On Realizing Transformation Independence in Open, Distributed Multimedia Information Systems

Ulrich Marder

University of Kaiserslautern
Dept. of Computer Science
P. O. Box 3049
D-67653 Kaiserslautern
Germany
marder@informatik.uni-kl.de

Abstract. In this article, we present our efforts[1] on realizing transformation independence in open, extensible, and highly distributed multimedia information systems. The main focus is on the abstract data and processing model called VirtualMedia which provides to application developers a kind of 'metaprogramming' interface 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. The article concludes with a brief outlook on future research objectives.

1 Introduction

About a decade ago, at the eve of the internet's success story, the concept of metacomputing was invented [15]. The basic idea behind metacomputing is that distributed computing resources interconnected by high-bandwidth local or wide area networks could be integrated, thus appearing as one large high-performance (meta-) computer. The first metacomputers were built more or less ad-hoc and/or were designed for special applications of high-performance computing, e. g. numerical simulations for weather prediction or cosmological research. From these beginnings, more general-purpose metacomputing environments have emerged [3], which are already applied for building geographical information systems (GIS) and, hence, are most probably also beneficial for the construction of open, distributed multimedia information systems (MMIS).

The general architecture of metacomputing environments is peer-to-peer. That means, in principle each node can be both client and server, which leads to a highly scalable system architecture enabling global information systems with possibly thou-

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

sands or even millions of peers. Obviously, such an architecture also raises questions, e. g., concerning availability, robustness, or security. We will, however, not discuss architectural issues of peer-to-peer systems in this article. Instead, we turn to the problem of programming (peer-to-peer) metacomputers efficiently, especially for multimedia processing in MMIS. The key question here is: how can we exploit the potential power of a metacomputer without exactly knowing *which* computing resources will be available at run-time and *where* they will be available? The most general answer to this question is that metacomputers require a kind of 'metaprogramming'. In this article, we will look at how this metaprogramming could be realized for multimedia processing—an area in which most programmers are still used to "thinking in bits and bytes".

In [3] a service-based approach to metacomputing is pursued, where the notion of a service is borrowed from middleware standards like CORBA. We believe that this quite reasonable approach is still too low-level for the application developers, since they have to be concerned with tasks like locating services matching their application semantics, organizing the (media) data flow between services, deciding on remote materialization of media objects, and so on. Instead, all this inconvenience should be managed, controlled, and optimized by a (high-level) media object service, leaving only the provision of sheer application semantics to the (meta-) programmer. This principle has been formerly introduced as transformation independence [9].

The rest of the article is organized as follows. Section 2 briefly summarizes the transformation independence abstraction. Then, section 3 will demonstrate how transformation requests are represented in our VirtualMedia approach. Section 4 presents an outline of the VirtualMedia model and section 5 concludes this article with some prospects on future work and research issues.

2 Transformation Independence

The notion of transformation independence was actually an outcome of our former work on realizing so-called *Media-specific Abstract Data Types* (MADT) in a distributed computing environment [7, 12]. In what follows, we give a brief summary of our introduction to the transformation independence abstraction presented in [9] (detailed descriptions in English are provided in [10, 11]).

The MADT concept's main objective was introducing 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. 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). However, different multimedia applications prefer different physical media formats. Therefore, we have to solve the problem of mapping media semantics efficiently onto varying physical representations of the media, which is roughly what we now call transformation independence.

More specifically, transformation independence can be 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). The key concepts of transformation independence are graph-based media transformations, transformation request resolution, redundant materialization, and distributed filter instantiation [11]. In the following sections, we will describe how the first three are realized in the VirtualMedia project.

There is currently no system known to be truly realizing transformation independence. We will, however, shortly mention some important approaches to solve (at least some of) the problems that we believe transformation independence will solve.

The concept of *Enhanced Abstract Data Types* (E-ADT) [16], which has been realized in the ORDBMS prototype *Predator*, provides a solution regarding the optimization of composite operations (transformation requests in our terminology). In particular, the E-ADT approach requires such transformation requests being specified (or interpreted, respectively) in a descriptive manner, thus enabling semantic optimization (e. g., permutation or replacement of operations). The salient feature of the E-ADT approach—its tight and elegant integration with traditional relational database technology—, however, also seems to prevent considering more advanced optimization strategies like cross-media optimization or optimization-driven materialization. Further, neither format independence nor the irreversibility problem are addressed.

Within the AMOS project at GMD IPSI a concept called *presentation independence* has been developed [14]. This abstraction aims at optimizing pre-orchestrated presentations for differently equipped clients. The QoS of such a presentation automatically adapts to the client's facilities at run-time. There are, however, no operations for ad-hoc creation or modification of presentations. Thus, the physical data independence provided is kind of 'static' (besides being dedicated to presentation only).

Commercial ORDBMSs (available, e. g., from Informix, IBM, and Oracle) are extensible by defining and implementing *User-defined Types* (UDT) and *User-defined Routines* (UDR). This mechanism is also extensively used to enhance those systems with media data types (for some examples see [5], [6]). 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 [4] a continuous media DataBlade providing device independence, location transparency, and presentation independence is described, and (2) [17] presents a DB2 Extender for images providing format independence where materialization is controlled through cost-based optimization.

3 Transformation Requests in VirtualMedia

The VirtualMedia project is targeted at realizing transformation independence in a distributed, heterogeneous MMIS (e. g. Web-based) in the first place. Generally, VirtualMedia is conceived as a framework providing an environment for distributed processing and persistent storage of media objects of any type. Any media processing serv-

ice or storage server conforming to the VirtualMedia (metaprogramming) model could be plugged into this framework. In particular, VirtualMedia addresses API, data model(s), architecture, DBMS-integration, optimization, protocol, visualization, and interoperability issues. However, the following introduction mainly focuses on API, data model, and optimization concepts.

3.1 VirtualMedia Descriptor

In VirtualMedia, transformation requests are sent to a server to access so-called virtual media objects (VMO). According to the transformation independence abstraction, 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. This so-called *VirtualMedia Markup Language* (VMML) [11] is based on XML.

In VMML a transformation request is called a *VirtualMedia Descriptor* (VMD). A VMD contains descriptions of source-MOs and of target-MOs (also called client-VMOs). If the source-MOs are DBMS-managed VMOs, specifying a reference (external database ID) is usually sufficient. This reference, however, may be accompanied by a media signature to enforce certain type, quality, or content properties of the MO. The description of a client-VMO contains two mandatory parts, the media signature and a transformation specification. If the client-VMO is to be materialized at the client its signature must provide at least complete type information. The transformation specification must at least reference one of the source-MOs from which the client-VMO should be derived. Additionally, operations on the MO(s) can be specified. All required input-MOs of these operations must be bound either to one of the source-MOs or to an output-MO of another operation or transformation. If there are multiple operations on the same MO, it is assumed that all these operations can be performed in any sequence order.

Effectively, a (successfully verified) VMD describes a directed acyclic graph (DAG). Source MOs become start nodes, operations become intermediate nodes, and client-VMOs usually become end nodes of this graph. The edges are derived from the (explicit or implicit) binding of source- or output-MOs to input-MOs or client-VMOs. Thus, this graph structure is a suitable internal model for describing any media transformation requested through VMDs.

3.2 Filter Graphs

Modeling and realizing the processing (i. e. transformation) of media objects through filter graphs is a probably well-known principle (see, e. g., [1] and [2]). However, to our knowledge it has never been applied to build an abstract media transformation concept.

Like our transformation requests, filter graphs are also DAGs. The start nodes of the graph are media sources (media objects stored in the database or anywhere else, maybe even live media sources) and the end nodes are media sinks (most often client

applications or the database). The intermediate nodes are media filters, the basic operations forming a media transformation, while the edges of the graph represent media streams flowing from one filter (or media source) to another filter (or media sink).

It is easy to define an isomorphism between the graph representation of VMDs and filter graphs. This, however, would not correctly reflect the semantic relationship between the both. 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*). By 'instantiatable' we mean that each media source is a materialization, 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[2]. Consequently, VirtualMedia's main optimization problem is finding the cost-optimal instantiatable filter graph for a given VMD (if one exists).

To support the transformation of request graphs into instantiatable filter graphs an integrated filter graph model is introduced. Such a VirtualMedia filter graph may contain both virtual elements and real (or instantiatable) elements: materializations (MO_m), VMOs with external ids (visible database objects), client-VMOs (specified through a transformation request), and filters (virtual and instantiatable).

4 The VirtualMedia Model

VirtualMedia provides a model for automatically transforming a virtual filter graph into an instantiatable filter graph (also called a transformation prescript graph). As demonstrated in [11], we have identified the following rule classes to be considered in this model:

Implementation selection. Rules that find filters implementing the semantics of a given virtual filter.

Type/format adaptation. Rules which resolve type or other signature mismatches between subsequent (virtual or instantiatable) filters.

Semantic optimization. Rules that exploit knowledge of semantic relationships between filters (e. g., reversibility or permutability relations).

Materialization selection/rejection. Rules that exclude materializations from being used as source objects. Additionally, rules are needed for selecting materializations that should be added to (or removed from) materialization graphs.

Each of these rules either adds, removes, or replaces nodes of a VirtualMedia graph. Hence, in order to assure that applying a rule always preserves the semantics of the transformation request, an appropriate formal model describing the VirtualMedia semantics has been specified. However, due to space limitations we only sketch this specification in the following subsections.

---

[2] The ∞ is due to the possibility of periodic repetition of semantically neutral subgraphs. Of course, this case is kind of pathologic and usually avoidable in practice.

4.1 Data Model for Media Objects and Filters

An object-oriented data model describing media object types and media filter types is defined. 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. 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 being 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* 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.

4.2 Semantic Equivalence Relations

All graph transformation rules are derived from a number of equivalence relations concerning (sets of) filters and MO-signatures. Most of these equivalence relations are explicitly modeled as relations within the object-oriented schema, while some may also be expressed in equational form.

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 will probably not consider each possible media property 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.

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 fil-

ters 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, e. g. quality neutrality (strong) and content neutrality (weak).

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 instantiation time.

(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.

The semantic reversibility of (de)compose-filters can be exploited for applying filters to single streams of a multiple-stream MO. Thus, the definition of reversibility is generalized in a sense that all other filters (i. e., not only neutral filters) are allowed inbetween a decompose/compose pair which is newly inserted into a VirtualMedia graph. In the case of a multiple-filter node with a multiple-stream input the filters in this node may be applicable to different streams of the multiple-stream MO (depending on their signature). Since the filters are classified as permutable there are no semantic dependencies between them. Hence, it is possible to split the multiple-filter node when it gets embraced by a decompose/compose pair.

Semantic Assimilation The semantic equivalence between a virtual filter and a possible implementation of this filter is called semantic assimilation. The implementation of a virtual filter X_V consists of an instantiatable filter X_I implementing the semantics of X_V and an arbitrary number of additional filters. The additional filters may be located before and after X_I. They must either be semantically neutral or, otherwise, a filter Y *before* X_I must be followed by its inverse Y^{-1} *after* X_I where (Y, Y^{-1}) conform to the generalized reversibility semantics. An implementation is called *complete* if (1) all filters are instantiatable, and (2) the signature distance between start and end point of all edges is zero.

4.3 Considerations on Graph Transformation Algorithms

All graph transformation rules 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).

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 rules as edges. Breadth-first search will find a solution to (1) if one exists. If no solution exists, breadth-first search might not terminate, because infinite branches may exist in the search graph. This infinite search space is due to our rules allowing unlimited growth of VirtualMedia graphs in principle. Thus, depth-first search might not even terminate when a solution exists.

It is, however, possible to define a cost function based on signature distance which behaves always monotonic on the path from the request graph to the solution graph. Then an A^*-like heuristic search algorithm could be applied to find a solution to (1) quite efficiently. It is not clear, yet, whether the monotony criterion can always be met if we try to solve (2) or (3) this way. This will have to be investigated in future work.

Finally, notice that a divide-and-conquer approach (dynamic programming) is not applicable 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.

4.4 VirtualMedia in Metacomputing Environments

In metacomputing environments like DISCWorld [3] instantiatable media filters could be realized (e. g., wrapped) as services which may be provided by any (or at least some) peer. Supporting VirtualMedia's request resolution concept additionally requires a query mechanism for finding media filter services that match the semantics of a given virtual filter. Finally, it must be possible to link the media filters by establishing (high-bandwidth) data channels between them. Unlike other metacomputing environments, DISCWorld addresses these both requirements, too. Thus, VirtualMedia could provide a high-level API for distributed multimedia computing being transparently mapped to such an environment. This mapping process includes some additional opportunities for optimization to be considered in the future.

5 Conclusions

In this paper, we propose VirtualMedia as an approach to realize transformation independence in open (e. g., web-based) MMIS. VirtualMedia solves the irreversibility problem[3] [9] 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) 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. Of particular interest are the following aspects, to name a few:

- Enhancing the data model with, e. g., hierarchical structures (explicit subgraphs) or a template concept (parameterized client-VMOs).
- Development of a reference architecture for a VirtualMedia service based on ORDBMS technology, metacomputing environments and VirtualMedia compatible media servers like Memo.real [8].
- Integrating adaptive QoS (feedback-controlled) and interaction (including interactive filters).
- Several API issues, e. g. improvement of the XML-based VirtualMedia markup language and consideration of non-functional properties as part of signatures.

References

1. 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.
2. Dingeldein, D.: Multimedia interactions and how they can be realized. In: Proc. Int. Conf. on Multimedia Computing and Networking, 1995.
3. Hawick, K., A., James, H., A., Silis, A., J. et al.: DISCWorld: An Environment for Service-Based Metacomputing. In: Future Generation Computer Systems, 15 (5–6), 1999, pp. 623–635.
4. 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.
5. Informix Digital Media Solutions: The Emerging Industry Standard for Information Management. Informix White Paper, Informix Software, Inc., 1997.
6. Informix Video Foundation DataBlade Module. User's Guide Version 1.1. Informix Press, June 1997.
7. Käckenhoff, R., Merten, D., Meyer-Wegener, K.: MOSS as Multimedia Object Server – Extended Summary. In: Steinmetz, R., (ed.): Multimedia: Advanced Teleservices and High

[3] Making irreversible media transformations persistent without sacrificing application neutrality.

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.
8. Lindner, W., Berthold, H., Binkowski, F., Heuer, A., Meyer-Wegener, K.: Enabling hypermedia videos in multimedia database systems coupled with realtime media servers. In: Proc. IDEAS 2000 (Yokohama, Japan, 18.-20. Sept.), Sept. 2000.
9. Marder, U.: Medienspezifische Datentypen für objekt-relationale DBMS: Abstraktionen und Konzepte. In: Proc. 8. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" BTW '99 (Freiburg, March 1–3, 1999), Ed. A. P. Buchmann, Springer-Verlag, Berlin 1999, pp. 210-231.
10. Marder, U.: Towards a Universal Media Server. SFB-Report 03/2000, SFB 501, University of Kaiserslautern, Feb. 2000, 19 pages.
11. Marder, U.: Transformation Independence in Multimedia Database Systems. SFB-Report 11/2000, SFB 501 University of Kaiserslautern, Nov. 2000, 24 pages.
12. 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.
13. 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.
14. 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.
15. Smarr, L., Catlett, C. E.: Metacomputing. In: Comm. ACM, Vol. 35 No. 6, June 1992, pp. 44–52.
16. 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.
17. 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.