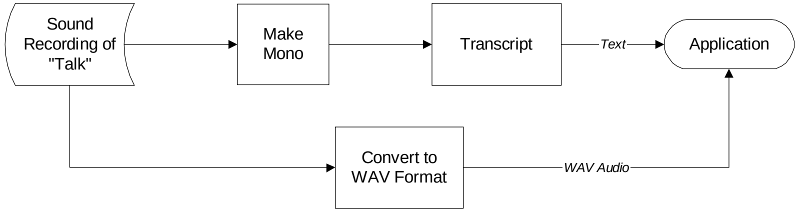

Figure 5: Improved Transformation Script Graph for the Example After Eliminating the Compose/Decompose Filter Pair.

ately followed by a decompose-filter, which obviously performs the inverse operation. Thus, the server can safely eliminate both filters from the transformation script finally getting the filter graph shown in Figure 5.

3.3.3 Exploiting Materialization of Virtual Media Objects

Materialization of the outcome of a media transformation (CVMO) or of an intermediate object (VMO) is another optimization option effectuated by transformation independence (cf. section 2.3). The decision (to be made by the server) which (C)VMO to materialize for later reuse is not specifically dependent on the processing model (i. e. the filter graph model). Rather, it probably depends on processing *costs* and maybe other conditions (e. g. statistics). What is, however, dependent on the processing model, is how materialized objects are internally represented in order to exploit them for optimization purposes.

To imagine how (C)VMO materialization works with filter graphs, assume the transcript filter in our example is a highly complex filter consuming a lot of processing resources (surely not an unrealistic assumption). This might, e. g., cause the user's quality of service (QoS) demands not being met (for too much delay) and, hence, provoke the server to keep the transcript internally, since that would increase the QoS dramatically if it were ever requested again by a client.

Making this optimization strategy effective, however, requires providing a means for recognizing when an existing materialized (C)VMO is applicable to optimize a given filter graph. For this purpose, the materialized object must be (internally) represented by a filter graph documenting its relation to (externally visible) database objects. Due to our equivalence class definition on filter graphs, however, there are many choices available.

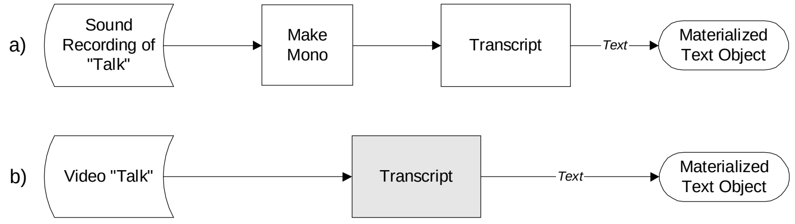

Figure 6: Two Options for Representing a Materialized Object. a) As a Real Filter Graph. b) As a (Maximal Simplified) Virtual Filter Graph.

Figure 6 presents two promising candidates (again referring to our example):

a) *The real filter graph* that was applied to create the materialized object. Obviously, this is the only representation making the exact physical 'history of origins' explicit. We may therefore consider it as a pragmatically defined normal form. 'Pragmatically', because its derivation generally is not repeatable (due to dynamic factors in cost functions applied during optimization). One major drawback of using this representation is that we must expect large distances (measured as minimal required number of graph transformations) to typical transformation requests (containing mostly virtual filters). It would, hence, be relatively expensive to match a materialized object with a given transformation request.

b) *The (maximal simplified) virtual filter graph* representing the 'pure semantics'. The 'maximal simplified' is put in brackets since, in general, it is not decidable which is actually the most simplified filter graph. One could, however, imagine heuristics providing an acceptable approximation. Starting with such a representation, matching a materialized object with a given transformation request could often be less expensive compared to case a). There is, however, a price to pay: we might inadvertently ignore system evolution. Referring to our example, after having created the materialized text object the original transcript filter might have been replaced by an improved one. Hence, since the filter graph in Figure 6 b) can be easily matched with the transformation request in Figure 3 without regarding any real transcript filter, additional effort would be required to assure that the materialized object is not obsolete.

Currently, it is hard to say which of the both approaches is to prefer. During ongoing research, it may even turn out that it is best applying both in a hybrid manner. With the following Figure 7, showing an optimized transformation script graph exploiting a materialized transcript being available, we conclude section 3 and our running example.

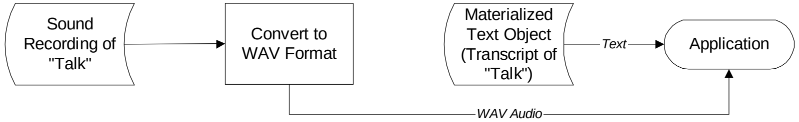

Figure 7: Final Transformation Script Graph for the Example Exploiting Materialization.

4 Related Work

The transition from our earlier MADT approach to the VirtualMedia approach also means switching from a local to more global approach regarding MADT design and solving the optimization and other problems. To our knowledge, such a strategy has not yet been pushed for realizing media data types. Therefore, we compare our approach to some others that are following a local design and/or optimization strategy, starting with the most closely related one: the E-ADT approach [Ses98].

The probably well-known concept of *Enhanced Abstract Data Types* (E-ADT) has already been realized in the ORDBMS prototype *Predator*. Its superiority over traditional ADTs is primarily constituted by an E-ADT's ability to optimize complex operations. Complex operations are combinations of elementary E-ADT-operations, e. g. *Clip(Sharpen(G.Photo), 0, 0, 100, 200)* (taken from [Ses98]). Thus, a complex operation is similar to a transformation request or VMD.

The E-ADT interprets a complex operation as an (algebraic) description of the transformation process to be applied to the (media) object. For optimizing the complex operation there are four classes of optimization rules available, each of which has its counterpart in VirtualMedia:

- *Algorithmic optimization:* There may be different algorithms realizing the same operations. Their performance might, e. g., depend on certain input characteristics like size or format of the media object. Determining the optimal algorithm corresponds to finding the optimal real filter implementing a given virtual filter in VirtualMedia.
- *Transformational optimization:* This means changing the order of operations according to (semantics-preserving) equivalence transformations, e. g. permuting the Clip- and the Sharpen-

operation of the example above. Obviously, this is rather similar to the semantics-preserving filter graph transformation in VirtualMedia.

- *Constraints:* The client's requirements regarding the quality (e. g. resolution) or other physical properties (e. g. external format) of the final outcome of the complex operation are exploited for optimization. According to [Ses98], such constraints apparently occur as side-effects of certain operations, e. g. ChangeResolution(). In VirtualMedia, these constraints generally are specified explicitly as parameters of the CVMOs in a VMD.
- *Pipelining:* Consecutive operations may be connected by a "data-pipeline" instead of executing all operations strictly one after the other with entirely creating and storing all intermediate results. Obviously, this principle is virtually "built-in" into VirtualMedia's filter graph processing model.

Generally, the E-ADT approach is more generic than VirtualMedia in a sense that it is not particularly targeted at stream-like (media) data. On that background, the decision to only provide for E-ADT-local optimization becomes comprehensible, since it would be rather difficult to specify a global optimizer for ADTs with only few common semantics that performs better than specialized ADT-local optimizers. Media data types, however, belong to those data types that mostly benefit from optimization (which probably is the reason, why they are most often exerted as examples in texts on ADTs as in [Ses98]). Further, different media data types not only have significant affinity regarding design and optimization principles—it is virtually impossible to find a set of abstract data types that properly and consistently describes the semantics of media data without introducing a high-grade overlap of the ADTs' implementations (algorithms, internal formats, etc.). Consider, e. g., the logic concepts *image*, *image-sequence*, and *video*. With the E-ADT approach (and also our own, now abandoned MADT approach) we would be required to think about creating an ADT for each of this concepts, since they all have different semantics. And, hence, we would have to create three separate optimizers not knowing from each other and, therefore, not being able to cooperate. But they all probably have to deal with partially the same physical data types (note that, e. g,, an AVI-file is a suitable physical data format for all three ADTs) and algorithms (e. g. image filtering). And cooperative optimization actually is mandatory for the close relationships between media-ADTs: A video is composed of image-sequences (and probably also audio data) and, in turn, an image-sequence is composed of images. Thus, (logical) composition and decomposition are quite natural (and frequent) operations on these ADTs (without necessarily changing the internal physical representation of the media objects). Since each (de-)composition potentially puts another ADT into play, ADT-local optimization would not be able to look across a (de-)composition-border, which is particularly awkward if different ADTs are able to share physical data formats. We must, therefore, conclude that complex operations comprising (de-)composition of media objects can not be sufficiently optimized by E-ADTs as with VirtualMedia (unless the E-ADTs are specified in a highly redundant and/or logically inconsistent fashion, e. g., only one "omnipotent" E-ADT for all media data).

There are some other major differences between E-ADTs and VirtualMedia. First, E-ADTs do not provide *virtual* objects that are manipulable without accidentally loosing data. Thus, E-ADTs are subject to the irreversibility problem. Second, the E-ADT approach does not consider materialization (automatically controlled by the server) as an optimization strategy. And, third, E-ADTs are designed to be tightly and fully integrated with (more or less) traditional database systems. VirtualMedia, on the other hand, is designed with only partial integration with extendible database systems like, e. g., ORDBMS in mind, while considerably relying on (distributed) media servers that are not likely to share many resources with an (OR-)DBMS.

The approaches and concepts considered in the following paragraphs only have minor commonness with VirtualMedia. Hence, they are examined less detailed than the E-ADT approach.

Within the AMOS project at GMD IPSI a concept called *presentation independence* has been developed [RKN96]. The focus here is on separating the content and logical structure of a presentation from the management of the quality of service (QoS). That is, the content and logical structure of a presentation can be defined independently from the physical representation of the media data. The actual QoS of a presentation depends on the resources available at the server and the client. Because the resources are not reserved, a mechanism called *Adaptive QoS Management* [Thi98] is applied to guarantee the smoothness of the presentation (accepting QoS degradation). Multimedia presentations can be easily defined on top of VirtualMedia (using VMDs). The QoS adaptation could be realized through special (semantically neutral) adaptation filters. In contrast to the solution presented in [Thi98], this would allow exploiting any kind and combination of scaling (spatial and temporal) for smoothly adjusting the data flow. The communication overhead for feedback chains, however, would be considerably higher. Also, the optimization algorithm proposed in [Thi98] (based on linear programming) possibly does not scale up with the increase of parameters and, hence, would have to be replaced by a computationally less complex heuristic-based algorithm.

The Hypermedia DBMS described in [PS96] provides format independence in a straightforward manner. The notion "format independence", however, is not used in [PS96]. Instead, another pair of abstractions—*media independence* and *storage independence*—which together have a similar meaning is introduced. At the time a media object is inserted the DBMS stores it using its external format, now called its *primary format*. If a client wants to retrieve the object using an external format different from the primary format, then the DBMS creates this format and stores it internally as a *secondary format* of the object. This solution is widely trouble-free, because operations manipulating the media objects are not considered. For the same reason, however, it is not a quarter as versatile and flexible as VirtualMedia or E-ADTs.

Commercial ORDBMSs (available, e. g., from Informix, IBM, and Oracle) are extensible by defining and implementing *User-defined Types* (UDT). This mechanism is also extensively used to enhance those systems with media data types (for some examples see, e. g., [Inf97a]). There are, however, very few such media types trying to satisfy any of the abstractions considered in section 2. One remarkable exception of this rule can be found in [HSH+98], which describes a *continuous media datablade* providing device independence, location transparence, and presentation independence (as introduced above). The functionality of this datablade is limited to media presentation—general media transformations (and optimization) are not considered.

In the KANGAROO project [MR97] a media server supporting format independence and media transformations using filters is being developed. This media server is able to *execute* VirtualMedia's filter graphs. However, it does not contain a VMD-based application programming interface (API) and the corresponding optimizer (it has a lower level API instead). Hence, it may become a *basic* component of a future universal media server.

5 Conclusions

In this paper, abstractions and concepts making the vision of a universal media server a bit more concrete are presented. Beside common abstractions like device and data independence we consider a newly developed abstraction called transformation independence. In principle, this abstraction requires a media server to solve the following problems:

- *Overcome irreversibility* of most of the operations that are applicable to media objects. First of all, this means isolating concurrent applications with respect to updates of media objects—at the fee of an increased resource demand, e. g. storage space for older versions of a media object.
- *Optimize media transformations* globally, i. e. (1) by considering the transformation request as a whole regardless of the type and number of media objects involved, (2) by exploiting general domain knowledge on multimedia processing (rules, cost functions), and (3) by collecting and evaluating statistical data. As a prerequisite, this requires a transformation request interface allowing to request media transformations in a descriptive manner. Transformation requests should (ideally) contain only statements of semantic relevance to the application.
- *Support format independence* by seamlessly integrating format-related operations into media transformations, which might either be caused by internal formats not being compatible with a requested transformation or by requested external formats not matching currently available internal formats.

As a possible approach to realize transformation independence the VirtualMedia concept is introduced. VirtualMedia solves the irreversibility problem by establishing a layer of virtual media objects which applications may unrestrictedly manipulate. We adopt the filter graph model to represent virtual media objects as transformation graphs. Semantics-preserving rules for transforming filter graphs allow for mapping virtual objects and operations to real (physically existent) objects and operations while applying different optimization strategies like materialization or cost-based evaluation of semantically equivalent filter graphs.

Rule-based transformation and optimization of operator graphs have been studied for more than a decade in the context of extendible database query optimizers [RH87, CZ96], optimizer generators [SS90, GM93], and query optimizers for object-oriented databases [VD91]. Structurally, filter graphs are nearly identical to operator graphs. Semantically, of course, they are different—there exist, however, several analogies:

- Logic operators (e. g., *join*) correspond to virtual filters, while the implementations of logic operators (physical operators, e. g., *nested-loop-join*) correspond to real filters.
- There are operators having no corresponding operator in the logic algebra (called *enforcers* in [GM93]). These are used to ensure certain physical properties of the data (e. g. sorting). In VirtualMedia such operators occur as format filters for conversion, scaling, or (de-)compression.
- Virtual methods, for which the appropriate implementation can only be determined at run time, are similar to virtual filters having no predefined implementation for the data type they are applied to, in a sense that the implementation must be deduced from the data type at run time (‘virtuality’ of filters, however, does not come from explicit type inheritance, but from an implicitly assumed type affinity). Algebraic support for types with virtual methods is described, e. g., in [VD91].

Consequently, the structural and (from an abstract point of view) also semantic similarity of operator and filter graphs gives reason to follow an algebraic approach for realizing filter graph transformation and optimization. Such an approach would mostly benefit from the vast amount of knowledge and experience already gained through the development of query optimizers (regarding, e. g., algebras, algorithms, verification methods, languages, and tools).

Another tempting approach is realizing the optimizer as a *fuzzy system* (based on fuzzy sets, fuzzy rules, and fuzzy reasoning) [TKS92, BC95]. One big advantage of this approach is its outstanding qualification for expressing and applying heuristic optimization rules. There exists, however, very little experience in using fuzzy reasoning for rule-based query optimization. Hence, it may not be possible to assess the superiority of this approach prior to scrutinizing the more traditional algebraic approach.

Besides the optimization problem, ongoing research also focuses on the refinement of several other aspects of the VirtualMedia concept and on the exploration of several open questions. Of particular interest are the following aspects, to name a few:

- Enhancing the filter graph model with, e. g., hierarchical structures or a template concept.
- Formal specification and experimental evaluation of the graph transformation rules.
- Development of a reference architecture for a VirtualMedia server based on ORDBMS technology and KANGAROO.
- Integrating adaptive QoS (feedback-controlled) and interaction (including interactive filters).
- Several API issues, e. g. definition of a comprehensible, easy-to-use VirtualMedia language (currently planned to be based on XML) and consideration of fuzzy rules for semantics-preservation control[2].

This list is by far not complete, but it should be sufficient to indicate that this work is still in an early phase and, hence, the realization of the ideas presented here is a moving target, yet.

Acknowledgements

The author is grateful to Henrike Berthold, Theo Härder, Andreas Henrich, Wolfgang Mahnke, Klaus Meyer-Wegener, Norbert Ritter, Günter Robbert, and Hans-Peter Steiert for lively discussions on the ideas presented in this paper.

References

[BC95] Bosc, P., Kacprzyk, J. (eds.): Fuzziness in Database Management Systems. Berlin: Physica-Verlag, c/o Springer-Verlag, 1995.

[CSV96] Candan, K. S., Subrahmanian, V. S., Venkat Rangan, P.: Towards a Theory of Collaborative Multimedia. In: Proc. IEEE International Conference on Multimedia Computing and Systems (Hiroshima, Japan, June 96), 1996.

[CZ96] Cherniack, M., Zdonik, S. B.: Rule Languages and Internal Algebras for Rule-Based Optimizers. In: Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data (Montreal, Canada, June 4–6), SIGMOD Record Vol. 25, Issue 2, June 1996, pp. 401–412.

[Din95] Dingeldein, D.: Multimedia interactions and how they can be realized. In: Proc. Int. Conf. on Multimedia Computing and Networking, 1995.

[2] This could give the optimizer an idea how exactly the semantics of a transformation request should be fulfilled. For example, a client demanding 16bit-audio might also be satisfied with 12bit or 24bit. Knowing this, the optimizer could possibly save an audio filter. Of course, the client must be able to control the fuzziness of the request, e. g., by specifying parameter ranges, additional confidence parameters, or symbolic fuzziness indicators like “lazy” or “high fidelity”.

[GM93] Graefe, G., McKenna, W. J.: The Volcano Optimizer Generator: Extensibility and Efficient Search. In: Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 209–218.

[HSH+98] Hollfelder, S., Schmidt, F., Hemmje, M., Aberer, K., Steinmetz, A.: Transparent Integration of Continuous Media Support into a Multimedia DBMS. In: Proc. Int. Workshop on Issues and Applications of Database Technology (Berlin, Germany, July 6–9), 1998.

[Inf97a] Informix Digital Media Solutions: The Emerging Industry Standard for Information Management. Informix White Paper, Informix Software, Inc., 1997.

[Inf97b] Informix Video Foundation DataBlade Module. User's Guide Version 1.1. Informix Press, June 1997.

[KMM94] Käckenhoff, R., Merten, D., Meyer-Wegener, K.: MOSS as Multimedia Object Server – Extended Summary. In: Steinmetz, R., (ed.): Multimedia: Advanced Teleservices and High Speed Communication Architectures, Proc. 2nd Int. Workshop IWACA '94, (Heidelberg, Sept. 26–28), Lecture Notes in Computer Science vol. 868, Berlin: Springer-Verlag, 1994, pp. 413–425.

[MR97] Marder, U., Robbert, G.: The KANGAROO Project. In: Proc. 3rd Int. Workshop on Multimedia Information Systems (Como, Italy, Sept. 25–27), 1997, pp. 154–158.

[NMB96] Narang, I., Mohan, C., Brannon, K.: Coordinated Backup and Recovery between DBMS and File Systems. IBM Research Report, IBM Almaden Research Center, Oct. 1996.

[PS96] Prückler, T., Schrefl, M.: An Architecture of a Hypermedia DBMS Supporting Physical Data Independence. In: Proc. 9th ERCIM Database Research Group Workshop on Multimedia Database Systems (Darmstadt, Germany, March 18–19), 1996.

[RH87] Rosenthal, A., Helman, P.: Understanding and Extending Transformation-Based Optimizers. In: Data Engineering Vol. 9(4), 1987, pp. 220–227.

[RKN96] Rakow, T., Klas, W., Neuhold, E.: Abstractions for Multimedia Database Systems. In: Proc. 2nd Int. Workshop on Multimedia Information Systems (West Point, New York, USA, Sept. 26–28), 1996.

[Ses98] Seshadri, P.: Enhanced abstract data types in object-relational databases. In: The VLDB Journal Vol. 7 No. 3, Berlin, Heidelberg: Springer-Verlag, Aug. 1998, pp. 130–140.

[SS90] Sciore, E., Sieg, Jr, J.: A Modular Query Optimizer Generator. In: Proc. 6th Int. Conf. on Data Engineering, 1990, pp. 146–153.

[Thi98] Thimm, H.: Optimal Quality of Service under Dynamic Resource Constraints in Distributed Multimedia Database Systems. GMD Research Series, No. 10, Sankt Augustin: GMD – Forschungszentrum Informationstechnik GmbH, 1998.

[TKS92] Terano, T., Kiyoji, A., Sugeno, M.: Fuzzy Systems Theory and Its Applications. London Academic Press, 1992.

[VD91] Vandenberg, S. L., DeWitt, D. J.: Algebraic Support for Complex Objects with Arrays, Identity, and Inheritance. In: Proc. of the 1991 ACM SIGMOD Int. Conf. on Management of Data (Denver, Colorado, May 29–31), SIGMOD Record Vol. 20, Issue 2, June 1991, pp. 158–167.