Proseminar Multimedia-Databases & Retrieval: An introduction to METIS - a Flexible Database Foundation for Unified Media Management

Sönke Greve

Institute For Web Science And Technologies
University of Koblenz-Landau

Abstract. This paper introduces the idea, architecture and purpose of METIS which is a flexible and highly costumizable multimedia database. It is the result of pushing a truly multimedia approach in the landscape of mostly single-media oriented databases. METIS can be depicted as a comprehensive system to all kinds of media, their metadata attributes, internal connections and requirements in feature-generation and comparison. What defines METIS is the elaborate and clean structure it offers for dealing with any given type of media. It is not a system that implements new storage or user interaction approaches as they can set up individually. The given structure can easily be adjusted to meet domain specific needs without loosing its ability to extend to new non-domain specific media types.

1 Introduction

Most of the so called "multimedia databases" that currently exist focus on the management of a single media type, such as videos [2, 4, 5] , images [3, 8, 7] or audio. Each of those systems deals with an own heterogeneous model when describing or indexing its contents. Combining several of those systems in one environment doesn't lead to a fully integrated and unified media management system. It rather creates parallel isolated silos of media under one hood. Querying such a system or comparing data across the silos' borders must be implemented uniquely for each of the silos to appear as single system. METIS also requires specific implementations for specific media types but all is dealt with in a unified manner in a single and fully integrated system. This results in three main benefits:

1. querying the database is unified for each media type
2. modelling is purged when dealing with multiple media types due to a media independent architecture.
3. new media types can be integrated at relatively low efforts

These benefits are provided through the design of the METIS system core which is created regarding four main concepts. First there is the abstract *data model* embracing all types of media content. Second are the *CMOs (Complex Media Objects)* which are template-based media-containers for the use of composed types like websites or multimedia document formats such as MHEG, SMIL, SVG [6] or even upcoming standards. Third is a highly customisable *functions and operators* repository that is usually application specific. Therefore it can be extended through loadable Java classes. And finally there is the *query processor* enabling a search for semantic classification of media, high-level characteristics, low-level features and the relationship to other media objects.

METIS carries the highly generic and customisable concept to all levels of its architecture. There is a *visualisation layer* based on eXtensible Server Pages (XSP) and the Apache Cocoon framework. It is extended to a rendering pipeline via XHTML-templates and XSLT stylesheets in order to map non-static core functionality to the user interface. At the other end of the architecture there is the *persistence abstraction layer* representing the storage back-end of the system. It is adaptable towards different storage techniques that are provided through classes implementing a database manager interface. These classes can be used to define your own storage solutions or use commonly supported concepts like SQL92 compliant relational databases as well as file system based methods using XML files. In order to reduce the administrative complexity of such a highly customisable system METIS introduced the idea of *semantic packs*. *Semantic packs* can be interpreted as saved configurations of an application specific solution concerning all level of METIS' architecture.

In the following the METIS database architecture is described in detail. Section 2.1 enlightens the core's features. The system's front- and back-end solutions are depicted in 2.2 about the *persistence abstraction layer* and 2.3 about the *visualisation layer*. Section 2.4 deals with the concept of the *semantic packs*. Section 3 and 4 display a brief view on open issues as well as a final statement about the system.

2 The METIS architecture

This section introduces the architectural elements of the METIS multimedia database depicted in figure 1. Paragraph 2.1 deals with core components of METIS. These are the *data model*, *complex media objects*, a *functions and operators* repository and the *query processor*. In 2.2 the storage back-end called *persistence abstraction layer* is presented and 2.3 explains the idea of user-interaction via the *visualisation layer*. Finally paragraph 2.4 shows the use of semantic packs.

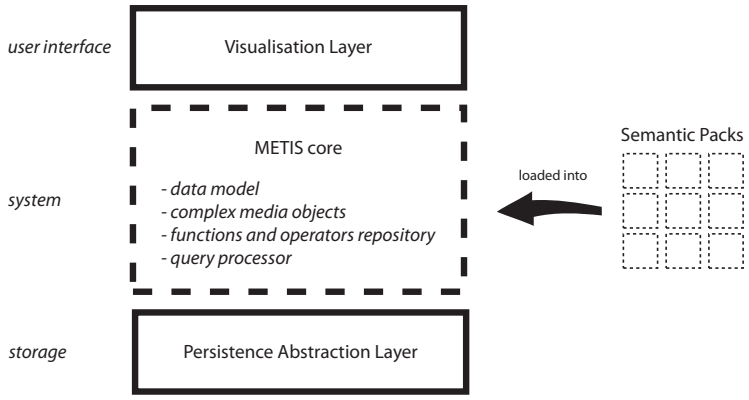

Fig. 1. METIS architecture

2.1 The METIS core

This section provides an insight on the core of the METIS system emphasising the flexibility and customisability of the framework. Regarding flexibility the core is implemented as a Java servlet and distributed as a web archive (WAR-file). This enables platform- as well as environment-independence of the system. Each web-server capable of treating Java servlets can be used as a distribution environment. The founders at the Research Studios Austria[1] started on top of the Apache Tomcat project.

[1] http://www.researchstudio.at

Data Model The internal representation and management of media as shown in figure 2 has to meet two major requirements.

1. deal with any kind of media
2. adapt any kind of application demands

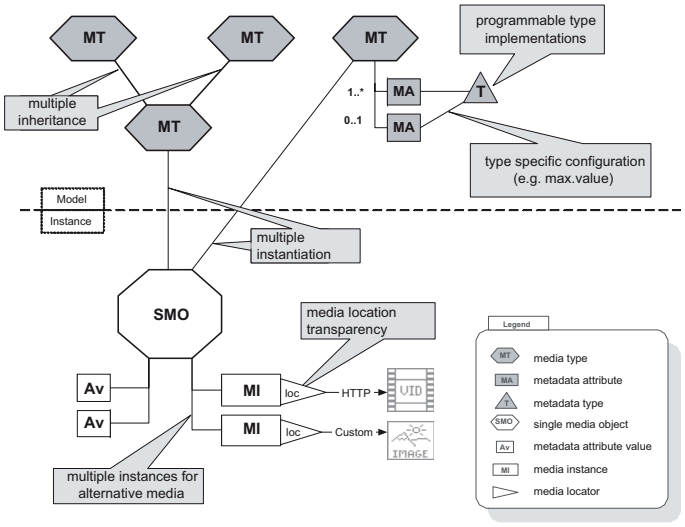

Fig. 2. The METIS core data model

Therefore media within METIS is represented by an abstract type called *single media object (SMO)* which basically is the ID of a database entry. Any *SMO* is connected to one or many *media instances* that carry attributes such as size, encoding format, bit rate (etc.). Each *media instance* is connected to a single *media locator*. They address the actual data stored in a file system, a web server or a database. This structure provides the ability to gather different physical manifestations of media under one logical representation. It can be used for load balancing, delivering the data with the best fitting resolution or considering bandwidth qualities when deploying an instance.

The semantic classification in METIS is done via the assignment of an arbitrary number of *media types* to each *SMO*. Those *media types* are collections of metadata attributes and can inherit from each other. The used concept of inheritance contradicts the idea of multiple assignments. This change of style is tolerated due to the advantage of assigning metadata attributes of parallel taxonomies to a single object. The authors of METIS [6] chose this kind of realisation in order to avoid artificial, hybrid media types. Metadata attributes are of different *metadata types* which can be freely defined as Java classes. The connection between a *metadata attribute* and a *media type* can be configured

with a specific cardinality stating the requirement or limitation of a number of *metadata attributes* within a single *media type*.

Another characteristic for classifying media is not only attribute- but connection-based. Therefore METIS supports *association types* which are freely definable Java classes. Those *association types* coexist with the *metadata attributes* within a *media type*. Connected *SMO*s must implement the same *media type* containing the specific *association type*. The cardinality is supported by this concept, too.

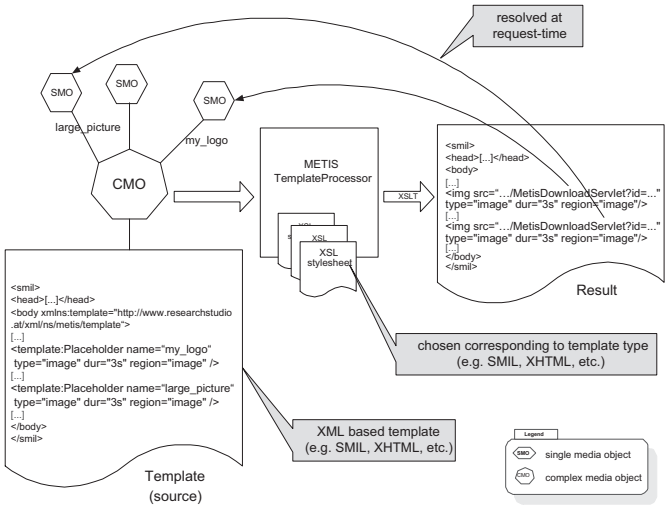

Fig. 3. Complex Media Objects

Complex Media Objects An extension of the *data model* is made by means of *CMOs (complex media objects)*. *CMO's* are very similar to *SMOs* as they instantiate *media types*. They are assigned attribute values and may participate in associations. But their main use is to serve as containers for *SMOs* what makes them a representation of multimedia documents within METIS. Following again the concept of customisability and flexibility it was avoided to declare another multimedia document format but to support existing or even upcoming ones with a simple template mechanism. An example for this mechanism can be found in figure 3.

Templates are first-class objects represented by XML documents and may be shared between multiple *CMO's*. Within those templates placeholders mark the media content used within. Requesting a *CMO* at runtime the placeholders are replaced by format-compliant references via an XSLT stylesheet representing the used *SMO's*. This way it first doesn't really matter if one wants to create a *based* on MHEG, SMIL, SVG or some new document type alike. And second

this technique reflects real world media productions very well where tasks of design and content management are separated. The limitations of this template mechanism are the unknown definitions within the template. METIS is unaware of temporal or spatial resolution of the used *SMO's* and can't therefore be queried against them.

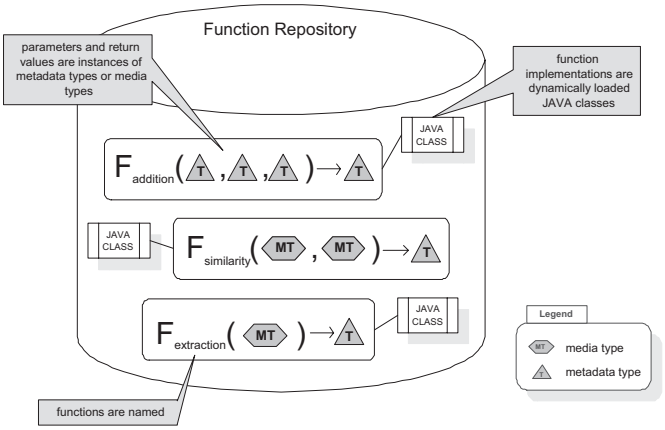

Fig. 4. The METIS functions and operators repository

Functions and Operators The functionality and handling of data in METIS is provided by a *functions and operators* repository depicted in figure 4. As this part of multimedia management, comparison, sorting and ordering is usually explicitly connected to a specific application domain it doesn't suffice to offer a static set of predefined functions (e.g. a philatelic database uses different similarity measures for its stamps than a passport photo database providing face recognition support). Therefore an extendable framework was integrated into the METIS core. Functions are implemented by loadable Java classes. These *function implementations* have full access to all resources within the database including the *data model*. For example the input parameter of a function could be a *SMO* classified by the assignment of a *media type* as a result. In other words functions can process any type of *SMO*, *CMO* or instances of *metadata types* resulting in a *media object* or *metadata type* instance.

Query Processor The query processor provided in METIS is capable of hybrid search for *SMO's* and *CMO's* through a simple query language. The media's semantic classification, high-level characteristics, low-level features and relationship to other media can be taken into account. The selection is done by testing all *media objects* against a given condition. A condition consists of nested conjunctions and disjunctions of selected predicates. So objects can be requested

belonging to a kind (e.g. single or complex), a *media type* they instantiate (directly or through inheritance), the values of their *metadata attributes* (compared by functions of the extended repository) or their participation in associations.

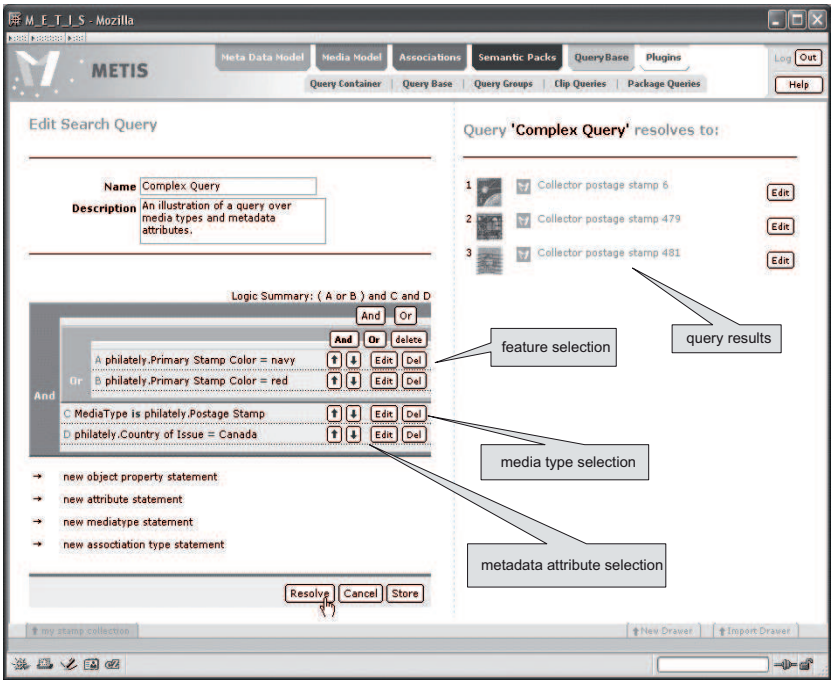

Fig. 5. The METIS web front-end with a query example

Figure 5 depicts the selection of predicates. The left displays the integrated graphical tool for the formulation of requests. The right shows a result set of *SMO's*. The expressiveness is still limited. For example METIS doesn't support joins between media objects at its current state. Most of the limitations can be by-passed by hard-coding the functionality in the *functions and operators* repository. This is nothing else than a workaround at first hand but an indication for the depth of flexibility of METIS on the second. Queries are persistent objects and can be stored for repetitive use.

2.2 Persistence Abstraction Layer

Though science is usually based on theoretical and/or idealised models of reality it sometimes has to deal with less objective problems - typically the *human factor*. The creators of METIS had to face this subject when negotiating with potential commercial project partners. Since the METIS structure is fully object

oriented the first prototypes were implemented in a object database system. That fact found few acceptance on the commercial side. The bias on this topic is chiefly based on the strong confidence in relational database technology in this sector than on technical issues. Another argument is that relational databases are often already present within enterprises. Therefore the people are already experienced on this kind of software and there is no or only few need for new licenses. To meet the resulting requirements and with regard to the flexibility paradigm the storage interface was re-implemented including a *persistence abstraction layer*.

All persistent objects in METIS are now derived from a *persistent object base class*. This class demands store, update and delete functions for each of its subclasses. Those functions do not directly access the storage layer, but rather an abstract *database manager interface*. This interface handles transactions, commits and rollbacks. This architecture enables a custom support for different storage back-ends. Every kind of storage must be implemented in a *database manager class*. Implementations for a serialised XML-based file system storage or SQL92 compliant database systems already exist. They can easily be adapted to other relational database systems ranging from low-cost systems to large-scale back-ends suitable for commercial use.

Furthermore the *persistence abstraction layer* provides a framework for the integration of new media locators. They are implemented via Java classes themselves implementing interfaces for the different kinds of storage locations (e.g. supporting read-only, read-write or random access).

2.3 Visualisation Layer

METIS is published with an own web-based administration front-end. It is based on a configurable *rendering pipeline* using a series of XSLT transformations. This provides flexibility in the *visualisation layer*. The need for flexibility in this place can have different reasons. One is of course the graphical representation like matching certain styleguides or corporate identity specifications. Another reason can be the user-dependant interface behaviour or multilingual adaptations of the system's surface.

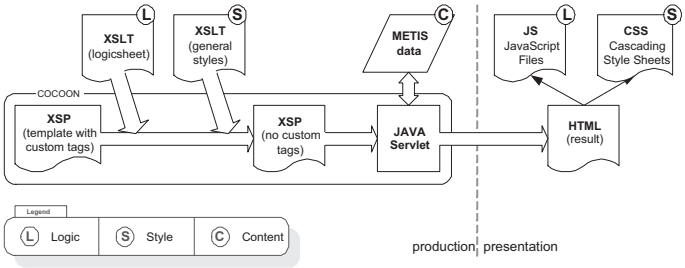

Fig. 6. The METIS rendering pipeline / template processor

In order to generate elaborate graphical design, compatible with different browsers it is necessary to create highly complex HTML code. To avoid limitations to these requirements a complex *rendering pipeline* was created. It is based on eXtensible Server Pages (similar to Java Server Pages) that are processed by the Apache Cocoon Framework. Starting with the XSPs (without custom tags) in figure 6 the right-hand side equals the process of building general XSP web applications. The created Java Servlet queries the METIS database and generates the final HTML output as well as required documents like JavaScript- and CSS-Files. To enable non-static solutions the left-hand side describes the way the XSP-document is generated. Web-developers are given the possibility to create templates including custom tags and additional replacement directives stored in XSLT logic- and stylesheets. This way creating a domain-specific GUI using the pipeline becomes a bit more complex than just offering a set of predefined METIS components for the use within XSP applications. But as it is hard to predict all required functionality the METIS components would have to offer the chosen pipeline architecture offers the highest level of creative freedom.

2.4 Semantic Packs

Keeping everything flexible and customisable creates two major problems. The first one is the complexity of creating a domain specific solution from the scratch. Each layer or part of the system core must be customised to meet the domain specific requirements. The second is the lack of defined semantics in declared types, attributes or objects. This is similar to XML documents wherein tags with the same name yet completely different meanings may be found in a single document.

To face these difficulties so called *semantic packs* were introduced at a certain stage of the METIS development. They establish spaces of semantically related METIS objects, analogous to XML namespaces. All object names in METIS are prefixed by the URI of the semantic pack they belong to. A graphical representation of a namespace prefix is indicated in the lower box of figure 7. Semantic packs also include definitions of METIS objects, able to be equipped with *metadata attributes*, functions, *media types*, associations and dynamically loaded Java classes. Those classes can contain simple *data type* implementations, *media locators*, *feature extractors* and even templates for the web front-end. In other words *semantic packs* enable pre-packaged, domain specific solutions for METIS that can easily be installed and reduce the administrative complexity. Those configuration-files of METIS are hierarchically organised and can make use of other *semantic packs*. For example most *semantic packs* will make use of the METIS base semantic pack, which contains primitive data types, such as string, integer or date, required by almost any application. The packs come as Java archives (JAR) that enable the authentication of their origins and the prevention of malicious code.

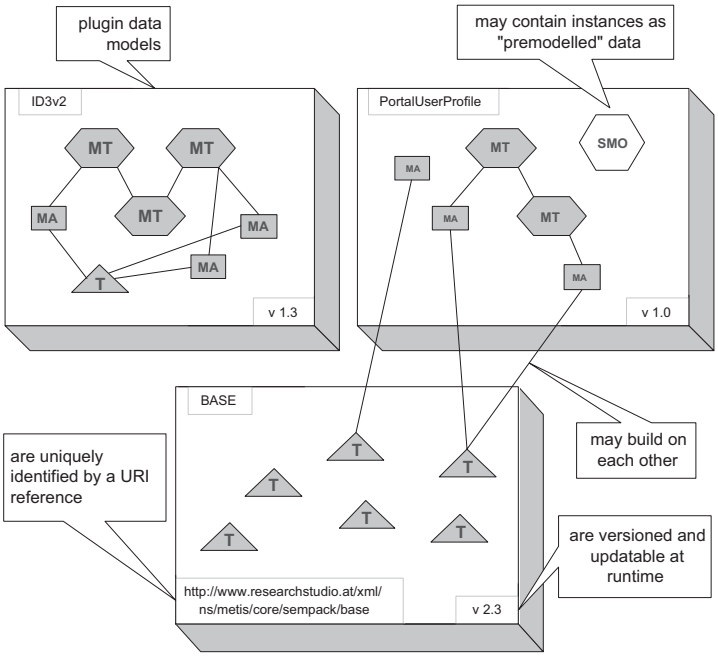

Fig. 7. Semantic Packs in METIS

3 Future Work

The authors of METIS point out 5 essential construction sites of their system:

Indexing
The indexing capabilities of METIS are currently very low. The integration of an indexing framework supporting arbitrary unordered, ordered and multidimensional index structures is under development. The goal is a more efficient querying of media.

Query Language
The current query language integrated in METIS is limited as depicted in Section 2.1 (Query Processor). It is strived towards relevant multimedia querying standards, such as SQL/MM [1] .

Version Control
In a multimedia production cycle it is necessary to keep track of different versions of the same media and its metadata. Versioning support for media objects is to be implemented in future releases.

User Management
There doesn't exist any security or user management support in METIS. This is essential for a commercial use of databases where multiple users, teams or companies access media material on a single server.

Caching & Networking
By optimising the *persistence abstraction layer* it is planned to improve caching techniques and investigated how distributed instances of METIS can work together on the basis of a peer-to-peer network (multi-server support).

4 Conclusion

METIS is pretty much a one-of-a-kind solution. There are few other databases following this concept of true unified multimedia management [6] . In terms of flexibility and customisability METIS is a definite advancement to them.

A topic almost ignored in the presentation papers of METIS is the performance which is - beside model mapping capabilities - always an important factor on the success of a database system. The architecture and data model look pretty smooth but there is no data available comparing a single-media database to a domain specific implementation of METIS. Its very complex and comprehensive structure as well as its implementation in Java might be paid with an immense loss of performance compared to uniquely developed domain specific solutions crossing the borders of some media silos. This must be especially taken into account when dealing with computer graphic constructs or data that benefit from the use of hardware acceleration.

The missing user and rights management makes the METIS version described in [6] almost useless for commercial applications. Testing future releases stays interesting though. Beyond that the ability to model almost every requirement of an application within a unified environment makes METIS a very powerful tool. METIS is currently applied in several projects, e.g. a news video database that automatically classifies the corresponding audio streams. Future projects and field studies will reveal the true power of METIS.

Bibliography

[1] ISO/IEC 13249-1:2000. *Information technology - Database languages - SQL Multimedia and Application Packages - Part 1: Framework*. International Organization for Standardization, Geneva, Switzerland, 2000.

[2] E. Oomoto and K. Tanaka. OVID: Design and implementation of a video-object database system. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), 1993.

[3] Myron Flickner, Harpreet S. Sawhney, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23–32, 1995.

[4] Harald Kosch, László Böszörményi, Alexander Bachlechner, Christian Hanin, Christian Hofbauer, Margit Lang, Carmen Riedler, and Roland Tusch. Smooth - a distributed multimedia database system. In *VLDB*, pages 713–714, 2001.

[5] Jobst Löffler, Konstantin Biatov, Christian Eckes, and Joachim Köhler. Ifinder: an mpeg-7-based retrieval system for distributed multimedia content. In *ACM Multimedia*, pages 431–435, 2002.

[6] Niko Popitsch, Ross King, and Gerd Utz Westermann. Metis: a flexible foundation for the unified management of multimedia assets. *Multimedia Tools and Applications*, 33(3):325–349, 6 2007.

[7] Vincent Oria, M. Tamer Özsu, Paul Iglinski, Bing Xu, and L. Irene Cheng. Disima: An object-oriented approach to developing an image database system. In *ICDE*, pages 672–673, 2000.

[8] Michael Ortega, Yong Rui, Kaushik Chakrabarti, Kriengkrai Porkaew, Sharad Mehrotra, and Thomas S. Huang. Supporting ranked boolean similarity queries in mars. *IEEE Trans. Knowl. Data Eng.*, 10(6):905–925, 1998.