Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Systems 30 (2005) 444–466
www.elsevier.com/locate/infosys

Towards a theory of multimedia metacomputing ☆

Ulrich Marder*, Theo Härder

Database and Information Systems Group, Department of Computer Science, University of Kaiserslautern, P.O. Box 3049, D-67653 Kaiserslautern, Germany

Received 23 May 2003; received in revised form 1 May 2004; accepted 1 June 2004

Abstract

Multimedia metacomputing is a new approach to the management and processing of multimedia data in web-based information systems. It offers high flexibility and openness while shielding the applications from any system internals. Starting with the vision of a completely open and globally distributed multimedia information system, we consider abstraction concepts required, especially transformation independence, and an appropriate semantic model.

Thus, the major focus of this paper is 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 instantiatable plans for materializing the requested media object(s) at the client.

MSC: primary C.2.4; D.2.11; H.2.4; H.3.5

Keywords: Media Data abstractions; Transformation independence; Distributed computing; Multimedia information systems

1. Introduction

We may consider multimedia information systems (MMIS) an enabling technology for infotainment: the integration of information, communication, and entertainment. While traditional MMIS only existed as "islands", e.g., on CD-ROMs or local networks, today—thanks to the world-wide web—we can easily imagine large, global MMIS supporting millions of users in the office, at home, and on the way. However, providing all of them with the best possible experience and benefit—at any time, any place, and any environment—is still a vision to become true.

One of the most challenging problems in such large and complex systems is the variety of end user devices, which may vary from small mobile devices to powerful multimedia workstations. This

☆ Recommended by Maurizio Lenzerini

*Corresponding author.
E-mail addresses: marder@informatik.uni-kl.de (U. Marder), haerder@informatik.uni-kl.de (T. Härder).

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

0306-4379/$ - see front matter © 2004 Elsevier B.V.. All rights reserved.
doi:10.1016/j.is.2004.06.001

makes the preplanning of actions to satisfy user requests rather difficult and most often inefficient. The ever increasing bandwidth of the world-wide networking infrastructure and the virtually unlimited number of potential server systems, however, enable new solutions for multimedia service providers to approach these challenges in a more dynamic and flexible manner. For instance, frequently required services can be offered by many servers at the same time, complex services can be distributed on several servers, and new services can be integrated easily and any time without the need to change the client software. Depending on the origin, such concepts are sometimes called metacomputing [1], peer-to-peer (P2P) computing [2], or grid computing [3].

However, using the internet as a platform for MMIS today is usually driven by the classic client/server paradigm (cf. Fig. 1). Media assets are managed by servers and can be referenced and located by URLs. While most of the servers are simple HTTP servers, there are also some special (streaming) media servers and multimedia database management systems (MM-DBMS). Generally, these servers provide publishing and retrieval functionality, but very little support for (content-based) searching and processing of media objects. For content-based searching, however, standard internet search engines can be used, e.g., Google™ Image Search [4]. Processing media objects on servers is partially supported by some object-relational DBMS [5–7].

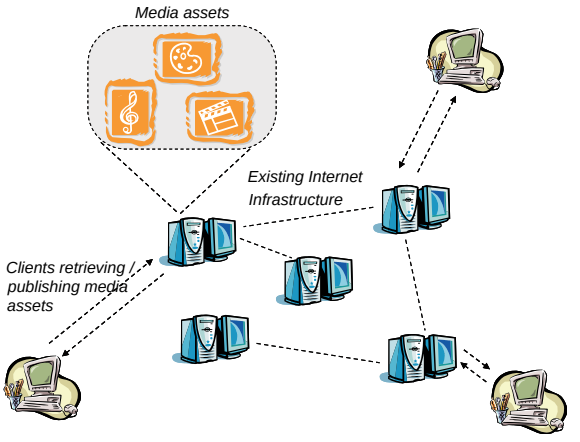

Fig. 1. Common usage of the internet as an MMIS platform.

In contrast to the client/server paradigm, the P2P paradigm relies on a network of coequal peers. Peers can be functionally equivalent, but also specialized on a specific subtask of an application. They can call each other (delegation) and, in the case of functional equivalence, also replace each other (redundancy). In many P2P architectures, there is some kind of registry allowing to integrate or remove peers dynamically. Only recently, some P2P applications like the popular file-sharing networks have gained a lot of attention. The primary objective of file-sharing networks is realizing the distributed storage and management of media files with some limited content-based access like searching by artist, title, etc. The P2P architecture makes it possible to get by with very few central components (e.g., Napster [8]) or even to do without them (e.g., Gnutella [9]).

There are other recently developed concepts for distributed computing like component-based metacomputing [10], web services [11,12], or the famous Seti@Home project [13], which still have to be evaluated for usefulness and deployment in web-based MMIS. Today, media processing functions are usually realized by client modules which rely on certain system software or special devices to be present (and available). Therefore, it is difficult to simply put these functions on the most advanced distributed computing architectures, because they are not enabled for cooperation with remote services—not to mention, dynamic detection and employment of remote functions. This is mostly true also for MM-DBMS which provide some kind of media processing functionality [5,14,15]. Despite the development of quite advanced abstract data models [16,17] these systems still assume a homogenous server environment, thus disregarding global distribution aspects, heterogeneity, redundancy, P2P concepts, dynamic extensibility, etc. These aspects, however, have driven the development of the concept of transformation independence [18], which we will briefly introduce below.

2. Objectives

The general functionality of a web-based, open, heterogeneous, and dynamically extensible MMIS

can be described as multimedia metacomputing [19]. Fig. 2 shows a typical application scenario of such a system.

At first, a content-based query is sent to a search engine ①. The search engine processes the query and returns a list of references to relevant media objects to the client. The realization of such search engines is an issue of current information retrieval research and, hence, not addressed in this paper.

Next, the application (or the user, resp.) has to decide on the subsequent actions regarding the media objects found, e.g., whether—and how—to present them on the client. These actions are described in the form of a so-called *transform&deliver request,* which is transmitted to the multimedia metacomputing service ②.

The multimedia metacomputing service interprets the transform&deliver request and then accomplishes the following tasks ③:

- locating the requested media objects, i.e., their physical representations,
- locating processing resources that are able and willing to execute the requested (or required) media operations,
- computing an optimal plan for executing the media operations,
- activating the data and processing resources (instantiation) and establishing the necessary data channels between them (interconnection).

The execution of the media operations starts either automatically after completion of the planning and instantiation phase or not until receiving a start signal from the client. The delivery of the processed media objects to the client ④ is generally somewhat delayed, depending on the size of the intermediate buffers between the processing steps, the processing time, and possibly synchronization constraints.

The whole extent of research issues to be addressed in the realization of multimedia metacomputing is still increasing. The retrieval problem and the distributed component architectures alone are areas with high activity in research communities and industrial labs. In this paper, however, we focus on a different problem: the "semantic gap" between the application-oriented transform&deliver requests and the application-neutral basic technologies like database systems, web and

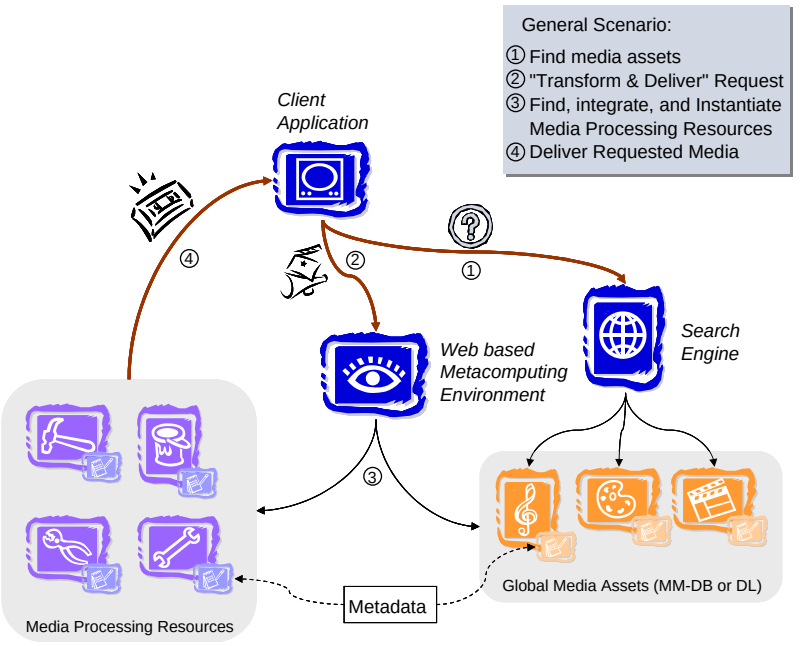

Fig. 2. Principle of multimedia metacomputing.

media servers, component-based middleware, etc. This gap has to be closed by an integrated data and processing model that solves the following problems:

- *Specification of media transformations.* Having no or just incomplete knowledge of the physical representation of media objects, the users should still be able to form these objects according to their imagination. Consequently, transform&deliver requests have to be based on a model allowing the semantically precise specification of media operations without referring to the interfaces of software modules that implement the operation or other hidden resources like, e.g., alternative representations of the media objects.
- *Automatic detection of required operations.* Due to their incomplete knowledge, the applications are not able to specify all operations required for the desired media transformations directly. Generally, there can be several soft- or hardware modules within the MMIS providing a specific processing functionality. Hence, a—not necessarily deterministic—mapping between the more or less descriptive specification of media transformations as a transform&deliver request and the available processing resources is required. For this reason, an important issue is how to describe the functionality of a processing resource such that an algorithm can decide whether it can perform a certain media transformation or not.
- *Dynamic connection and coordination of different processing resources.* Generally, complex media transformations are realized by dynamically combining different processing resources to form a chain or graph of "media processors". This concept, which we also call *coordinated joint processing*, has both logical and physical implications. At the logical layer, the functional properties of processing resources have to be considered, resulting in a model supporting the dynamic synthesis and refinement of complex media transformations. At the physical layer, especially distribution and heterogeneity aspects have to be addressed. For instance, the servers on which processing resources are employed as part of a coordinated joint processing must be interconnected by sufficiently reliable and fast networks. These processing resources must also be able to communicate using common protocols in order to exchange and to synchronize media streams where required (e. g., if two streams meet at one processing resource).

In the remainder of this paper, we will particularly address theoretical foundations for multimedia metacomputing. In Section 3, several general media data abstractions described in the literature are discussed. We point out that our newly developed abstraction concept of transformation independence is crucial for the multimedia metacomputing approach. Accordingly, the VirtualMedia concept, a formal realization of transformation independence, is presented in Section 4 in some detail. Section 5 considers the formalization of VirtualMedia focusing on the fundamental definitions, followed by some ideas on advanced request processing issues discussed in Section 6. We conclude the paper in Section 7, also summarizing some results of the architectural considerations already presented in [19].

3. Media data abstractions

A major requirement that immediately follows from the given objectives is data independence. This abstraction concept is realized in most today's DBMS in order to support a variety of external representations of the data that are specific to a certain application or host language. Media data, however, is usually composed of binary raw data and structured registration data which provides information that is required to interpret the raw data appropriately, e. g., a format identifier. Hence, it is reasonable to distinguish between logical and physical data independence [20]. Logical data independence means that all metadata (from which the registration data is only a part) is modeled and managed independently from the format of the raw data. This requires, e.g., a logical addressing scheme allowing references to spatial or temporal coordinates within the raw data that are stable even in

the case of a format conversion. If content-based features of the media object are included in the metadata, it is also possible to support content-based searching independently from the raw media data.

Physical data independence, however, is targeted directly on the physical representation of the media object, i.e., the internal structure or schema of the raw data, which is also often referred to as *format* or *encoding*. Therefore, we generally use the term *format independence* as a synonym for physical data independence. In our application scenario, format independence is crucial because of the contradicting requirements of clients and servers regarding the properties of the media formats:

- The servers need lossless, universal formats that preserve the highest data quality and support all possible operations with a maximum of performance and quality of service.
- The clients usually need special-purpose formats that are optimal for a given presentation device, internet connection or other specific constraints including data compression and reduction of the data quality, e.g., image size or frame rate reduction.

Until today, format independence has only been realized successfully when the overall functionality of the system, i.e., the variety of media operations supported, was almost reduced to zero. That means, the systems were able to deliver (and receive) media objects in different formats and quality, but without providing any further functionality (except searching on metadata) [15,20]. This phenomenon which we call the format independence problem is due to the following three implications of format independence:

1. With format independence, each media object that is to be stored in the system has to be converted into a system-determined internal format. This does not only cause a significant loss of performance, but also a potential loss of quality because some very popular compressed media formats like JPEG or MPEG can not be exactly (or at least with equal quality) restored from the uncompressed data.
2. The optimization of a sequence of media operations can not be left to the clients, if the internal format is not revealed. However, without an algebra that allows semantic optimization, the system can not effectively optimize the operations either.
3. Operations may have side effects that can not be reversed. For instance, an operation may update a media object, e.g., applying a smoothing-filter on an image, in a way that renders the object useless for other clients.

Fortunately, none of these implications is mandatory—they rather depend on the traditional method of realizing format independence which was adopting the realization of data independence in common DBMS. With the introduction of the concept of *transformation independence* [18], it was shown that format independence is achievable without running into the problems mentioned above. The pivotal innovation of transformation independence is to abandon the traditional data-centric approach in favor of a more dynamic, operation-centric view. In this approach, format independence is supported by a very flexible materialization concept that distinguishes between primary, secondary, and derived representations of media objects. The primary representation is identical with the original external representation of the media object and protected from any kind of modification, thus avoiding the first and the third implication stated above. The other two kinds of representation may be used for any other purpose like optimization or semantic extension (e.g., providing an alternative representation of a media object). Furthermore, a specific processing concept is defined that poses basic requirements on all media operations like being free of side effects and randomly combinable. Operations that meet these requirements are called *media filters*. An arbitrary combination of such filters applied to a set of media objects is called a *transformation request* or—if it is directly executable—a *media transformation*. The idea behind this distinction is that a transformation request must only express the semantics of an application and that algebra is provided to enable the system to transform the request into a semantically equivalent media

transformation. Besides the semantics-based optimization this request processing can also include cost-based optimization.

The data model implied by transformation independence is more generic than other media data models, which is a tribute to the fact that the—both formal and semantic—polymorphism of media objects in an environment as described in the previous section is at most partially predictable. These generic features in particular prevent the realization of transformation independence based on existing models that already support data independence like, e.g., the MADT (media-specific abstract data types) concept [21]. Hence, a new media data model that is fully based on the generic concepts of transformation independence has been developed.

4. 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.

This section is divided into three subsections dealing with the following fundamental aspects:

- *Transformation requests*. How can a user describe his media processing needs independently from implementation details?
- *Filter graphs*. How can media transformations be formally modeled?
- *Request processing*. What are the semantic and formal foundations for the transformation of filter graphs?

4.1. Transformation requests in VirtualMedia

To illustrate the major aspects of the VirtualMedia concept a continuous 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 media objects (MO) 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. 3 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 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. Because 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 *VirtualMedia objects* (VMO). Because both of these must be materialized at the client, exact type information is required for the external format which is given by specifying a *media signature*. Optionally, a media signature may include quality properties as demonstrated with the second VMO and content properties (see first VMO).

The transformation section is mandatory for each VMO, because even if there is no explicit transformation operation to be specified (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 "implicit" 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 to find 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

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

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

   <VIRTUAL NAME="TranscriptedSpeech" MODE="by_value">
      <SIGNATURE>
         <PROPERTY NAME="MAINTYPE" CLASS="typespec"><IT>TEXT</IT></PROPERTY>
         <PROPERTY NAME="SUBTYPE" CLASS="typespec"><IT>PLAIN</IT></PROPERTY>
         <PROPERTY NAME="ENCODING" CLASS="typespec"><IT>UTF-8</IT></PROPERTY>
         <PROPERTY NAME="LANGUAGE" CLASS="content"><IT>EN</IT></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" MODE="deferred">
      <SIGNATURE>
         <PROPERTY NAME="MAINTYPE" CLASS="typespec"><IT>AUDIO</IT></PROPERTY>
         <PROPERTY NAME="SUBTYPE" CLASS="typespec"><IT>WAVEFORM</IT></PROPERTY>
         <PROPERTY NAME="ENCODING" CLASS="typespec"><IT>WAV</IT></PROPERTY>
         <PROPERTY NAME="SAMPLING_FREQUENCY" CLASS="quality"><IT>44100</IT></PROPERTY>
         <PROPERTY NAME="SAMPLE_DEPTH" CLASS="quality"><IT>16</IT></PROPERTY>
      </SIGNATURE>
      <TRANSFORMATION>
         <OPERATION SEMANTICS="implicit">
            <INPUT ALIAS="i2" REF="BC_Video"/>
         </OPERATION>
      </TRANSFORMATION>
   </VIRTUAL>

</VMDESC>
```

Fig. 3. A sample transformation request (VirtualMedia descriptor).

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. Therefore, this graph structure is a suitable internal model for describing any media transformations requested through VMDs.

4.2. The filter graph model for media processing

Modeling and realizing the processing (i.e., transformation) of media objects as filter graphs is a well-known principle in theory [22,23] and practice [24,25]. However, to our knowledge it has never before been applied to model abstract media data types. In this section, we first present a short, informal overview of the model to support the comprehensibility of the example. More details

regarding the formalization of the model follow later on in Section 5.

4.2.1. Filter graphs

A filter graph is directed and acyclic (cf. Fig. 4). 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 building blocks of 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).

It is easy to define an isomorphism between the graph representations of VMDs and filter graphs. This, however, would not correctly reflect the corresponding semantic relationship. A filter graph specifies an *instantiatable* media transformation, whereas a VMD describes a *virtual* media transformation (thus, we could call the corresponding graph a *virtual filter graph*). "Instantiatable" means that each media producer is a unique materialization, each filter has a determined 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 semantically* equivalent filter graphs ($n \in [0 \cdots \infty]$). Consequently, VirtualMedia's main optimization problem is finding the optimal filter graph for a given VMD (if one exists). The optimality criterion may vary from very simple (e.g., minimize the graph size) to very complex (e.g., minimize the execution costs) as long as it can be evaluated for any given graph.

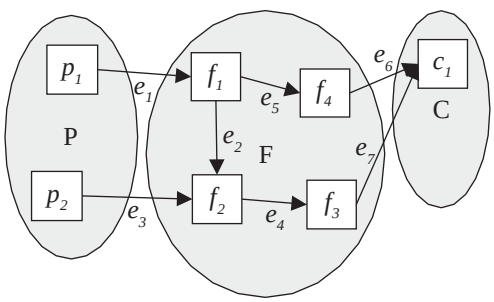

Fig. 4. Illustration of the filter graph model.

4.2.2. Data model for media objects and filters

To support the transformation of request graphs into instantiatable filter graphs, a data model for virtual filter graphs is introduced. Such a VirtualMedia filter graph may contain both virtual elements and real (or instantiatable) 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 instantiatable (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 instantiatable. If a virtual filter specifies input or output signatures, these are considered as part of its *semantics*. If an instantiatable 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 state that 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. Considering such properties when processing requests sounds 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 continuous example, the VMD (Fig. 3) may now be translated into a request graph according to the data model introduced above. The resulting graph is shown in Fig. 5. 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 must be virtual because its input and output signatures are not given.

4.3. Request processing

In the following, we describe some characteristic steps performed during request processing. The general goal of this process is to find a semantically equivalent graph containing instantiatable 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.

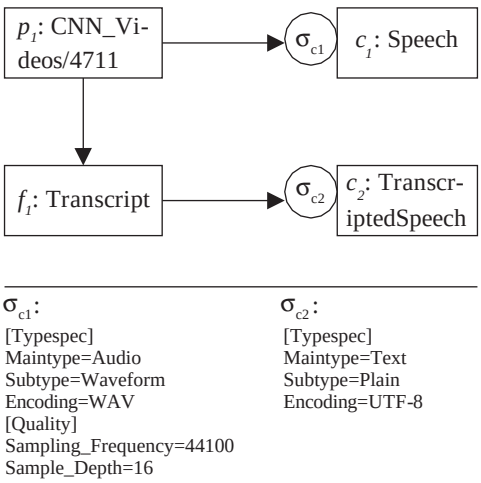

Fig. 5. A sample VirtualMedia request graph.

4.3.1. 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. 6 shows a possible materialization graph for the VMO "CNN_Videos/4711" of our example.

As mentioned earlier, we distinguish three types of materialization: primary, secondary, and derived materialization. The first two types occur in Fig. 6: p_2, p_3, and p_4 are primary materializations and p_5 is a secondary materialization. Primary materialization is supplied at the time of creation of the VMO and is assumed to provide the maximum available quality of the VMO. Consequently, this materialization may not be altered or destroyed unless the VMO is destroyed itself. On the other hand, secondary materializations are either created by the server for optimization purposes, usually without informing the applications, or by a user in order provide an alternative representation. Hence, the server may create 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 instantiatable like f_4. This is possible because the input data formats of materializations are always known.

4.3.2. 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 one or two nodes). 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:

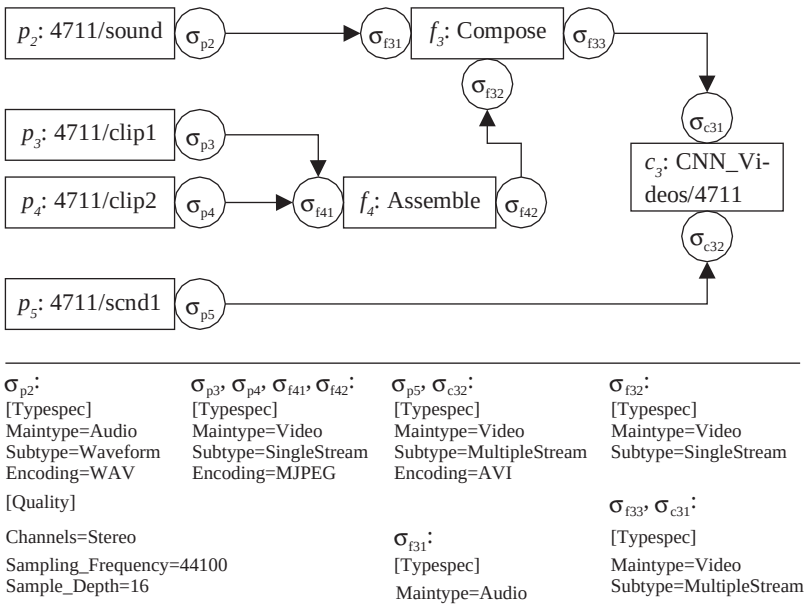

Fig. 6. A sample VirtualMedia materialization graph.

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. 5) and c_3 of the materialization graph (Fig. 6). (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 being part of the VirtualMedia data model (cf. Section 5.5).

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. Because application neutrality is a major objective of VirtualMedia in order to be applicable in a multimedia metacomputing environment, only such 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 by distinguishing between

content neutrality, quality neutrality, and type 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. 7). The implementation of a virtual filter f_v consists of an instantiatable 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 instantiatable, and (2) the signature distance[2] 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".

Fig. 8 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. Because the transformation request does not specify how the video object is to be converted into an audio object, the request processing 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 various 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.

(De-)Composition semantics. Filters that compose or decompose multiple-stream MOs work without information loss (by definition). That means, for example, that a decompose-filter must not only provide all the single streams but also the synchronization information. Thus, compose- and decompose-filters are reversible. Because no information gets lost, they are also kind of

[2]The formal definition of the signature distance function Δ is presented in Section 5.3.

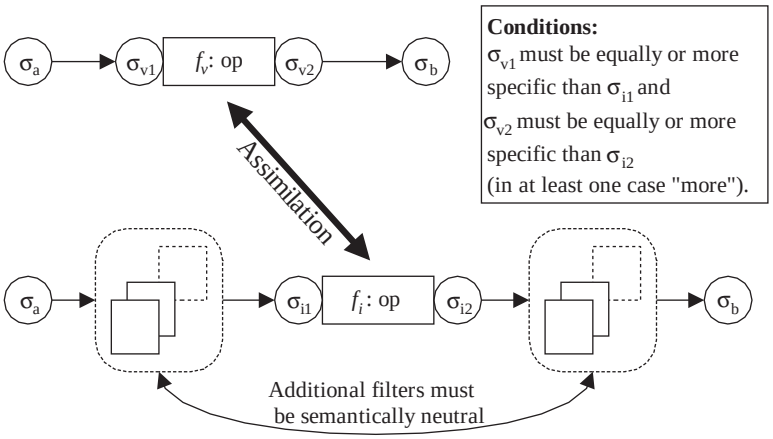

Fig. 7. Illustration of semantic assimilation.

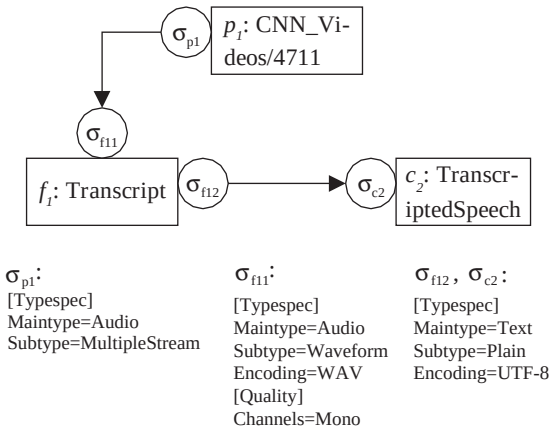

Fig. 8. Assimilation of the transcript-filter.

semantically neutral. Anyhow, only two of four possibilities to insert/remove a (de)compose-filter are reasonable (cf. Fig. 9), 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. 10), 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 embracement 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). Because 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.

We may now continue our example by applying the rule of Fig. 9, case 2, as the first adjustment step. This results in a graph with a newly added decomposition filter, which is (partly) depicted in Fig. 11.

The next adjustment step exploits the reversibility relation between composition and decomposition filters in order to simplify the current graph (adjustment by reduction). Fig. 12 motivates this by showing another excerpt of the graph in Fig. 11 shifting the focus to the composition filter f_3 (originating from the merged materialization graph) and the newly inserted decomposition filter f_2. Because 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 composition filter and the decomposition filter (and, of course, also the VMO node).

Concluding the example. There is now only little work left to complete the optimal[3] solution of our

[3]Assuming the (simple) criterion "use the minimal number of filters to solve the given request".

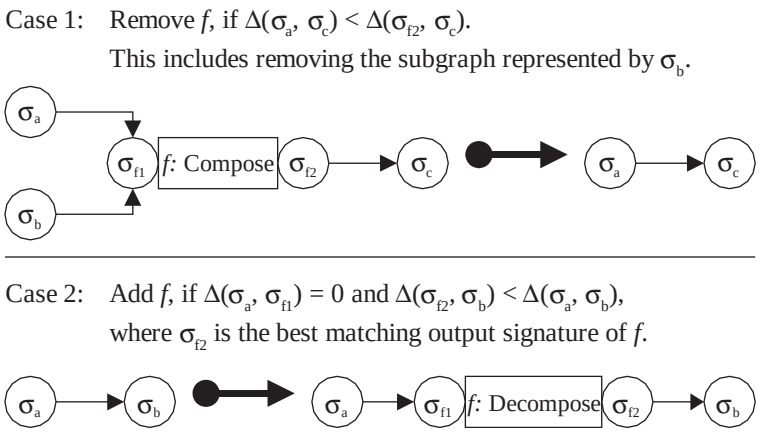

Fig. 9. Exploiting the semantic neutrality of (de-)composition filters.

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[4] converter filter is available, we perform a final adjustment step by adding this filter between p_2 and f_1 (adjustment by addition). Fig. 13 shows the resulting final media transformation graph realizing the sample transformation request (cf. Fig. 3).

In this section, we demonstrated (by example) how the transformation request processing is *expected* to work. The basic rules driving this process have been introduced and their application has been shown by 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 decomposition filter) and semantic reversibility of (de-)composition filters (elimination of the compose/decompose pair). Finally, a format adaptation is realized by inserting a semantically neutral (more precisely: content-neutral) converter. To automate this process a complete formalization of the VirtualMedia concept has been developed.

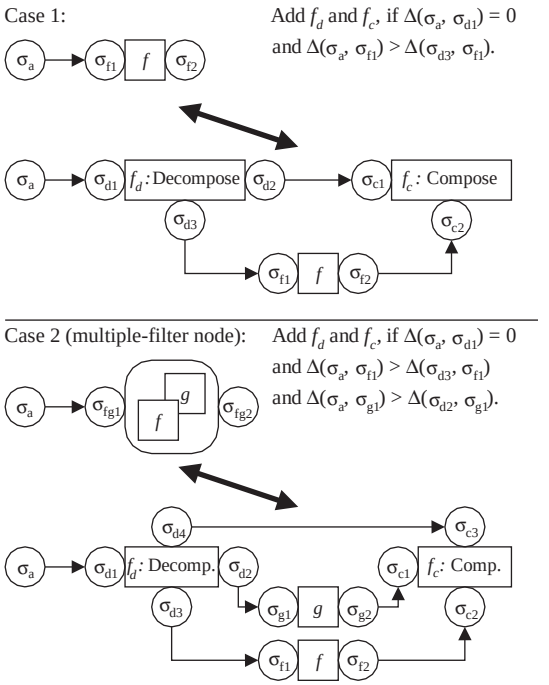

Fig. 10. Exploiting the semantic reversibility of (de-) composition filters.

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

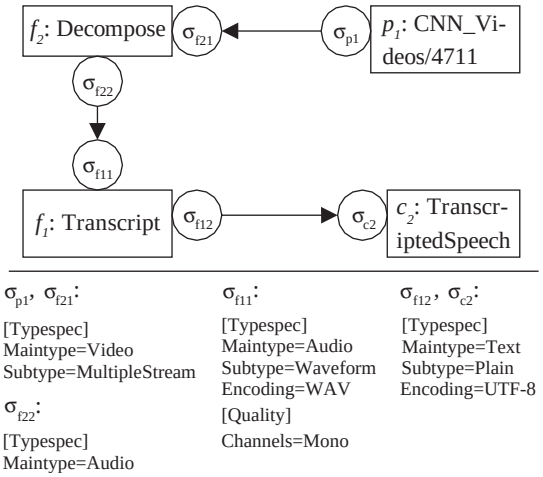

Fig. 11. Decomposition of the video object added.

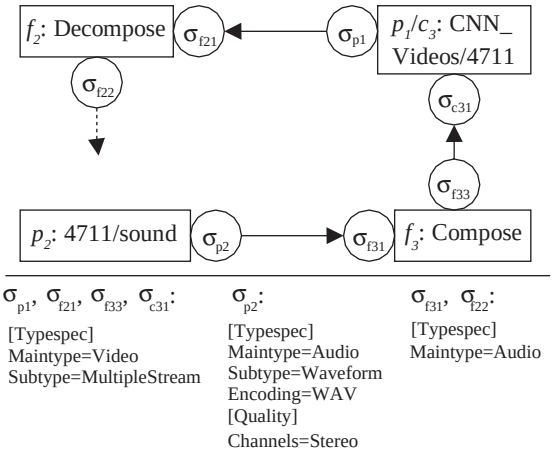

Fig. 12. The composition filter f_3 neutralized by a decomposition filter f_2 (p_1/c_3 is a neutral VMO node).

5. Formal model

5.1. Introduction

In this section, the formalization of the VirtualMedia concept is briefly presented. The formal foundation include

- the general structure and semantics of VirtualMedia graphs and

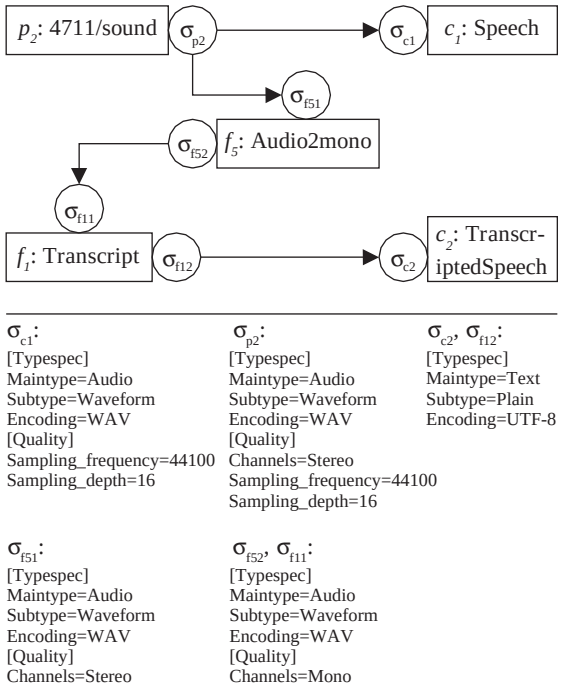

Fig. 13. The final media transformation graph.

- the definition of media-specific semantics by means of properties and signatures.

Advanced aspects of the formalization include a comprehensive set of media-specific semantic equivalence relations, complexity calculations, heuristic score functions and their application in request processing, a hierarchical metadata schema, materialization management, and the definition of the query language VMML. For brevity, we only present the formalization of one equivalence relation and discuss the general ideas of request processing optimization. For a complete presentation of the model refer to [26].

The formalization is based on traditional algebra and first-order predicate logic. Therefore, it can be easily realized using common programming languages. Moreover, common heuristic problem solving approaches [27] are applicable in a straightforward manner. The following subsection starts the formalization by first defining the general structure and semantics of VirtualMedia graphs.

5.2. VirtualMedia graph

A *VirtualMedia graph* (VMgraph) G is a tuple (V, E). V is a set of nodes; E is a set of directed edges. G is free of cycles, i.e., G is a directed acyclic graph (DAG). The edges represent channels transmitting media objects, while nodes represent resources that produce, consume, filter (process, manipulate) or distribute media objects.

An edge $e \in E$ can alternatively be described as a tuple (u, v), where $u \in V$ identifies the start node and $v \in V$ identifies the target node of the edge.

5.2.1. Producers, consumers, and filters

Generally, the nodes of a VMgraph can be subdivided into source, sink, and intermediate nodes building mutually disjoint subsets of the node set V (cf. Fig. 4). In the VirtualMedia model, each of these subsets gets assigned the specific semantics of producers, consumers, and filters, respectively. In the following paragraphs these semantics are formally defined.

Producers. The set of producers $P \subset V$ contains all nodes having no incoming edges, formally,

$$P = \{p \in V | \exists u \in V : (p,u) \in E \wedge \neg \exists v \in V : (v,p) \in E\}.$$

Given that G is acyclic, we conclude: $V \neq \varnothing \wedge E \neq \varnothing \Rightarrow P \neq \varnothing \wedge P \subset V$.

Consumers. The set of consumers $K \subset V$ contains all nodes having no outgoing edges, formally,

$$K = \{k \in V | \exists u \in V : (u,k) \in E \wedge \neg \exists v \in V : (k,v) \in E\}.$$

Given that G is acyclic, we conclude: $V \neq \varnothing \wedge E \neq \varnothing \Rightarrow K \neq \varnothing \wedge K \subset V$.

Filters. The set of filters $F \subset V$ contains all nodes having a minimum of one incoming and one outgoing edges, formally:

$$F = \{f \in V | \exists u, v \in V : (u,f) \in E \wedge (f,v) \in E\}.$$

Given a VMgraph, the following is true: $F = V \backslash (P \cup K)$, iff $\{v \in V | \neg \exists u : (v,u) \in E v(u,v) \in E\} = \varnothing$, i.e., if there does not exist a node having neither incoming nor outgoing edges. In the following, we generally assume that $F = V \backslash (P \cup K)$. The predicate `Filter`, which is used below, is defined as `Filter` $(v) : \Leftrightarrow v \in F$.

5.2.2. Topological classification of filters

We distinguish three topological filter classes:

- *Basic filters* are filters having exactly one incoming and exactly one outgoing edge. The corresponding predicate `Basic_Filter` is defined as follows:

  `Basic_Filter`(v): $\Leftrightarrow$ `Filter`$(v) \wedge$
  $(\forall x, y \in V: (x,v) \in E \wedge (y,v) \in E \Rightarrow x = y) \wedge$
  $(\forall x, y \in V: (v,x) \in E \wedge (v,y) \in E \Rightarrow x = y)$.

  In Fig. 4, f_3 und f_4 are basic filters.
- *Split filters* are filters having exactly one incoming and at least two outgoing edges. The corresponding predicate `Split_Filter` is defined as follows:

  `Split_Filter`(v): $\Leftrightarrow$ `Filter`$(v) \wedge$,
  $(\forall x, y \in V: (x,v) \in E \wedge (y,v) \in E \Rightarrow x = y) \wedge$
  $(\exists x, y \in V: (v,x) \in E \wedge (v,y) \in E \wedge x \neq y)$.

  In Fig. 4, f_1 is a split filter.
- *Merge filters* are filters having at least two incoming and exactly one outgoing edges. The corresponding predicate `Merge_Filter` is defined as follows:

  `Merge_Filter`(v) : $\Leftrightarrow$ `Filter`$(v) \wedge$
  $(\exists x, y \in V: (x,v) \in E \wedge (y,v) \in E \wedge x \neq y) \wedge$
  $(\forall x, y \in V: (v,x) \in E \wedge (v,y) \in E \Rightarrow x = y)$.

  In Fig. 4, f_2 is a merge filter.

Each filter in a VMgraph has to belong to exactly one of these classes. Thus, filters with two or more incoming *and* two or more outgoing edges are not allowed. We assume that the semantics of such so-called merge/split filters can generally be split into separate merge and split filters.

Note that topological classification does not imply any specific media semantics, yet. For instance, classification as a split filter does not specify, in which way the incoming media stream is distributed to the outgoing streams. Actually, incoming and outgoing streams may be completely independent from each other. While this would be kind of a pathological case, however, we can still imagine

different "reasonable" semantics, e.g., distribution of the incoming media to several outgoing channels (multicast), or decomposition of the incoming media into several parts. Hence, the next section introduces methods to assign more media-specific semantics to the elements of a VMgraph.

5.3. Media-specific semantics

A VMgraph models media streams being manipulated by filters. Each edge of the graph represents one media stream or media object, respectively. A media object can be further characterized by a number of individual *properties*. A set of such properties fulfilling some yet to specify conditions is called a media signature or *signature* for short.

A producer generates a media object. Hence, its semantics is sufficiently determined by the signature of the generated media object. Obviously, consumers can be treated analogously. The semantics of a filter, however, is not adequately specified by the signatures of its incoming and outgoing media streams. In fact, enabling a semantically correct transformation of VMgraphs requires a considerably more precise modeling of the different semantic properties of different kinds of filters.

The following subsections introduce the notions of properties and signatures formally. The formalization of filter semantics, however, can only be sketched. For a full description see [26].

5.3.1. Properties

We consider properties as symbols from a (countable infinite) symbol set M. The individual meanings of the properties do not have to be defined; therefore, new properties can easily be introduced. Furthermore, let a special symbol $\varepsilon \in M$ denote the *empty property*.

While individual semantics of properties is not important for the VirtualMedia model, it must be possible to classify them according to the typical elements of media or filter signatures. This is accomplished by means of predicates, e.g.:

- `Maintype`(p) evaluates to *true*, iff $p \in M$ specifies a media object main type (e.g. *Image*, *Audio*, *Video*, …), otherwise it evaluates to *false*.
- `Subtype`(p) evaluates to *true*, iff $p \in M$ specifies a media object subtype (e.g. *Raster*, *Waveform*, *Multiplestream*, …), otherwise it evaluates to *false*.
- `Encoding`(p) evaluates to *true*, iff $p \in M$ specifies a media object encoding (e.g. *GIF*, *WAV*, *MPEG*, …), otherwise it evaluates to *false*.
- `Operation`(p) evaluates to *true*, iff $p \in M$ specifies a filter operation (e.g. *Clip*, *Sharpen*, *Extract*, *Normalize_Volume*, …), otherwise it evaluates to *false*.
- `Parameter`(p) evaluates to *true*, iff $p \in M$ specifies a (filter-) parameter (e.g. *Threshold*, *Quantization_Matrix*, *Tolerance*, …), otherwise it evaluates to *false*.

Further, we demand that the equality of any two properties is (easily) computable. Then a property distance function $\delta : M \times M \to \{0, 1\}$ can be defined as follows:[5]

$$\forall p, q \in \mathrm{M} : \delta(p, q) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{iff } p = q, \\ 1 & \text{otherwise.} \end{cases}$$

More complex distance functions would be possible (e.g., by exploiting some of the predicates above), but that is not necessary for the model to work. However, a second, less strict variant called δ_c (where c stands for *compatible*) is required:

$$\forall p, q \in \mathrm{M} : \delta_c(p, q) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{iff } p = q \vee p = \varepsilon \vee q = \varepsilon, \\ 1 & \text{otherwise.} \end{cases}$$

Compared to the "normal" distance function, with δ_c properties have a distance of zero (instead of one) to the empty property.

5.3.2. Signatures

A signature is a quadruple (m, s, e, O) with $m, s, e \in M$ and $O \subset M$. These four signature elements are subject to the following constraints:

[5] In [26] properties are defined as a structure and an accordingly refined definition of δ is given.

Definition 1. Let S the set of all signatures, then

$$\forall(m,s,e,O) \in S:$$

$$(m \in M \wedge (m=\varepsilon \vee \texttt{Maintype}(m))) \wedge (s \in M \wedge (s=\varepsilon \vee (\texttt{Subtype}(s) \wedge m\neq\varepsilon))) \wedge (e \in M \wedge (e=\varepsilon \vee (\texttt{Encoding}(e) \wedge s\neq\varepsilon))) \wedge (\forall p \in O: \neg\texttt{Maintype}(p) \wedge \neg\texttt{Subtype}(p) \wedge \neg\texttt{Encoding}(p))$$

That means, m indicates the main type of the media object, where $m = \varepsilon$ is permitted meaning that the main type is unknown or not specified. The second element, s, indicates the subtype. This is, however, permitted only if a main type is given, otherwise $s = \varepsilon$ must apply. The third element, e, indicates the encoding. This is also permitted only if a subtype (and, hence, implicitly a main type) is given, otherwise $e = \varepsilon$ must apply. The set O, which may be empty, can contain arbitrary further properties, however, none of which may indicate a main type, subtype, or encoding (because these three kinds of properties may occur at most once in a signature).

Definition 2. Let $g=(m_g, s_g, e_g, O_g) \in S$, then the *distance function* $\Delta : S \times S \to \mathbf{R}_0^+$ is defined as follows:

$$\forall g,h \in S: \Delta(g,h) \stackrel{\text{def}}{=} c_m\delta(m_g,m_h) + c_s\delta(s_g,s_h) + c_e\delta(e_g,e_h) + c_O\frac{|(O_g \cup O_h)| - |(O_g \cap O_h)|}{|(O_g \cup O_h)| + 1}$$

with firmly selected constants $c_m > c_s > c_e > c_o > 0$.

The last condition guarantees that the main type property gets the highest weight concerning the signature distance, the subtype property the second highest weight etc. One can now easily verify that $\forall g,\ h \in S: g = h \Leftrightarrow \Delta(g,h) = 0 \wedge g \neq h \Leftrightarrow \Delta(g,h) > 0$. Furthermore, if we choose the constants c_i such that $(c_m > c_s + c_e + c_o) \wedge (c_s > c_e + c_o)$ is true, then Δ features the following additional characteristics:

1. $\forall g,\ h \in S$: $m_g \neq m_h \Leftrightarrow \Delta(g,\ h) \geqslant c_m$
2. $\forall g,\ h \in S$: $m_g = m_h \wedge s_g \neq s_h \Leftrightarrow c_m > \Delta(g,\ h) \geqslant c_s$
3. $\forall g,\ h \in S$: $m_g = m_h \wedge s_g = s_h \wedge e_g \neq e_h \Leftrightarrow c_s > \Delta(g,\ h) \geqslant c_e$
4. $\forall g,\ h \in S$: $m_g = m_h \wedge s_g = s_h \wedge e_g = e_h \Leftrightarrow \Delta(g,\ h) < c_o$.

The benefit of these characteristics is that the distance value alone already indicates which of the three type properties are equal or unequal, respectively. Hence, if two signatures g and h according to formula (4) meet the inequality $\Delta(g,h) < c_o$, then g and h are called *type equivalent*, shortly written as $g \cong_T h$.

We further define a function Δ_c analogously to Δ by replacing δ with δ_c. Then, if two signatures g and h meet $\Delta_c(g,h) = 0$, they are called *compatible*, in short $g \cong_c h$, because all properties occurring in both signatures must be equal.

5.4. Assigning signatures to edges and nodes

Let $G = (V, E)$ be a VMgraph and $e = (u, v)$ with $e \in E$ and $u, v \in V$ an edge of G. Then the partial function $\sigma : E \times V \to S$ assigns to all pairs (e, u) and (e, v) a signature $\sigma(e, u)$ or $\sigma(e, v)$, respectively. To illustrate this, think of the start node u as an emitter of a media object with signature $\sigma(e, u)$ and the target node v as an acceptor of a media object with signature $\sigma(e, v)$.

As mentioned earlier, edges represent channels. More precisely, we can now state that $e = (u, v)$ represents a *real channel*, iff $\Delta(\sigma(e, u), \sigma(e, v)) = 0$ applies, thus, a real channel, by definition, is supposed to leave the signature of the transmitted media object untouched. However, a real channel must not necessarily also be instantiatable, because this would require knowledge of the complete type information of the media to transmit. Therefore, an *instantiatable channel* needs the additional constraint $e_{\sigma(e,u)} \neq \varepsilon$ to be met. If, on the other hand, $\Delta(\sigma(e, u), \sigma(e, v)) > 0$ applies, then e is called a *magic channel* assumed to be capable of transforming the media object with signature $\sigma(e, u)$ into a media object with signature $\sigma(e, v)$. In principle, the same effect may be achieved by a VMgraph, i.e., a magic channel can be replaced by a VMgraph without corrupting semantics (and vice-versa). Thus, we may consider a magic channel as a placeholder for a (usually unknown) VMgraph.

We assign semantics to the nodes in a similar way. Two functions $\iota : V \to S^2$ und $\omega : V \to S^2$ assign to each node a set of input and output signatures, respectively. That means: $\iota(v) = \{\sigma(e, v)$

$| \exists\ u \in V\colon e=(u, v)\}$ and $\omega(v)=\{\sigma(e, v) \mid \exists\ u \in V\colon e=(v, u)\}$. A third function $\kappa : V \to M^2$ assigns to each node a set of optional properties, where $\kappa(v) \neq \varnothing \Rightarrow \exists\ m \in \kappa(v)$: `Operation`$(m)$ is assumed.As a consequence, we can also distinguish between nodes representing instantiatable or virtual processing resources (as a general term for producers, consumers, and filters): Let $\Sigma(v)=\iota(v) \cup \omega(v)=\{\sigma(e, v) \mid \exists\ u \in V\colon e=(u, v) \vee e=(v, u)\}$ be the set of all media signatures directly associated to the node v. Then v represents an *instantiatable processing resource*, iff

$$\forall g \in \Sigma(v) : e_g \neq \varepsilon \text{ (implying } m_g \neq \varepsilon \text{ and } s_g \neq \varepsilon).$$

Let $\xi = (\varepsilon, \varepsilon, \varepsilon, \varnothing)$ be the *empty signature*. Then, the following, derived from definitions 1 and 2, applies

$$\forall g \in \Sigma(v) : e_g \neq \varepsilon \Leftrightarrow \forall g \in \Sigma(v) : \Delta(g, \xi) \geqslant c_m + c_s + c_e.$$

Hence, it follows that v represents a *virtual processing resource*, iff

$$\exists g \in \Sigma(v) : \Delta(g, \xi) < c_m + c_s + c_e.$$

Given these measures to distinguish between magic and instantiatable edges and between virtual and instantiatable nodes, we can ultimately define the *virtual VMgraph*:

A VMgraph is called virtual, iff at least one of its nodes is virtual or one of its edges is magic.

5.5. Semantic equivalence relations

We can now reformulate the request processing problem as finding an algorithm to transform a given virtual VMgraph into a semantically equivalent non-virtual VMgraph. In [26] this semantic equivalence is formally defined by means of equivalence relations. For brevity, only one of these relations is presented here to exemplify their formalization. All other relations including those illustrated in Figs. 7, 9 and 10 are defined using a similar approach.

5.5.1. Neutrality

A filter must never behave completely neutral or it would not be of any use. Hence, attributing neutrality to a filter should always be restricted to certain media properties. In [26] three categories of properties are considered: *type* (or format alternatively), *quality,* and *content* properties. All properties that may occur in a signature must be classified as belonging to exactly one of these categories. Following this observation, we have to consider three subtypes of neutrality called *type neutrality*, *quality neutrality*, and *content neutrality*.

5.5.2. Format converters

Filters being both quality and content neutral generally have little if any application-level semantics. This class of filters can be best characterized as all sorts of format converters. Hence, the deployment of these filters should be driven by both freedom and economy. The following formally defined equivalence relation adheres to those principles.

Definition 3. Let $G=(V,E)$ and $G'=(V',E')$ VMgraphs. Then G and G' are semantically equivalent, iff

$$\exists u, v \in V, x \in V', e \in E, e_1, e_2 \in E' :$$

$$\begin{aligned}
&\texttt{content_neutral}(x) \wedge \texttt{quality_neutral}(x) \wedge \\
&x \notin V \wedge V' = V \cup \{x\} \wedge \\
&e=(u,\ v) \wedge e_1=(u,\ x) \wedge e_2=(x,\ v) \wedge \\
&E' = (E \backslash \{e\}) \cup \{e_1, e_2\} \wedge \\
&\quad ((\Delta(\sigma(e, u), \sigma(e, v)) > \Delta(\sigma(e_1, u), \sigma(e_1, x)) \\
&\quad \wedge \Delta(\sigma(e, u), \sigma(e, v)) > \Delta(\sigma(e_2, x), \sigma(e_2, v))) \\
&\quad \vee (\Delta(\sigma(e, u), \sigma(e, v)) \leqslant \Delta(\sigma(e_1, u), \sigma(e_1, x)) \\
&\quad \wedge \Delta(\sigma(e, u), \sigma(e, v)) \leqslant \Delta(\sigma(e_2, x), \sigma(e_2, v)))).
\end{aligned}$$

This definition states that one edge can be replace by two other edges with a neutral filter in-between (cf. Fig. 14). The signatures assigned to these edges have to meet certain constraints. That is, either

(a) the signature difference of the one edge (e) must be greater than the signature differences of both replacing edges (e_1 and e_2) or
(b) the signature difference of the one edge (e) must be less or equal than the signature differences of both replacing edges (e_1 and e_2).

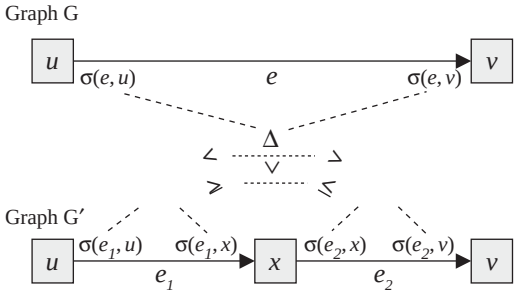

Fig. 14. Graphical illustration of the definition of semantic neutrality.

Thus, we do not allow the signature difference of e_1 being greater than e's together with e_2's signature difference being less than e's and vice-versa, because this would violate the principle of economy.

With regard to the request processing problem we can derive the following two rules from Definition 3: If condition (a) is met, then the filter x should be "built in" (i.e., transition from G to G'). If, on the other hand, condition (b) is met, then the filter x should be removed (i.e., transition from G' to G).

Furthermore, for realizing case (a) it is possible to tighten the condition in the following manner: the signature difference of e_1 must be zero and the signature difference of e_2 must be less than e's. This results in considering only those solutions in which format converters x are built in that match exactly the signature $\sigma(e_1, u)$ of the input media. Thereby, no practical solution would be lost, because, if channel e_1 is not realizable in the first place, then the realization of channel e_2 degrades to a problem without any practical relevance.

As a final remark, note that in practice many format converters are actually not completely quality neutral, often depending on the user's subjective perception of quality. From the perspective of a (VirtualMedia) system designer, there is probably no ideal solution to this problem. One possible approach could be giving (experienced) users some control over the evaluation of the predicate `quality_neutral`.

6. Considerations on request processing algorithms

6.1. General complexity of the problem

All graph transformation rules (except graph composition) can be derived from the equivalence relations (partially) defined in the previous section. Obviously, these rules are applicable to drive the transformation of a VMgraph 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, for example:

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).

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

Generally, the number of transformation rules applicable to any given graph varies without a fixed upper bound. The number of atomic problems to solve, however, is always limited to a maximum of $n = |V| + |E| - 1$. Hence, we may start by selecting atomic problems and applying best matching rules to them one by another according to a breadth-first or depth-first search algorithm, resulting in a search graph with VMgraphs as nodes and rule application as edges. This is also known as *exhaustive search* [27]. Breadth-first search will find a solution to (1), if one exists. Because the search graph can contain cycles, 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. Actually, it can be shown that—because the atomic problems are not solvable independently from each other—the complexity of computing an optimal solution is equivalent to the traveling salesman problem, i. e., O(n!) [26]. Hence, if the aim is finding an optimal solution, we should avoid computing all possible solutions, because in a large multimedia metacomputing system the response time must be kept short. Thus, a heuristic search algorithm is required.

There exist many such algorithms [27] and it is not at all obvious which one to choose. However, because the selection of an algorithm is considered being part of the implementation of the VirtualMedia model (but not part of the model itself), we provide some information on which approaches appear promising.

6.2. Applying heuristics

Good heuristics may considerably reduce the effort required for deciding on both which atomic problem to address and which rule to apply. For the first decision, two different approaches are considered:

- The *a priori* heuristic H_1 determines the "most serious" atomic problem based on the formal definitions given in Sections 5.3 and 5.4. H_1 is defined as follows:

$$H_1(e) \stackrel{\text{def}}{=} \Delta(\sigma(e,u),\sigma(e,v)) \text{ with } e=(u,v)\in E,$$

$$H_1(v) \stackrel{\text{def}}{=} \frac{1}{|\imath(v)\cup\omega(v)|}\sum_{g\in\imath(v)\cup\omega(v)}\left(c_m+c_s+c_e+c_O\frac{|O_g|}{|O_g|+1}-\Delta(g,\xi)\right) \text{ with } v\in V.$$

  Addressing only the atomic problem yielding the highest value of H_1 (costs) by applying one matching rule significantly reduces the complexity of request processing to $O(n)$.
- The *a posteriori* heuristic uses a function H_2 that aggregates the costs of all atomic problems found in a VMgraph. H_2 is defined as follows:

$$H_2(G) \stackrel{\text{def}}{=} \frac{w}{|V|}\sum_{v\in V}H_1(v)+\frac{1-w}{|E|}\sum_{e\in E}H_1(e) \text{ with } G=(V,E), w\in]0,1[.$$

  In this approach, all atomic problems are addressed in turn. The resulting VMgraphs are compared using H_2 and the one with the lowest costs is selected for further processing, thus, yielding a complexity of $O(n^2)$. The rationale behind this is due to H_1 not always determining the rule that reduces the value of H_2 the most.

The heuristic for choosing rules works by assigning a priority to each rule primarily based on how a rule affects the size of a VMgraph. That means, rules that reduce the size of a VMgraph get a higher priority than rules that increase the size.

Applying these heuristics in the most straightforward manner corresponds to the well-known principle of a *greedy algorithm*. In this case, the complexity depends on which of the two heuristics—a priori or a posteriori—is implemented. This algorithm, however, is likely to get stuck in a local optimum, thus missing the global optimum we are looking for. Experiments have shown that the greedy algorithm works quite satisfying, if the request primarily requires the application of format converters [26]. If the request contains descriptively specified content manipulation, however, the greedy algorithm is more likely to fail. There are several traditional algorithms which may 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**, for instance, 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, the cost

functions H_1 and H_2 are not expected to work with these kinds of algorithm.

Instead, we propose to employ randomizing algorithms like *simulated annealing* or *evolutionary algorithms* [26]. Such algorithms may also miss the global optimum, but usually find a better solution than the simple greedy algorithm and, in general, find more often a solution. The computational complexity of both algorithms is basically $O(n)$, but their implementations are considerably more complex compared to the greedy algorithm. In particular, evolutionary algorithms have the advantage of computing several solutions in parallel. This feature may be highly beneficial in multimedia metacomputing, because the heuristic generally can not take volatile conditions like, for example, the availability of all processing resources into consideration. Thus, the optimal solution found by request processing may turn out to be not instantiatable later on, because a required resource is temporarily unavailable, and, hence, the system becomes more robust, if it always has some substitute solutions at hand.

7. Conclusions

In this paper, we described the so-called multimedia metacomputing approach that aims at the formation of a large scale, loosely coupled multiprocessing environment providing a distributed architecture to perform transformations on media objects.

In particular, we point out the importance of data abstractions and formal concepts for multimedia metacomputing systems. Beside common abstractions, such as device and data independence we consider a newly developed abstraction called transformation independence. In principle, this abstraction requires a multimedia system 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 cost of an increased resource demand, for example, storage space for different materializations 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 model 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 (instantiatable) media transformation graphs while applying different heuristic optimization strategies like cost-based evaluation of semantically equivalent graphs or creating and exploiting redundant materialization.

In [19] we already discussed architecture concepts for multimedia metacomputing, yielding the following conclusions:

- Operations that perform the transformations on media objects can be provided in the form of special media processing components.
- Each media processing component should provide a signature to formally describe the runtime environment it requires during its deployment, types of media objects it accepts and the transformations it performs.

- It proves to be essential to exploit services of a repository as a distributed storage mechanism for processing components; in comparison to other solutions, a repository may provide additional functionality related to component versioning as well as combining component versions into valid configurations capable of cooperation in the process of media object transformation.
- An abstract semantic model like the VirtualMedia model has to be provided to ensure (semantically) correct request processing and robustness of client programs against changes of the metacomputing environment like, for example, exchange of components, processors, or media object materializations.

A global multimedia metacomputing environment would, in principle, allow delivering global media data to any client and any kind of multimedia device without the need to generate especially adapted materializations of the media data in advance. Moreover, computationally complex transformations and manipulations of the data are dynamically delegated to the most appropriate processing resources at run-time, thus optimizing response time and utilization of expensive special-purpose hardware. Ultimately, a "pluggable" model for vendors of components providing media transformations and computing resource providers could form the (technical) foundation of a flexible business model for offering and vending multimedia services over the Web.

8. Outlook

There are many perspectives on future research and development in the field of multimedia metacomputing. Obviously, one major direction is towards realizing the concept based on the emerging web and grid services architectures. Subsequently, practical evaluation of the concept and theory certainly will be tackled.

Besides improving and refining the theory for its original application domain—media data processing—it is also worthwhile to think about generalizing the theory, thus making it applicable to other domains in which the automated composition of eServices is an issue. This is a challenging goal, because in our opinion any semantic model serving the purpose of composing eServices must be found using a very careful and thorough analysis of the rationality underlying the present, yet manual composition of such services. Even so, models generally have limitations a human could easily cross, and they sometimes happen to show "unreasonable" behavior (which, of course, humans are also capable of). Considering this, multimedia metacomputing still offers a rich playground for studying and evaluating semantic models without bearing much risk of causing serious damage.

References

[1] L. Smarr, C.E. Catlett, Metacomputing, Comm. ACM 35 (6) (1992) 44–52.
[2] K. Kant, R. Iyer, V. Tewari, A framework for classifying peer-to-peer technologies, in: Proceeding of the Second IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany, May 22–24, 2002, pp. 368–375.
[3] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, San Francisco, CA, 1998.
[4] Google Image Search, Google Website, Google, 2004, http://images.google.com.
[5] S. Hollfelder, F. Schmidt, M. Hemmje, K. Aberer, A. Steinmetz, Transparent integration of continuous media support into a multimedia DBMS, in: Proceeding of the Third Biennial World Conference on Integrated Design and Process Technology (IDPT), Issues and Applications of Database Technology (IADT'98), vol. 2, Berlin, July 6–9, 1998, pp. 192–199.
[6] Informix Digital Media Solutions: The Emerging Industry Standard for Information Management, Informix Software, Inc., 1997.
[7] Oracle8i interMedia Audio, Image, and Video User's Guide and Reference, Oracle Corporation, 2000.
[8] Napster, Napster Website, Napster, Inc., 2004, http://www.napster.com.
[9] Gnutella, Gnutella Website, Gnutella.com, 2004, http://www.gnutella.com.
[10] K.A. Hawick, H.A. James, A.J. Silis, D.A. Grove, C.J. Patten, J.A. Mathew, P.D. Coddington, K.E. Kerry, J.F. Hercus, F.A. Vaughan, DISCworld: an environment for service-based metacomputing, Future Gener. Comput. Sys. 15 (5–6) (1999) 623–635.

[11] G. Alonso, Myths around Web Services, IEEE Data Eng. Bull. 25 (4) (2002) 3–9.
[12] S. Tsur, Are web services the next revolution in E-commerce?, in: Proceeding of the 27th International Conference on Very Large Data Bases, VLDB 2001, Roma, Italy, September 11–14, 2001, pp. 614–617.
[13] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, M. Lebofsky, SETI@home—Massively Distributed Computing for SETI, Comput. Sc. Eng. 1 (2001) 78–83.
[14] G.D. Speegle, X. Wang, L. Gruenwald, A meta-structure for supporting multimedia editing in object-oriented databases, in: Proceedings of the Advances in Databases, 16th British National Conference on Databases, BNCOD 16, Cardiff, Wales, UK, July 6–8, 1998, pp. 89–102.
[15] M. Wagner, S. Holland, W. Kießling, Towards self-tuning multimedia delivery for advanced internet services, in: Proceedings of the First International Workshop on Multimedia Intelligent Storage and Retrieval Management (MISRM '99) in conjunction with ACM Multimedia Conference Orlando, FL, October, 1999.
[16] T.C. Rakow, W. Klas, E.J. Neuhold, Abstractions for multimedia database systems, in: Proceedings of the Second International Workshop on Multimedia Information Systems, West Point, New York, USA, September 26–28, 1996, pp. 41–46.
[17] P. Seshadri, Enhanced abstract data types in object-relational databases, The VLDB J 7 (3) (1998) 130–140.
[18] U. Marder, On Realizing Transformation Independence in Open, Distributed Multimedia Information Systems, in: A. Heuer, F. Leymann, D. Priebe (Eds.), Proceedings of the Ninth GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", BTW '2001, Oldenburg, March 7–9, Springer, Heidelberg, Berlin, 2001, pp. 424–433.
[19] U. Marder, J. Kovse, Multimedia metacomputing, in: Proceedings of the Seventh International Workshop on Multimedia Information Systems, MIS 2001, Capri, Italy, November 7–9, 2001, pp. 173–182 (also in: ACM SIGMOD Digital Symposium Collection 2002).
[20] T. Prückler, M. Schrefl, An Architecture of a Hypermedia DBMS Supporting Physical Data Independence, in: Proceedings of the Ninth ERCIM Database Research Group Workshop on Multimedia Database Systems, Darmstadt, March 18–19, 1996, pp. 45–57.
[21] R. Käckenhoff, D. Merten, K. Meyer-Wegener, MOSS as multimedia object server, extended summary, in: R. Steinmetz (Ed.), Proceedings of the Second Internantional Workshop on Advanced Teleservices and High Speed Communication Architectures, IWACA '94, Springer, Heidelberg, Berlin, September 1994, pp. 413–425.
[22] K.S. Candan, V.S. Subrahmanian, P. Venkat Rangan, Towards a theory of collaborative multimedia, in: Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan, June 1996, pp. 279–282.
[23] D. Dingeldein, Multimedia interactions and how they can be realized, in: Proceedings of the SPIE/IS&T International Conference on Multimedia Computing and Networking, San Jose, CA, February 5–10, February 1995, pp. 46–53.
[24] DirectShow Documentation, MSDN Library, Microsoft Corporation, 2002, http://msdn.microsoft.com/library/.
[25] Java Media Framework API Guide, Sun Microsystems, Inc, 1999.
[26] U. Marder, Multimedia-Metacomputing in Web-basierten multimedialen Informationssystemen, Logos Verlag, Berlin, 2003 (in German).
[27] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer, Berlin, Heidelberg, ISBN 3-540-66061-5, 2000.