Transformation Independence for Multimedia Systems

Ulrich Marder
University of Kaiserslautern
Dept. of Computer Science
Database and Information Systems Group
P. O. Box 3049
D-67653 Kaiserslautern
Germany

marder@informatik.uni-kl.de

ABSTRACT

We present our efforts on realizing transformation independence in open, extensible, and highly distributed multimedia (database) systems. Transformation independence is an abstraction concept which allows clients to request operations on media objects without knowledge of the actual physical media representation. Systems supporting transformation independence are expected to dynamically determine an optimal realization of such a request, depending on the actually available materialization, processing and communication resources, and other criteria (e. g., a cost limit).

Besides a detailed discussion of general media data abstractions, we focus on the abstract data and processing model called VirtualMedia[1]*, which provides a transformation independence framework for multimedia processing. In particular, we describe how transformation requests are represented and processed, exploiting semantic equivalence relations on filter graphs and redundant materialization, finally yielding instantiable plans for materializing the requested media object(s) at the client.*

Keywords

Media data abstractions, Media processing framework, Multimedia Database Systems

[1]This work is supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Sonderforschungsbereich (SFB) 501 “Development of Large Systems with Generic Methods”.

Contents

1 Introduction

In the age of the world-wide web global media data—often called media assets—, stored in multimedia database management systems (MM-DBMS), increasingly become accessible to virtually everyone. Hence, lots of heterogeneous clients with different, not safely predictable capabilities of storing, processing, and presenting media data are willing to access these MM-DBMSs. In general, there exist two categories of client applications. The first only want to retrieve media objects for presentation or possibly printing. The second (additionally) create media objects or modify them through editing and/or composition. Assuming a sort of unbalance between these application types sounds reasonable: Usually there will be many applications of the presentation type and much less of the other type.

Traditional database management systems are not sufficiently prepared to support such world-wide applications. In fact, the management of media data like images or video 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 unknown).

This resembles very much the situation in which media objects are simply stored in files, as usually done in most today's multimedia information systems (MMIS). Generally, two different categories of media servers are used in such MMIS. First, generic file servers (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.

Until recently, application neutrality has not to been considered an important issue with MMIS. The ubiquitous world-wide web, however, has initiated the development of large open distributed MMISs, thus making application neutrality increasingly become 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. Now, the crux is that we really want equally *well* service instead of equally bad service. This means that the (ideal) media server should be capable of satisfying almost every requirement—including performance—a potential multimedia client might pose. Obviously, such a goal is hard to achieve, and 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.

Recalling the assumed dominance of presentation applications, one could be tempted to optimize the media server(s) for presentation only. This could, however, easily result in disregarding the stronger quality and performance demands of many media editing applications. Consequently, media servers being an (essential) part of MM-DBMSs should uncompromisingly provide physical data independence. Today's media servers—especially continuous media servers—, however, do not provide physical data independence at all. One—if not the main—reason for this, in fact, is performance, which is due to several problems we are facing on attempting to provide physical data independence with a media server (or MM-DBMS, respectively): Such systems tend to require frequent format conversions inevitably resulting in bad performance. They may inadvertently lose data due to irreversible updates. Moreover, hiding the internal data representation from the client also means that all the strongly necessary optimization is to be accomplished by the server, which is both more difficult and more promising than leaving optimization to the applications.

Thus, since physical data independence *without optimization* costs a lot of performance, considering this optimization problem (and solving it) is crucial for future MM-DBMSs. Our approach is based on transformation independence, which is a stronger abstraction than physical data independence. It roughly means separating the semantics of media data from the technical aspects like media data formats, materialization, and optimization. In the following two sections, our ideas towards realizing transformation independence are presented. Namely, section 2 recalls basic data abstractions and discusses transformation independence in detail while section 3 introduces the VirtualMedia concept we are proposing as a general framework for realizing transformation independence. Section 4 and 5 present related work and conclusions, respectively.

2 Media Data Abstractions

Basically, our approach originates from an attempt to realize so-called *Media-specific Abstract Data Types* (MADT) [KMM94, 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.

Thus, the question arises whether it is generally feasible 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 that could be useful in finding a solution are worked out in detail.

2.1 Location Transparency and Device Independence

Location transparency 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 DBMSs (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., IDS/UDO[2] with its Video Foundation DataBlade [Inf97b]). Obviously, this approach takes the burden of being really concerned with media presentation or special device

[2] Informix Dynamic Server with Universal Data Option

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.

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 will not 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 rely 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 metadata (i. e., data describing the raw data) together. The relation between these two is kind of unbalanced, since the metadata is always needed to interpret the raw data, whereas the metadata alone by all means may be reasonably interpreted. For instance, the metadata might absolutely allow answering many queries concerning the content of a media object without (the DBMS) ever examining the raw data. Such reasoning about media data requires *logical data independence*. For instance, talking about a specific scene in a video assumes a method for addressing that scene which is independent from the video’s data format. Logical data independence, however, does not shield the raw data’s internal format, e. g., when an application wants to access the above mentioned scene for presentation. Hence, applications may still have to deal with a raw data format which they do not prefer or—much worse—which they do not even know.

Consequently, full data independence has to consider the raw media data as well. This aspect of data independence is usually called *physical data independence* or *format independence*. From the beginning, 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 (cf. Fig. 1).

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[3] with the internal format. Moreover, information loss occurs as a side effect of many operations on the media data, e. g., clipping, down-

[3]DCT (discrete cosine transformation) is a very common technique in image and video compression, because it allows to remove information being scarcely noticed by the human eye.

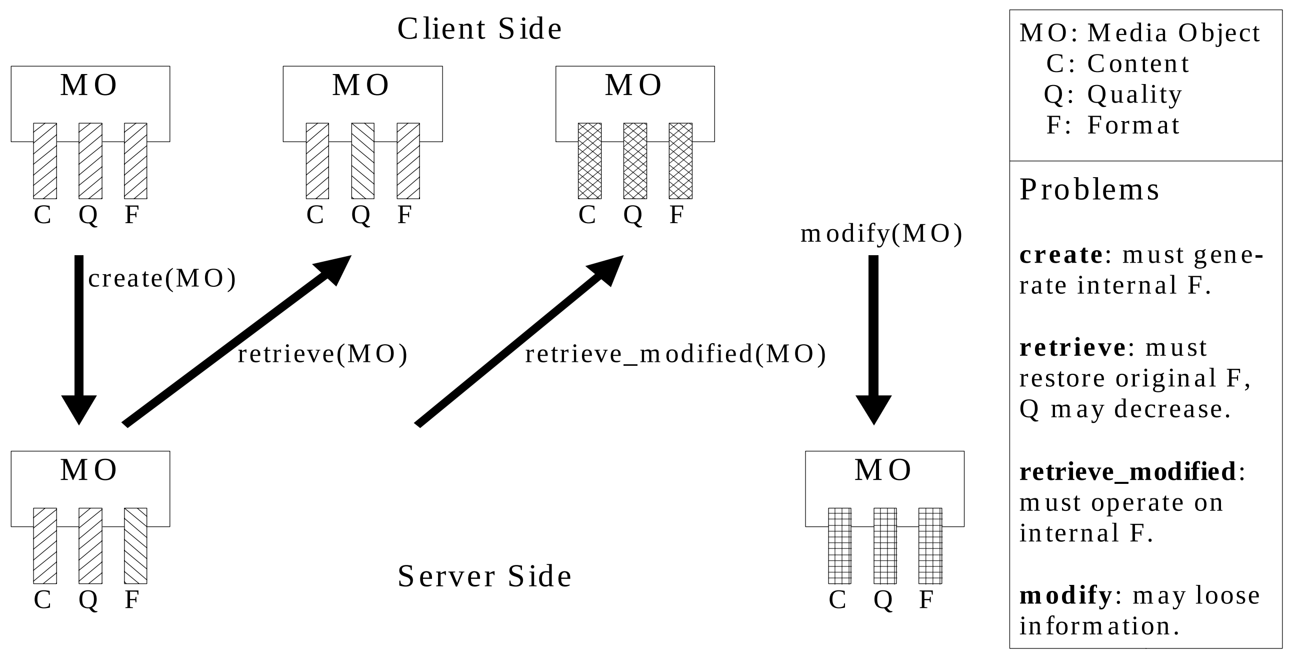

Client Side
MO
C Q F
create(MO)
retrieve(MO)
retrieve_modified(MO)
modify(MO)
Server Side
MO: Media Object
C: Content
Q: Quality
F: Format
Problems
create: must generate internal F.
retrieve: must restore original F, Q may decrease.
retrieve_modified: must operate on internal F.
modify: may loose information.

FIGURE 1: Problems Caused by Format Independence During an MO's Life-cycle

scaling, 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 small 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 a 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 physical data independence as illustrated above would be a first improvement. However, it would probably not perform very well, because using a hidden internal format effectively prevents any client-side application-specific optimization. First, as Fig. 1 indicates, in the case of the retrieve-operation the server quite frequently performs format conversions whereby an MO is converted back to its original external format. Obviously, this should never happen[4], but there is nothing the application (programmer) could do to inhibit such behavior. To illustrate the second problem recall the example above, in which

[4] Not only for performance reasons: Such "back and forth" conversion might also adulterate the MO or reduce its quality due to computational inaccuracy.

creating the external image format involves scaling and color transformations. Semantically, the sequence in which these operations are applied to the MO does not matter at all. Depending on the actual internal data format, however, both performance and MO quality may vary substantially if different operation sequences are chosen. Again, the application has no chance figuring out the optimum, because the internal format is out of reach.

Thus, if we still insist on realizing an MM-DBMS (or media server) providing physical data independence, the following consequences must be drawn from the previous considerations:

- First, applications are (of course) allowed to modify MOs. Since (normally) the MO is a shared object, the media server then must guarantee that other applications are not automatically affected by any modification. Therefore, some kind of version control is required. In contrast to more traditional versioning models, however, the access path to the older (original) version is always to be prioritized. This assures that an application maintaining an external reference to an MO can safely ignore any MO-modifications made by other applications.
- Considering optimization, the most common case should be optimized in the first place. That obviously means that the most popular external formats also have to be used as internal formats in order to prevent "back and forth" format conversions whenever possible. What *is* popular, however, may vary during the life-cycle of an MO. This variation is also likely to introduce redundancy - that is materialization in different formats at the same time.
- Last but not least, optimizing media modifications *at and by the server* is required. As a precondition, all operations must be expressible without any reference to or even assumptions on the internal format. This would make dynamic optimization based on the actual internal format(s) possible. Presumably, the mapping of such abstract operations to really executable operations is subject to semantic fuzziness. Hence, a formally defined semantic model should be introduced to limit fuzziness.

Putting it all together yields a new abstraction, which we call *transformation independence.* This abstraction embraces physical data independence while clearly indicating a shift towards a single solution to both the optimization problem and the format independence problem.

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.

Media Transformations 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 real 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 (called a *media manipulation*), while another transformation might create a textual transcript from a speech recording, thus changing the media type from audio to text (called a *media translation*). One can also imagine transformations taking several input objects amalgamating them into only one output object (called a *media composition*), or vice versa (called a *media decomposition*).

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 inadvertently affect another application. 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).

Transformation Requests 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 prescript* specifying all the necessary filters and how to connect them.

Filter Instantiation A transformation request only cares about semantics, thus ignoring, where (i. e., on which machine) the filters eventually selected in the transformation script should be instantiated. Hence, having computed the transformation prescript 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. If we consider such a filter being implemented as a web service, we could even imagine a mechanism similar to UDDI[5] to discover and apply the filter at runtime.

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.

Materialization Transformation independence keeps the 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 must explicitly instruct the server to create a new object from the transformation.

[5]Universal Description, Discovery, and Integration, see http://www.uddi.org for more information.

This distinction, however, must not affect materialization, which means, 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 prescript 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 at the server 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:* Another 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. Furthermore, the client may allow using these local materializations in fulfilling other clients' requests, thus realizing true peer-to-peer computing.

Transformation Independence and the MADT Concept 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 prescript). This new concept has been named *VirtualMedia* concept.

3 The VirtualMedia Concept

The VirtualMedia concept is particularly targeted at realizing transformation independence in a distributed, heterogeneous (e. g., Web-based) MMIS. Generally, VirtualMedia addresses API, data model(s), architecture, DBMS-integration, optimization, protocol, visualization, and interoperability issues. However, using a small example, the following introduction mainly focuses on API, data model, and optimization concepts.

3.1 Transformation Requests in VirtualMedia

To illustrate the major aspects of the VirtualMedia concept a running example is being used. As mentioned before, accessing a virtual media object requires creating an appropriate transformation request and sending it to a VirtualMedia-enabled server. Only semantics, logical structure and general media type information on VMOs may be exposed to the clients. Hence, VirtualMedia uses a kind of media description language suited for specifying media transformations at this abstraction level.

Fig. 2 shows the VirtualMedia transformation request for our example, thus illustrating some of the major features of the *VirtualMedia Markup Language* (VMML). In VMML a transformation request is called a *VirtualMedia Descriptor* (VMD). Starting with the semantics of this sample request, assume a video object is stored in the database and shows a talk given by a famous scientist. A client of the database wants to hear this

```
<?xml version="1.0" encoding="UTF-8"?>
<?DOCTYPE VMD SYSTEM "vmd.dtd"?>
<VMDESC>

   <SOURCE>
      <MOID ALIAS="BC_Video" EXT_REF="CNN_DB/Videos/4711"/>
   </SOURCE>

   <VIRTUAL NAME="TranscriptedSpeech">
      <SIGNATURE>
         <PROPERTY NAME="MAINTYPE" CLASS="typespec">TEXT</PROPERTY>
         <PROPERTY NAME="SUBTYPE" CLASS="typespec">PLAIN</PROPERTY>
         <PROPERTY NAME="ENCODING" CLASS="typespec">UTF-8</PROPERTY>
      </SIGNATURE>
      <TRANSFORMATION NAME="Transcription">
         <OPERATION SEMANTICS="Transcript">
            <INPUT ALIAS="i1" REF="BC_Video"/>
            <PARAM NAME="Language" VALUE="EN"/>
         </OPERATION>
      </TRANSFORMATION>
   </VIRTUAL>

   <VIRTUAL NAME="Speech">
      <SIGNATURE>
         <PROPERTY NAME="MAINTYPE" CLASS="typespec">AUDIO</PROPERTY>
         <PROPERTY NAME="SUBTYPE" CLASS="typespec">WAVEFORM</PROPERTY>
         <PROPERTY NAME="ENCODING" CLASS="typespec">WAV</PROPERTY>
         <PROPERTY NAME="SAMPLING_FREQUENCY" CLASS="quality">44100</PROPERTY>
         <PROPERTY NAME="SAMPLE_DEPTH" CLASS="quality">16</PROPERTY>
      </SIGNATURE>
      <TRANSFORMATION>
         <OPERATION SEMANTICS="nop">
            <INPUT ALIAS="i2" REF="BC_Video"/>
         </OPERATION>
      </TRANSFORMATION>
   </VIRTUAL>

</VMDESC>
```

FIGURE 2: A Sample Transformation Request (VirtualMedia Descriptor)

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 request first references the video object as a source MO. Since this MO is well known to the server, specifying any type information is optional and, thus, omitted. Next, the request specifies the two target objects as VMOs. Since these both must be materialized at the client, exact type information (signature) is required for the external format. Optionally, a signature may include quality properties as demonstrated with the second VMO and content properties (not included in the example).

The transformation section is mandatory for each VMO, because even if there is no "real" transformation operation to specify (as with the second VMO), it must be present defining at least one source object from which the VMO is to be materialized. Note that one could also think of replacing the "NOP" operation by some other operation that separates the audio part from the video part of the source MO. Thus, by omitting this operation we rely on the server knowing an appropriate default operation for extracting the audio. This is a reasonable assumption if the video has only one audio track. Otherwise (if, e. g., multiple languages are provided), it would indeed be necessary to refine the second transformation section.

Any VMO and any intermediate object (named transformation or named operation output) may be chosen as input to operations. The only restriction is of course, that no media object may be input to itself, neither directly nor indirectly. To build such source-target relationships, multiple transformation sections (within one VMO) and multiple operation sections (within one transformation section) are allowed. With respect to the introductory character of this article the example does not further demonstrate these advanced features.

Effectively, a (successfully verified) VMD describes a directed acyclic graph (DAG). Source objects become start nodes, operations become intermediate nodes, and only VMOs become end nodes of this graph. The edges are derived from the source-target relationships. Thus, this graph structure is a suitable internal model for describing any media transformations requested through VMDs[6].

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 abstract media data types.

3.2.1 Filter Graphs

A filter graph also is a DAG (Fig. 3). The start nodes of the graph are media producers p_i (media objects stored in the database or anywhere else, maybe even live media sources) and the end nodes are media consumers c_i (most often client applications or the database). The intermediate nodes are media filters f_i, the basic operations forming a media transformation, while the edges of the graph represent media streams flowing from one filter (or media producer) to another filter (or media consumer).

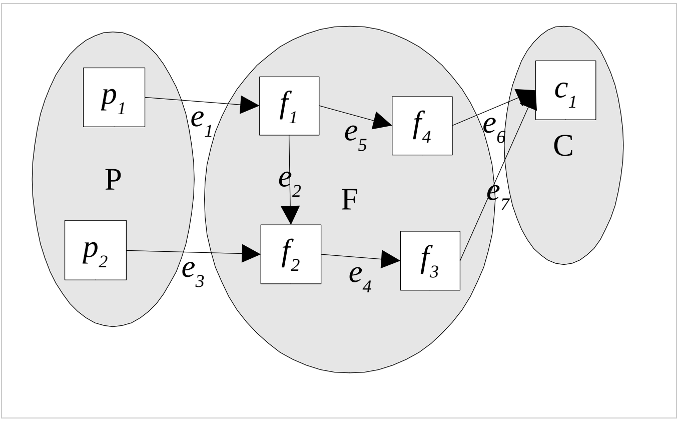


FIGURE 3: A Sample Filter Graph

It is easy to define an isomorphism between the graph representation of VMDs and filter graphs. This, however, would not correctly reflect the corresponding semantic relationship. A filter graph specifies an *instantiable* media transformation whereas a VMD describes a *virtual* media transformation (thus, we could call the corresponding graph a *virtual filter graph*). "Instantiable" means that each media producer is a unique ma-

[6]The graph model also appears being attractive for visualization of VMDs at the client.

terialization, each filter has an implementation, and all input data formats meet the respective requirements. Hence, if we assume that for each source object in a VMD exists at least one materialization and for each operation exists at least one implementation, then the conclusion is that for this VMD exist $n \in [0..\infty]$ *semantically* equivalent filter graphs. Consequently, VirtualMedia's main optimization problem is finding the cost-optimal filter graph for a given VMD (if one exists).

3.2.2 Data Model for Media Objects and Filters

To support the transformation of request graphs into instantiable filter graphs a data model for virtual filter graphs is introduced. Such a VirtualMedia filter graph may contain both virtual elements and real (or instantiable) elements.

We define an object-oriented data model describing media object types and media filter types. The MO part of the model does not define a (traditional) media type hierarchy. Instead, all attributes of an MO like main type, subtype, encoding, and further optional characteristics are modeled as properties which may be dynamically assigned to MOs as a signature (generally denoted with σ). The assignment of contradictory properties may be prevented by defining appropriate constraints. We believe that this approach is more flexible and extensible than a type hierarchy built on inheritance and, thus, better supports the framework character of VirtualMedia.

The filter part of the data model describes both virtual filters and instantiable (implemented) filters. A filter is characterized by its functional and non-functional properties. The functional properties are defined as a set of input and output signatures. These signatures are interpreted differently depending on the filter being virtual or instantiable. If a virtual filter specifies input or output signatures, these are considered as part of its *semantics*. If an instantiable filter specifies input or output signatures it specifies *requirements* on actual input-MOs and *assertions* on actual output-MOs. That means, a filter implementing a virtual filter is not required to specify "compatible" signatures. To give an example: Let the virtual filter *F* say its input should be audio, then we could imagine an implementation of *F* accepting video as input (but, of course, affecting only the audio part).

By non-functional filter properties we mean features like resource consumption, computational complexity, or quality degradation coefficients. Considering such properties during transformation request resolution sounds quite reasonable. How this should be realized, however, has not yet been examined in detail. Whether there exist meaningful non-functional properties of virtual filters that are to be modeled and considered by graph transformation rules, is also still an open question.

Resuming the running example, the VMD (Fig. 2) may now be translated into a VirtualMedia filter graph according to the data model introduced above. The resulting graph is shown in Fig. 4. The start node p_1 of this graph is the video object which is the source object of the transformation request. The two virtual objects of the transformation request become end nodes c_1 and c_2 of the graph. The end nodes are attributed with the requested MO signatures. The transcript operation specified in the request is turned into an according *virtual media filter* f_1, which is placed within the data flow from the source object to the text object c_2. At this time, f_1 can only be virtual because its input is virtual.

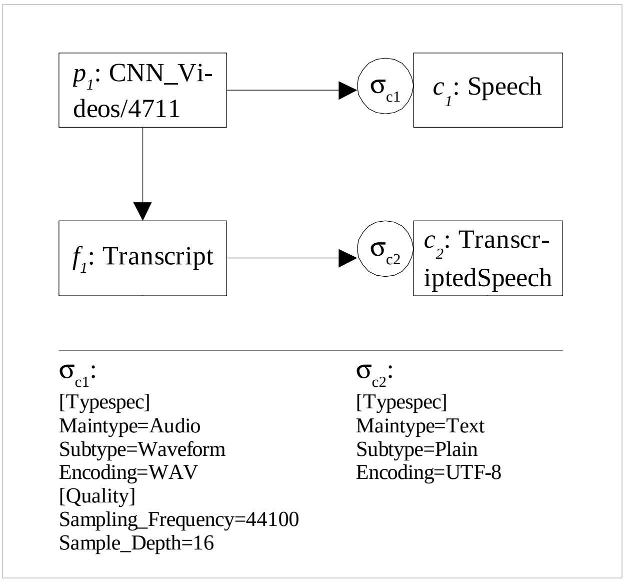


FIGURE 4: A Sample VirtualMedia Transformation Request Graph

3.3 Transformation Request Resolution

In what follows, we will describe some characteristic steps performed during request resolution. The overall goal of this process is to find a semantically equivalent graph containing instantiable filters instead of virtual filters and materializations instead of VMOs. Thus, we have to consider how materializations should be represented in the VirtualMedia model and how to get rid of the “virtual nodes” in request graphs.

3.3.1 Materialization Graphs

For representing the materialization of VMOs we use the same data model as introduced above. That means, a so-called materialization graph of a VMO describes how certain physical data objects form the materialization of this VMO. Fig. 5 shows a possible materialization graph for the VMO “CNN_Videos/4711” of our example.

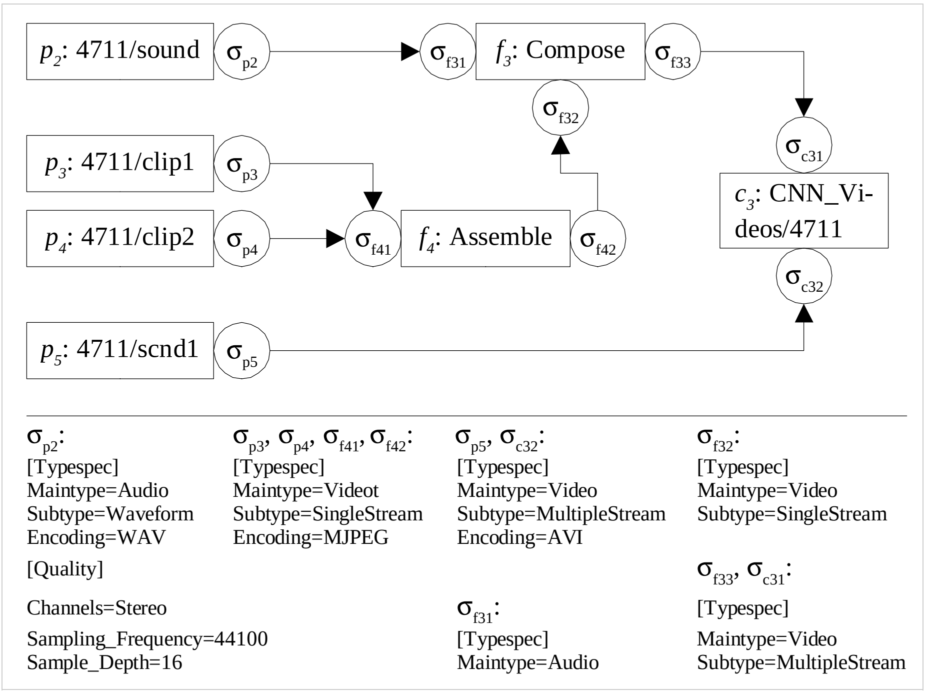

p_2: 4711/sound
σ_{p2}
σ_{f31}
f_3: Compose
σ_{f33}
σ_{f32}
σ_{c31}
p_3: 4711/clip1
σ_{p3}
c_3: CNN_Videos/4711
p_4: 4711/clip2
σ_{p4}
σ_{f41}
f_4: Assemble
σ_{f42}
σ_{c32}
p_5: 4711/scnd1
σ_{p5}
σ_{p2}: [Typespec] Maintype=Audio Subtype=Waveform Encoding=WAV [Quality] Channels=Stereo Sampling_Frequency=44100 Sample_Depth=16
σ_{p3}, σ_{p4}, σ_{f41}, σ_{f42}: [Typespec] Maintype=Videot Subtype=SingleStream Encoding=MJPEG
σ_{p5}, σ_{c32}: [Typespec] Maintype=Video Subtype=MultipleStream Encoding=AVI
σ_{f31}: [Typespec] Maintype=Audio
σ_{f32}: [Typespec] Maintype=Video Subtype=SingleStream
σ_{f33}, σ_{c31}: [Typespec] Maintype=Video Subtype=MultipleStream

FIGURE 5: A Sample VirtualMedia Materialization Graph

We distinguish three types of materializations: primary, secondary, and derived materializations. The first two types occur in Fig. 5: p_2, p_3, and p_4 are primary materializations and p_5 is a secondary materialization. Derived materializations will be considered later. Primary materializations are supplied at the create-time of the VMO and are assumed to provide the maximum available quality of the VMO. Consequently, these materializations may not be altered or destroyed unless the VMO is destroyed itself. On the other hand, secondary materializations are created by the server purely for optimization purposes and usually without informing the applications. Hence, the server may create or destroy secondary materializations whenever this seems likely to improve the system’s performance. As shown in the example, materialization graphs can also contain media filters. In contrast to the transformation request graph, however, these media filters may already be instantiable like f_4. This is possible because the input data formats of materializations are always known.

3.3.2 Graph Transformation

Generally, there are two kinds of graph transformations called (1) replacement (of one node by another one or by a subgraph) and (2) adjustment (adding or removing a node). The replacement steps are the driving parts of the transformation because the “virtual nodes” are replaced by (more) real ones. Any replacement may induce a number of adjustment steps necessary to make the signatures compatible.

Thus, in the case of the example, the following two tasks are to be accomplished:

1. Finding an appropriate materialization of the object “CNN_Videos/4711”, and

2. Replacing the virtual filter “Transcript” with its optimal (if several are found), semantically correct implementation.

Having a transformation request graph and a matching materialization graph we can merge them by unifying the corresponding VMO-nodes p_1 of the request graph (Fig. 4) and c_3 of the materialization graph (Fig. 5). (This is not fully shown in the following figures due to space limitations.) The resulting graph now offers different materializations to be alternatively utilized in fulfilling the request. The final choice depends on the subsequent process of finding implementations for virtual filters and adapting media stream types and formats.

Semantic Equivalence Relations

The graph transformation is driven by rules which are derived from a number of equivalence relations concerning (sets of) filters and MO-signatures. These equivalence relations are considered as being part of the VirtualMedia data model.

Notice that how ever we constitute our data model and equivalence relations they will probably not conform to *any* application’s semantics. This is because such an abstract model can not consider all the media properties an application might depend on. Hence, an application programmer should be aware of this model and the equivalences it defines in order to avoid erroneous transformation requests. Since application neutrality is a major objective of VirtualMedia, only equivalences are defined on which the majority of applications could agree.

There are three basic semantic equivalence relations:

Semantic Neutrality: Classifying a filter as being semantically neutral means it may (in principle) be inserted anywhere in a VirtualMedia graph (or removed) without changing the semantics of the graph. Obviously, putting all the format conversion filters in this equivalence class is crucial for automatic format adaptations to work. Actually, the formal VirtualMedia model defines several different context-sensitive (with respect to media signatures) varieties of semantic neutrality.

Semantic Reversibility: Some filter operations are reversible by corresponding inverse filters. This means, connecting a reversible filter with its inverse filter yields a semantically neutral filter pair. Hence, if such a pair occurs in a VirtualMedia graph it may be removed safely. At first glance, inserting such a pair does not appear to make much sense. An important exception, however, is the composition and decomposition of multiple-stream MOs, which is discussed below.

Semantic Permutability: If the sequence in which two filters are applied to an MO does not matter, they are permutable without changing the graph semantics. Besides being stated a priori, permutability may also be stated ad hoc in a transformation request: A single transformation can contain several operations on the same source. If there are no specified input/output dependencies between these operations, they are considered permutable. Instead of permuting such permutable filters it is also possible to merge them in a multiple-filter node (*super-filter*), thus deferring the decision on the actual sequence to a secondary optimization step.

All these basic relations only lead to adjustment rules. To continue our example, however, we need a replacement rule for exchanging the virtual transcript-filter. This rule is based on a relation called *semantic assimilation*.

Semantic Assimilation

The semantic equivalence between a virtual filter and a possible implementation of this filter is called semantic assimilation (cf. Fig. 6). The implementation of a virtual filter f_v consists of an instantiable filter f_i implementing the semantics of f_v and an arbitrary number of additional filters. The additional filters may be located before and after f_i. They must either be semantically neutral or otherwise a filter f *before* f_i must be followed by its inverse f^{-1} *after* f_i where (f, f^{-1}) conform to the generalized reversibility semantics (explained in the following section on (de-)composition). An implementation is called *complete* if (1) all filters are instantiable, and (2) the signature distance[7] between start and end point of all edges is zero.

The primary rule derived from the semantic assimilation relation is that a virtual filter may be replaced by another filter with the same semantics but more specific signatures. Afterwards, some adjustment rules may be applied to “complete the implementation”.

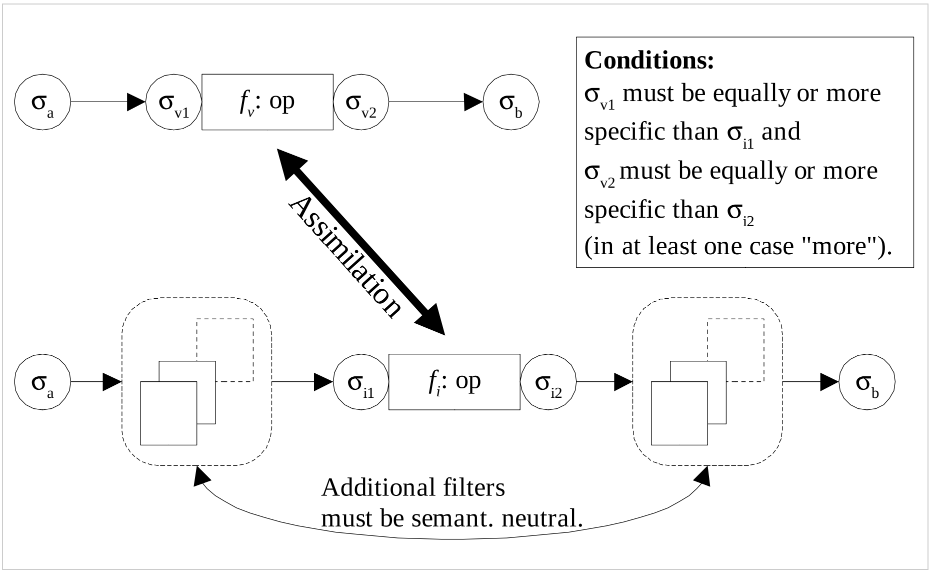

σa
σv1
fv: op
σv2
σb
Conditions:
σv1 must be equally or more specific than σi1 and
σv2 must be equally or more specific than σi2
(in at least one case "more").
Assimilation
σa
σi1
fi: op
σi2
σb
Additional filters must be semant. neutral.

FIGURE 6: The Equivalence of Virtual Filters and (possible) Implementations defined as Semantic Assimilation

Fig. 7 shows (a part of) the example graph after replacing the virtual transcript-filter with a suitable non-virtual version. Note that the signature σ_{p1} of the VMO has been gained from merging the request graph with the materialization graph.

The non-virtual transcript-filter provides a signature σ_{f11} for the input MO and a signature σ_{f12} for the output MO. While the output signature matches the client’s request, i. e., $\Delta(\sigma_{f12}, \sigma_{c2}) = 0$, the input signature does not match the signature of the source MO, i. e., $\Delta(\sigma_{p1}, \sigma_{f11}) > 0$, because the Maintype-properties are different.

Any reasonable implementation of the transcript-operation will most probably operate on audio data. Hence, it is quite unlikely that a transcript-filter exists with $\Delta(\sigma_{p1}, \sigma_{f11}) = 0$. Therefore, an adjustment step is required to complete the implementation of the virtual transcript-filter. Since the transformation request does not specify how the video object is to be converted into an audio object, the resolution algorithm is free to find a suitable converter. In our case, the source MO is a composition of several sub-MOs which is indicated by the property “Subtype=MultipleStream”. Such MOs can be decomposed to restore the single sub-MOs. In order to see how this fact can be exploited for adjustment, the specific semantics of composition and decomposition have to be considered.

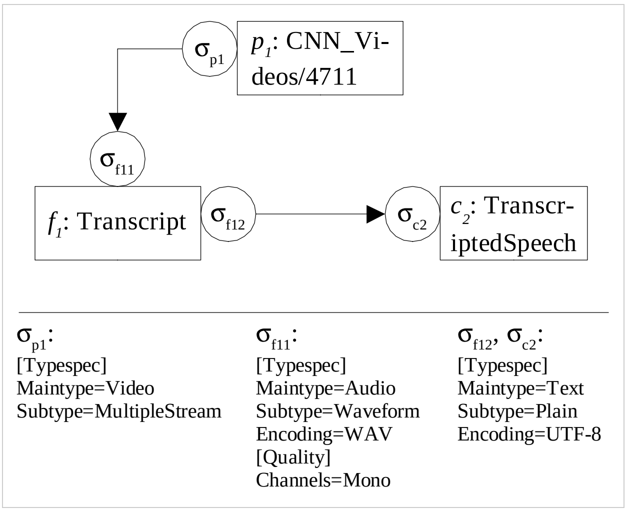


FIGURE 7: Semantic Assimilation of the Transcript-Filter

(De-)Composition Semantics

Filters that compose or decompose multiple-stream MOs work without information loss (by definition). That means, e. g., that a decompose-filter must not only provide all the single streams but also the synchronization information. Thus, compose- and decompose-filters are reversible. Since no information gets lost they are also kind of semantically neutral. Anyhow, only two of four possibilities to insert/remove a

[7]The signature distance is a discrete distance function Δ defined on media signatures based on a classification of the parts (properties) of a signature.

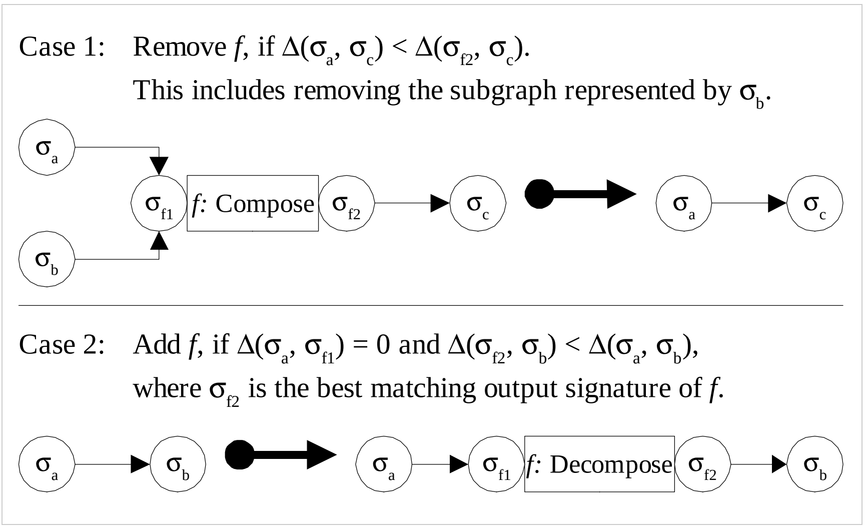

Case 1: Remove f, if $\Delta(\sigma_a, \sigma_c) < \Delta(\sigma_{f2}, \sigma_c)$.
This includes removing the subgraph represented by σ_b.
σ_a
σ_b
σ_{f1}
f: Compose
σ_{f2}
σ_c
σ_a
σ_c
Case 2: Add f, if $\Delta(\sigma_a, \sigma_{f1}) = 0$ and $\Delta(\sigma_{f2}, \sigma_b) < \Delta(\sigma_a, \sigma_b)$,
where σ_{f2} is the best matching output signature of f.
σ_a
σ_b
σ_a
σ_{f1}
f: Decompose
σ_{f2}
σ_b

FIGURE 8: Exploiting Semantic Neutrality of (De)compose-Filters

(de)compose-filter are reasonable (cf. Fig. 8), which is the reason why the graph transformation arrows are pointing only to the right.

The semantic reversibility of (de)compose-filters may be exploited for applying filters to single sub-MOs of a composed MO. In case 1 (cf. Fig. 9) a filter *f* gets embraced by a decompose/compose pair (*f* may as well represent a whole subgraph). Thus, the definition of reversibility is generalized in a sense that all other filters (i. e., not only neutral filters) are allowed in-between a decompose/compose pair which is newly inserted into a VirtualMedia graph. In case 2 the embracing of a multiple-filter node is shown. The filters in a multiple-filter node may apply to different sub-MOs of a composed MO (depending on their signature). Since the filters are classified as permutable there are by definition no semantic dependencies between them. Hence, it is possible to split the multiple-filter node when it gets embraced by a decompose/compose pair.

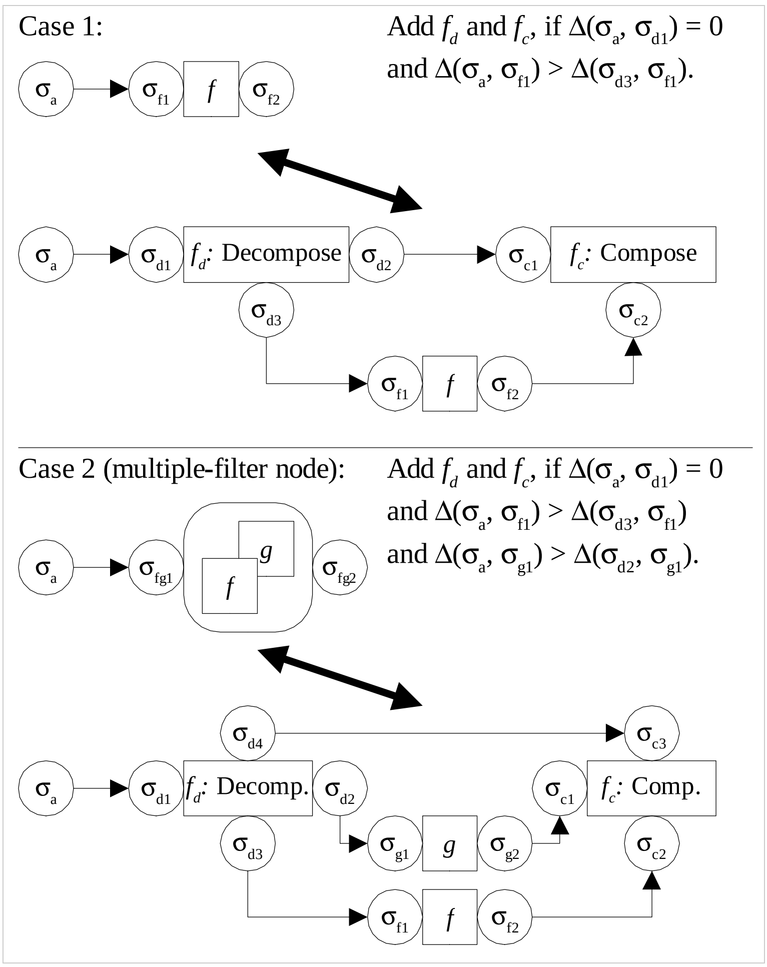


FIGURE 9: Exploiting Semantic Reversibility of (De)compose-Filters

We may now continue our example by applying the rule of Fig. 8, case 2, as the first adjustment step. This results in a graph with a newly added decompose-filter, which is (partly) depicted in Fig. 10.

The next adjustment step exploits the reversibility relation between compose- and decompose-filters in order to simplify the current graph (adjustment by reduction). Fig. 11 motivates this by showing another excerpt of the graph in Fig. 10 shifting the focus to the compose-filter f_3 (originating from the merged materialization graph) and the newly inserted decompose-filter f_2. Since the VMO-node p_1/c_3 between these two filters is functionally neutral, they are actually direct neighbors neutralizing each other semantically. Thus, we can eliminate both the compose-filter and the decompose-filter (and, of course, also the VMO-node).

Concluding the Example

There is now only little work left to complete the optimal[8] transformation prescript graph of our example. After removing the compose/decompose pair the (primary) materialization p_2 of the soundtrack of the video becomes directly connected to both the transcript-filter f_1 and the end node c_1 “Speech”. Thus, we find that $\Delta(\sigma_{p2}, \sigma_{c1}) = 0$, but $\Delta(\sigma_{p2}, \sigma_{f11}) > 0$. However, the non-zero result in the latter evaluation only comes from a different value of the quality-property “Channels” (stereo vs. mono). Hence, assuming a suitable[9] converter-filter is available, we perform a final adjustment step by adding this filter between p_2 and f_1 (adjustment by addition). Fig. 12 shows the resulting final transformation prescript graph realizing the sample transformation request (cf. Fig. 2 on page 8).

In this section we demonstrated (by example) how the transformation request resolution process is *expected* to work. The basic rules driving this process have been introduced and their application has been shown in the example: An implementation of the virtual transcript-filter is found by semantic assimilation. The optimal source object is found by exploiting both semantic neutrality (insertion of the decompose-filter) and semantic reversibility of (de-)compose-filters (elimination of the compose-decompose pair). Finally, a format adaptation is realized by inserting a semantically neutral (more precisely: content-neutral) converter. The next section discusses the algorithm we propose for request resolution.

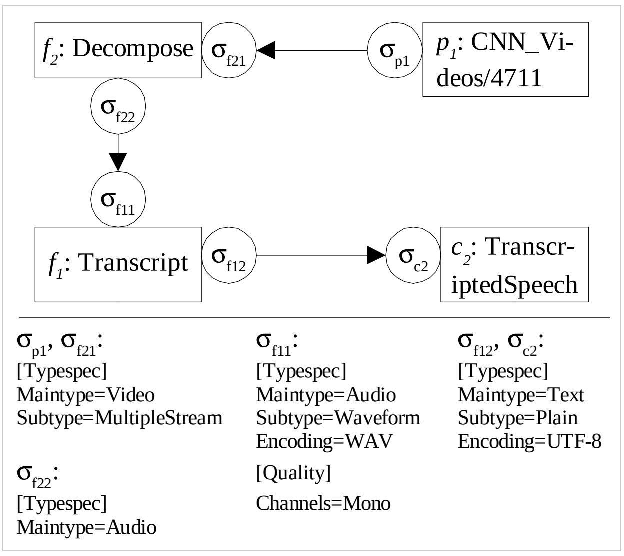

f_2: Decompose
σ_{f21}
σ_{p1}
p_1: CNN_Videos/4711
σ_{f22}
σ_{f11}
f_1: Transcript
σ_{f12}
σ_{c2}
c_2: TranscriptedSpeech
σ_{p1}, σ_{f21}: [Typespec] Maintype=Video Subtype=MultipleStream
σ_{f11}: [Typespec] Maintype=Audio Subtype=Waveform Encoding=WAV [Quality] Channels=Mono
σ_{f12}, σ_{c2}: [Typespec] Maintype=Text Subtype=Plain Encoding=UTF-8
σ_{f22}: [Typespec] Maintype=Audio

FIGURE 10: Adding a Semantically Neutral Decompose-Filter

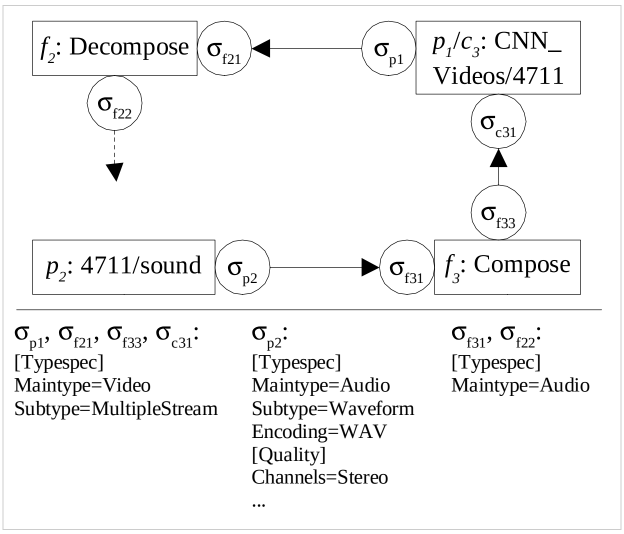

f_2: Decompose
σ_{f21}
σ_{p1}
p_1/c_3: CNN_Videos/4711
σ_{f22}
σ_{c31}
σ_{f33}
p_2: 4711/sound
σ_{p2}
σ_{f31}
f_3: Compose
σ_{p1}, σ_{f21}, σ_{f33}, σ_{c31}: [Typespec] Maintype=Video Subtype=MultipleStream
σ_{p2}: [Typespec] Maintype=Audio Subtype=Waveform Encoding=WAV [Quality] Channels=Stereo ...
σ_{f31}, σ_{f22}: [Typespec] Maintype=Audio

FIGURE 11: The Compose-filter f_3 Neutralized by a Decompose-filter f_2 (p_1/c_3 is a Neutral VMO-Node)

3.4 Considerations on Graph Transformation Algorithms

All graph transformation rules (except graph composition) can be derived from the equivalence relations defined in the previous section. Obviously, these rules are applicable to drive the transformation of a VirtualMedia graph in very different directions, some of which will probably not lead to an acceptable result. What constitutes an acceptable result, however, may be defined in various ways, e. g.:

1. A complete implementation of the client’s transformation request.
2. A complete implementation, optimized according to one of the following criteria: resource consumption (min.), delivery latency (min.), perceivable quality (max.). (This list may still be extended.)
3. A complete implementation with multidimensional optimization (two or more of the criteria listed above).

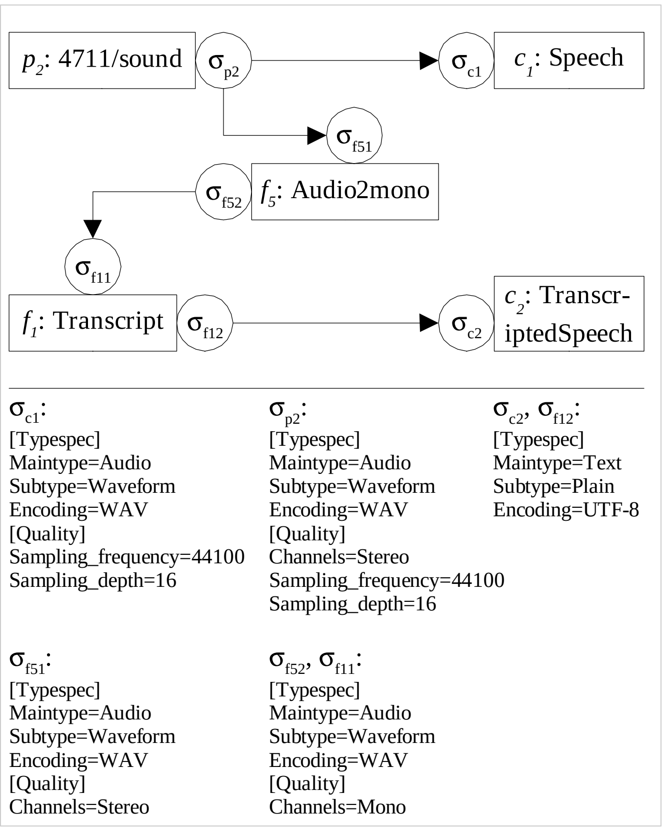
[8]Assuming the (simple) criterion “Use the minimal number of filters to resolve the given request”.

[9]This filter (in this example called “Audio2mono”) should match the given signatures σ_{p2} and σ_{f11} and must be classified as “content-neutral”.

Note that goals (2) and (3) may imply the consideration of non-functional properties as part of signatures. This is still an open problem needing further investigation.

Generally, the number of transformation rules applicable to any given graph lies between 0 and *n*. Hence, we may start by selecting rules according to a breadth-first or depth-first search algorithm, resulting in a search graph with VirtualMedia graphs as nodes and rule application as edges. Breadth-first search will find a solution to (1), if one exists. Since the search graph can contain cycles, a cycle detection must be added to the algorithm, if breadth-first search should find all solutions, which would be necessary to solve (2) or (3).

The size of the search graph does not only depend on the size of the transformation request graph, but also on the size of the affected materialization graphs and the size of the filter base which are both unbound. Hence, if our aim is finding an optimal solution, we should avoid computing all possible solutions, because in a real multimedia system the response time must be short. Thus, a heuristic search algorithm is required.



σ_{c1}:
[Typespec]
Maintype=Audio
Subtype=Waveform
Encoding=WAV
[Quality]
Sampling_frequency=44100
Sampling_depth=16

σ_{p2}:
[Typespec]
Maintype=Audio
Subtype=Waveform
Encoding=WAV
[Quality]
Channels=Stereo
Sampling_frequency=44100
Sampling_depth=16

σ_{c2}, σ_{f12}:
[Typespec]
Maintype=Text
Subtype=Plain
Encoding=UTF-8

σ_{f51}:
[Typespec]
Maintype=Audio
Subtype=Waveform
Encoding=WAV
[Quality]
Channels=Stereo

σ_{f52}, σ_{f11}:
[Typespec]
Maintype=Audio
Subtype=Waveform
Encoding=WAV
[Quality]
Channels=Mono

FIGURE 12: The Final Transformation Prescript Graph

There exist many such algorithms [MF00] and it is not obvious which one to choose. However, since the selection of an algorithm is considered being part of the implementation of the VirtualMedia model (but not part of the model itself), we give some hints on which approach appears to be useful and which one does not.

The simplest applicable approach is the greedy algorithm. This algorithm, however, is likely to get stuck in a local optimum, thus missing the global optimum we are looking for. There are several traditional algorithms which avoid this pitfall, e. g., branch-and-bound or the A* algorithm. For these algorithms to work, the cost function evaluating the benefits of applying a certain rule must meet certain conditions. For A*, e. g., it must always behave monotonic on the path from the request graph to the (optimal) solution graph. This is impossible if the cost function is only based on the signature distance, because replacement rules usually increase the distance while adjustment rules generally decrease the distance. Hence, another function would be needed to measure the distance between a virtual filter and a non- (or less) virtual filter, which would compensate the increase of signature distance. Another interesting yet less traditional approach are evolutionary algorithms. Such an algorithm may also miss the global optimum but usually finds a better solution than the simple greedy algorithm.

Finally notice, that a divide-and-conquer approach (especially dynamic programming) is not useful because of the context-sensitivity of most of the rules. That means, combining optimal solutions of subproblems (i. e., subgraphs of the request graph) does not (generally) yield an optimal solution of the global problem. Thus, the dynamic programming preconditions are not met.

3.5 Reusing Derived Materialization

When a prescript graph is finally instantiated and executed, each MO represented by an edge in the graph has to be materialized. Usually, this materialization is volatile, because the client is only interested in the delivery of the MOs specified in the transformation request. The materialization, however, could also be made persistent without bothering the client. This leads to the opportunity of creating a (redundant) materialization for later reuse. The decision (to be made by the server), which MO is to be materialized permanently, could be made depending on processing costs (make "expensive" MOs persistent), statistics (make frequently requested MOs persistent), and the like.

A "materialized edge" can be located (1) on the path between a physical MO and a VMO or (2) on the path between a VMO and a client-requested MO. In case (1), it is called a secondary materialization, because it may serve as a substitute for the corresponding primary materialization of the VMO (cf. Fig. 5, pg. 11). In case (2), it is called a *derived materialization* (DerMat), because it is the materialization of an MO that has been derived from the VMO at the beginning of the path. An interesting question is how to determine whether a DerMat could be reused in resolving a given transformation request.

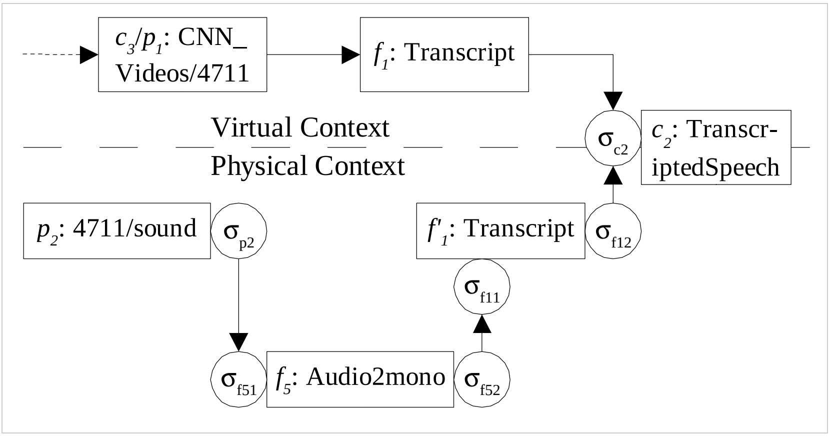


FIGURE 13: Adding a Derived Materialization c_2 to the Materialization Graph (old parts of the graph not shown)

In order to be reusable, a DerMat must be represented in the corresponding materialization graph. This is demonstrated by continuing the running example. As a motivation, assume the implementation of 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, cause the server to keep the transcript internally as a DerMat, since that would increase the QoS dramatically if it were ever requested again by a client.

Fig. 13 shows both possibilities how this DerMat could be reasonably represented in the materialization graph. The first is the *virtual context*, i. e., a subgraph of the original request graph containing all paths ending in the DerMat. The second is the *physical context*, i. e., a subgraph of the final prescript graph containing all paths ending in the DerMat. The virtual context represents the application semantics of the DerMat. Hence, given a transformation request referencing this VMO, the corresponding request graph has to be matched against all virtual contexts of the VMO's materialization graph.

In the case of an exact match, the corresponding subgraph of the request graph may be replaced by a node representing the DerMat. A partial match occurs if a subgraph is detected that contains all nodes of the virtual context, but has additional nodes in-between and/or different topology. In this case, equivalence transformations may be applied to find an exact match.

In case a DerMat can be reused, the trade-off between the costs of recomputing the materialized MO and a possibly lower quality of the DerMat has to be considered. This is due to media filters reducing the quality of the media (which may not always be intended). Hence, recomputation of a DerMat applying different filters (e. g., improved implementations that have become available after the initial creation of the DerMat) may yield a better media quality. This (possible) quality gain can be assessed by evaluating the physical context of the DerMat.

4 Related Work

The transition from our earlier MADT approach to the VirtualMedia approach also means switching from a local to a more global approach regarding MADT design and solving the optimization and other problems. To the best of 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].

4.1 Enhanced ADTs in Predator

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 the E-ADTs' ability to optimize complex operations. Complex operations are combinations of elementary E-ADT-operations, e. g. *Clip(Sharpen(G.Photo), 0, 0, 100, 200)* (example 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 instantiable 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 "semantic permutability" equivalence relation 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 are generally specified explicitly as part of the signature of the client-requested MOs 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 would perform 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 these 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). Moreover, 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 as sufficiently be optimized by E-ADTs as we can do in 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 unintentionally 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 their resources with an (OR-)DBMS.

4.2 Other Media Servers

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 means, 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 by far not as versatile and flexible as VirtualMedia or E-ADTs. The idea of introducing primary and secondary materialization in VirtualMedia, however, originates from this work.

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, [Inf97a]). While the vast majority of these media extensions do not provide physical data independence, two exceptions from this rule should be pointed out: (1) In [HSH+98] a continuous media DataBlade providing device independence, location transparency, and presentation independence is described, and (2) [WHK99] presents a DB2 Extender for images providing format independence where materialization is controlled through cost-based optimization.

4.3 Optimization

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 and although they differ semantically there exist 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 instantiable 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, e. g., 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 our filters, however, is not stemming from explicit type inheritance, but from a deducible media 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 additionally motivates our algebraic approach for realizing filter graph transformation and optimization, since 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., algebra, algorithms, verification methods, languages, and tools).

5 Conclusions

In this paper, abstractions and concepts for multimedia systems providing physical data independence 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 different materialization of a media object.
- *Optimize media transformations* globally, i. e., (1) by considering the transformation request as a unit 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 an 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. Semantic equivalence relations defined for such VirtualMedia graphs allow for transforming request graphs into (instantiable) prescript graphs while applying different optimization strategies like materialization or cost-based evaluation of semantically equivalent graphs.

The VirtualMedia graph transformation and optimization algorithm will continue being the main objective of our work in the near future. Besides that, ongoing research also focuses on the refinement of several other aspects of the VirtualMedia concept and on the exploration of several open questions. To name a few, the following aspects are of particular interest:

- Enhancing the data model with, e. g., hierarchical structures (explicit subgraphs) or a template concept (parameterized VMOs),
- development of a reference architecture for a VirtualMedia server exploiting the potential of peer-to-peer metacomputing environments,
- integrating adaptive QoS (feedback-controlled) and interaction (including interactive filters), and
- several API issues, e. g., improvement of the XML-based VirtualMedia language and consideration of non-functional properties as part of signatures.

Recently, the major database system manufacturers introduced so-called universal database systems, thus claiming the ability to manage any kind of data. None of these commercial systems, however, realizes transformation independence or at least physical data independence, which we consider crucial for MM-DBMSs. Hence, the question what should actually be the fundamental nature of a genuine *universal* media server is still to be discussed.

Acknowledgments

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

6 References

[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.

[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.

[MF00] Michalewicz, Z., Fogel, D. B.: How to Solve It: Modern Heuristics. Berlin, Heidelberg: Springer-Verlag, ISBN 3-540-66061-5, 2000.

[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.

[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.

[WHK99] Wagner, M., Holland, S., Kießling, W.: Towards Self-tuning Multimedia Delivery for Advanced Internet Services. In: Proc. 1st Int. Workshop on Multimedia Intelligent Storage and Retrieval Management (MISRM '99) in conjunction with ACM Multimedia Conference, Orlando, Florida, Oct. 1999.