EMMA – A Query Algebra for Enhanced Multimedia Meta Objects

Sonja Zillner, Utz Westermann, and Werner Winiwarter

Department of Computer Science and Business Informatics
University of Vienna, Austria
{sonja.zillner,gerd-utz.westermann,werner.winiwarter}@univie.ac.at

Abstract. Enhanced Multimedia Meta Objects (EMMOs) are a novel approach to multimedia content modeling, combining media, semantic relationships between those media, as well as functionality on the media, such as rendering, into tradeable knowledge-enriched units of multimedia content. For the processing of EMMOs and the knowledge they contain, suitable querying facilities are required. In this paper, we present EMMA, an expressive query algebra that is adequate and complete with regard to the EMMO model. EMMA offers a rich set of formally-defined, orthogonal query operators that give access to all aspects of EMMOs, enable query optimization, and allow the representation of elementary ontology knowledge within queries. Thereby, EMMA provides a sound and adequate foundation for the realization of powerful EMMO querying facilities.

1 Introduction

Multimedia content formats we find today (e.g. SMIL[1], HyTime [2], and SVG [3]) primarily encode the presentation of content but not the information the content conveys. However, this *presentation-oriented* modeling only permits the hard-wired presentation of multimedia content; for advanced operations like retrieval and reuse of content, automatic composition, and adaptation of content to a user's needs, valuable information about the semantics of content is lacking. In parallel to research on the Semantic Web [4], one is able to observe a shift in paradigm towards a *semantic* modeling of multimedia content: not the presentation of media is described but their semantic interrelationships.

In order to facilitate a semantic modeling of multimedia content in content sharing and collaborative applications, we have developed *Enhanced Multimedia Meta Objects (EMMOs)* [5] in the context of the EU-funded CULTOS project[1]. EMMOs establish tradeable knowledge-enriched units of multimedia

[1] CULTOS was carried out from 2001 to 2003 by partners from 11 EU countries and Israel and aimed at providing a collaborative multimedia platform for researchers in intertextual studies enabling them to share and communicate their knowledge about the relationships between cultural artifacts. See http://www.cultos.org for more information.

content that indivisibly combine three of the content's aspects into a single object:

– *The media aspect:* an EMMO encapsulates the basic media objects of which the multimedia content consists.
– *The semantic aspect:* an EMMO further encapsulates semantic associations between its media objects.
– *The functional aspect:* an EMMO may define arbitrary, domain-specific operations on the content that can be invoked by applications, e.g. an operation dynamically rendering the EMMO considering the user's context.

EMMOs are *versionable,* enabling the collaborative authoring of multimedia content, and can be bundled and moved in their entirety including all three aspects and the versioning information enabling content sharing applications.

As part of the CULTOS project, a distributed infrastructure of *EMMO containers* [6] and an *authoring tool* for the creation of EMMOs were developed. The missing part in this infrastructure has been an adequate query mechanism for the access to and the processing of the information captured by EMMOs.

The contribution of this paper is to provide this missing part. We introduce *EMMA*, a query algebra for EMMOs. EMMA is adequate and complete with regard to the EMMO model, addressing the media, semantic, and functional aspects of an EMMO. Featuring an extensive set of orthogonal, formally defined query operators consisting of extraction operators, navigational operators, selection predicates, constructors, and a join operator, EMMA allows one to pose complex queries against EMMOs and facilitates sound query rewriting and optimization. The operators of EMMA are sufficiently expressive to represent elementary ontological knowledge within queries, such as supertype/subtype relationships, transitive and inverse associations, etc. Thus, EMMA constitutes a solid foundation for the implementation of advanced queries on EMMOs.

The remainder of the paper is organized as follows. Section 2 explains the EMMO model in more detail. Section 3 analyzes the requirements of a query algebra for EMMOs. Section 4 takes a look at related approaches and Sect. 5 presents the structure of EMMA and its major operator classes. Section 6 concludes this paper and gives an outlook to current and future work.

2 The EMMO Model

As mentioned before, an EMMO is a self-contained unit of multimedia content that encompasses three aspects, i.e. the media, semantic, and functional aspect, and provides versioning support. We use Fig. 1 showing the EMMO "Dracula Movies" to illustrate the EMMO model.

The formal constituents of the EMMO model are the so-called *entities*, which occur in four different kinds: *logical media parts*, *ontology objects*, *associations*, and *EMMOs* themselves. Each entity, regardless of its kind, is globally and uniquely identified by a *UUID* and carries a human-readable *name*, e.g. "The Cabinet of Dr. Caligari", which we employ in our figures for enhanced readability.

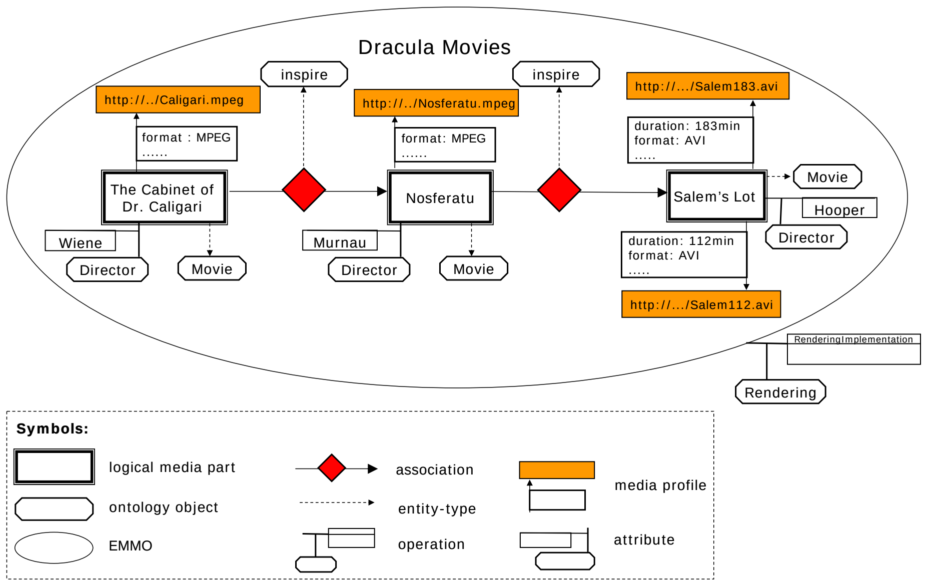

Fig. 1. EMMO "Dracula Movies" (e_{movies})

An EMMO addresses the *media aspect* of the piece of content it models by means of logical media parts. Logical media parts represent media objects or parts of media objects at a logical level. Media data that physically manifests these logical media objects can be attached to logical media parts via an arbitrary number of *media profiles*. A media profile not just directly embeds media data or – if embedding is not feasible, e.g., because of the size of the data or the media data is a live stream – references media data via an URI; it also carries arbitrary low-level, physical metadata about the media data in form of simple attribute-value pairs. In our example figure, this is illustrated with the logical media part "Salem's Lot" logically representing the corresponding movie. The attached media profiles indicate that there are two video files of the movie available ("Salem183.avi" and "Salem112.avi"). As expressed by the profiles' metadata attribute values, both video files are in AVI format and of 183 minutes and 112 minutes duration, respectively.

Addressing the content's *semantic aspect*, the EMMO model facilitates a semantically rich description of entities. An entity can be given an arbitrary number of *types*, thereby obtaining meaning. An entity type is a concept taken from an ontology, with the concept being represented as an ontology object thus being an entity itself. In our figure, for example, it is expressed that the logical media part "Salem's Lot" is of type "Movie". The EMMO model does not define an ontology itself; this has to be done by applications. As we have not developed a dedicated ontology language for EMMOs so far, we rely on existing languages like OWL[7] and RDF[8] Schema.

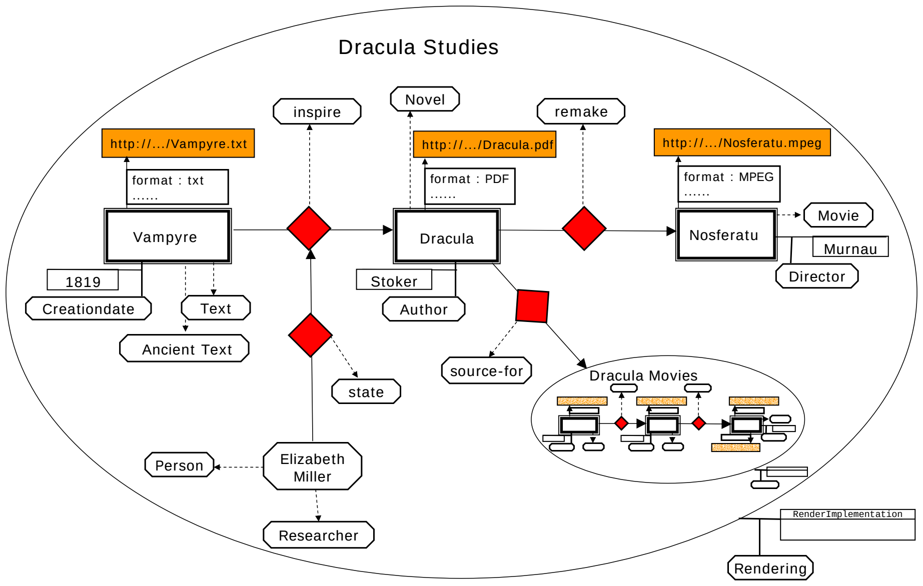

Fig. 2. EMMO "Dracula Studies" $(e_{studies})$

Semantic relationships between entities can be modeled by associations, which establish a binary directed relationship between a source and target entity. As associations are entities as well, the type of the relationship is given by the association's entity type. In this way, graph-based knowledge structures can be created. In the figure, for instance, it is stated by associations of type "inspire" between the depicted logical media parts that the movie "The Cabinet of Dr. Caligari" inspired the movie "Nosferatu", which again inspired "Salem's Lot". Moreover, as associations are first-class entities, they can take part in associations as well, effectively permitting the reification of statements within the EMMO model. For example, Fig. 2 articulates that the researcher "Elizabeth Miller" (represented as an ontology object of the same name) states that "Dracula" was inspired by "Vampyre".

As a further means of description, an arbitrary number of attribute-value pairs can be attached to an entity, with the attribute being again a concept of the ontology captured by an ontology object and the value being an object of arbitrary type. In Fig. 2, the attribute "Director" is attached to the logical media part "Nosferatu" with a string value "Murnau", expressing that the movie was directed by Friedrich Murnau.

EMMOs, finally, allow the grouping of semantically interrelated entities into a logical unit, thereby establishing pieces of semantically modeled multimedia content. In Fig. 1, the semantic descriptions of the logical media parts "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot" are grouped in the single EMMO "Dracula Movies". As EMMOs themselves are entities, they can be included within other EMMOs as well. Thus, EMMOs can be arbitrarily nested into hierarchical structures, a powerful means for the logical organization of multimedia content. The EMMO "Dracula Studies" in Fig. 2, for example, contains the EMMO "Dracula Movies". Also, an EMMO can take part in associations just like any other entity, allowing the expression of knowledge about the EMMO. Within the EMMO "Dracula Studies" it is stated that the novel "Dracula" was the source for the construction of EMMO "Dracula Movies".

Addressing the *functional aspect* of multimedia content, the EMMO model allows an EMMO to offer a set of *operations*. Each operation consists of an ontology object acting as the operator's *designator*, and the operation's *implementation*, which can be any mathematical function, taking an EMMO and an arbitrary sequence of parameters as its arguments. For example, EMMO "Dracula Movies" of Fig. 1 features an operation "rendering" that refers to a mathematical function which generates either an HTML or a SMIL document of the EMMO depending on the value of its single parameter.

In order to allow the collaborative construction of EMMOs in distributed scenarios, the EMMO model incorporates dedicated versioning support. An entity can refer to an arbitrary number of entities of the same kind as predecessor versions and an arbitrary number of entities as successor versions. As the version of an entity constitutes again an entity, different versions of an entity can be interrelated just like any other entities, allowing one to establish semantic relationships between versions. Fig. 3 shows several versions of the EMMO "Dracula Movies" and their interrelationships.

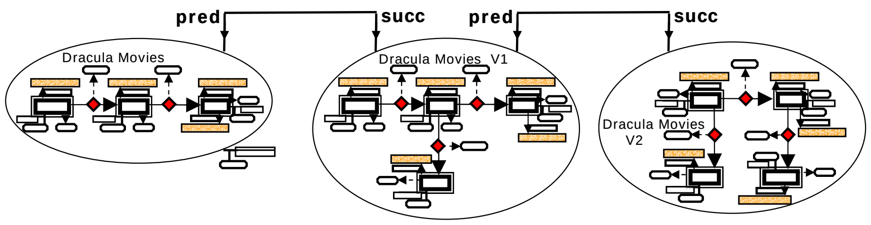

Fig. 3. The versioning information of EMMO "Dracula Movies"

3 Requirements of a Query Algebra for EMMOs

As a foundation for the querying of EMMO structures as described in the previous section, an algebra providing a set of formal query operators suiting the EMMO model is needed. In the following, we highlight essential requirements for such a query algebra.

First and most importantly, a proper EMMO query algebra has to be *adequate* and *complete* with regard to the EMMO model. Thus, the algebra should offer operators for the access to all three aspects of multimedia content that are covered by an EMMO:

– *Media aspect:* Operators should be available that give access to logical media parts and their media profiles in queries.
– *Semantic aspect:* The algebra should further offer operators for the querying of all kinds of entities contained in an EMMO, for the querying of the types of these entities and their attribute values, as well as for the traversal of the associations between them. The operators must be expressive enough to cope with the more advanced constructs of the EMMO model, such as the reification of associations and the nesting of EMMOs.
– *Functional aspect:* The algebra should give access to and permit the execution of the operations of an EMMO.

In order to fully suit the EMMO model, the algebra should also be able to deal with versioning and provide operators for the traversal of versioning relationships between entities.

Secondly, an EMMO query algebra should satisfy classic query algebra requirements. Its operators should be *formally* defined with *precise* semantics to lay the ground for query rewriting and optimization. The operators should also be *orthogonal* and arbitrarily *nestable* to facilitate expressive queries.

Thirdly, the algebra should support *joins* between entities in order to allow the correlation of information contained in different EMMOs. Furthermore – even though the *construction* and *manipulation* of EMMOs is not our primary intention in this paper (there exists a graphical authoring tool for EMMOs) – a suitable algebra should support some basic operators for this purpose, such as union, intersection, and difference.

Since the EMMO model makes use of concepts of an ontology (i.e., ontology objects) to describe the meaning of the entities contained in an EMMO and the associations between them, a suitable EMMO query algebra finally should be expressive enough to capture basic *ontological knowledge* within a query. For instance, it should be possible to reflect supertype/subtype relationships, transitive and inverse associations, etc.

4 Related Approaches

On the search for a suitable query algebra for EMMOs, we take a look at related query algebras and languages in the context of multimedia content and examine their adequacy and completeness with regard to the EMMO model.

In the literature, several query algebras for multimedia content have been proposed, such as GCalculus/S [9], Algebraic Video [10], or the Multimedia Presentation Algebra (MPA) [11]. These algebras have in common that they largely address the *media aspect* of multimedia content. They focus on the querying of the temporal and spatial presentation relationships between the basic media of multimedia content and the construction of new presentations out of these media. However, they ignore semantic relationships between media as well as the functional aspect of multimedia content.

In the context of the Semantic Web, several standards have emerged that can be used to model the semantic relationships between the basic media of multimedia content addressing the content's *semantic aspect*, such as RDF [12, 8], Topic Maps [13], and MPEG-7 (especially MPEG-7's `Graph` tools for the description of content semantics [14]). For these standards, a variety of proposals for query languages and algebras have been made.

Since the RDF data model, compared to the EMMO model, rather neglects the media aspect of multimedia content, it does not address the functional aspect of content, and does not provide explicit support for versioning and a hierarchical structuring of resource descriptions; the same is generally true for RDF-based query approaches as well. This leaves these approaches incomplete and inadequate with regard to the EMMO model.

Moreover, we find that many proposals of RDF query languages (representative examples are RQL [15] and SquishQL [16]) lack formally rigid definitions of the semantics of their operators and thus do not provide sound foundations for query evaluation, rewriting, and optimization. The only formal RDF query algebra we know of that has been developed with the optimization of RDF queries in mind is RAL [17].

The situation for Topic Maps is quite similar to RDF. The Topic Map data model focuses on the semantic aspect as well and – considering the EMMO model's ability to include raw media data and metadata about the media by means of media profiles within an EMMO – neglects the media and functional aspects of multimedia content. Moreover, although Topic Maps like EMMOs can be hierarchically nested, they have no explicit versioning support. Consequently, query languages for Topic Maps are generally incomplete and inadequate with regard to the EMMO model.

Within the context of the ongoing standardization of a Topic Maps query language TMQL [18], several query approaches, such as Tolog [19], TMPath [20], XTMPath [21], or [22] have been introduced. But again, those proposals remain on the syntactic level and do not provide formal definitions of their operators. No formal algebra as a sound foundation for the querying of Topic Maps exists so far.

Concerning the querying of semantic descriptions of multimedia content on the basis of MPEG-7's `Graph` tools, we find quite a few approaches adapting XQuery for the querying of MPEG-7 media descriptions [23]. But these approaches do not provide specific operators that would allow a reasonable processing of the `Graph` tools.

To summarize, we have not been able to find a formally sound foundation that would allow an adequate querying of EMMOs. Although there are some formal algebras available for querying the media aspect of multimedia content like GCalculus/S, Algebraic Video, or MPA, and the semantic aspect of multimedia content such as the RDF-based RAL, they are neither adequate nor complete with regard to the EMMO model, which addresses the media, semantic, as well as the functional aspects of multimedia content (not to mention the EMMO model's inherent support for versioning).

As a consequence, we were forced to develop a dedicated algebra to obtain a sound foundation for EMMO querying. At least for the design of this algebra, however, we were able to gain valuable insights from the approaches examined above and to incorporate aspects of their design.

5 EMMA – The EMMO Algebra

The design of the EMMO query algebra EMMA was in the first place driven by the requirement of accessing the complete information stored within an EMMO, i.e. the access to the three aspects of the EMMO, as well as its versioning information. To enable query optimization, the query algebra's operators are of limited complexity and orthogonal. Through the combination and nesting of modular operators, complex queries can be formulated.

EMMA's query operators can be divided into five general classes: the *extraction operators* provide means to query an EMMO's three aspects, as well as its versioning information. The *navigational operators* allow the navigation along an EMMO's semantic graph structure and also facilitate the integration of basic ontological knowledge. The *selection predicates* enable the selection of only those entities fulfilling a specific characteristic. The *constructors* allow one to modify, combine, and create new EMMOs, and finally, the *join operator* relates several entities or EMMOs with a join condition.

In the following subsections, we introduce all five classes of EMMA operators along illustrative examples. Due to limited space, we only discuss some representative examples of operators and cannot provide the formal definitions of these operators. The complete list of operators and their formal definition can be found in [24]. Finally, we conclude this section with a summary explaining how these operators contribute to fulfil the requirements for an EMMO query algebra.

5.1 Extraction Operators

The extraction operators allow access to the information stored within an EMMO. In the following, we show examples of extraction operators for the three different aspects, as well as for the versioning information.

Media Aspect

Logical media parts model media objects at a logical level, and additionally maintain connections to media profiles representing these objects along with their metadata. For attaining all logical media parts contained within an EMMO, the operator *lmp* can be used, e.g. the operation

$$lmp(e_{movies}) = \{l_{caligari}, l_{nosferatu}, l_{salem}\}$$

yields the three logical media parts "The Cabinet of Dr. Caligari", "Nosferatu" and "Salem's Lot" contained within EMMO "Dracula Movies"in Fig. 1.

The operator *MediaProfiles* can be used for locating media profiles. Applying the operator *MediaProfiles* to a logical media part returns the union of all its associated media profiles, e.g. the query expression

$$\begin{aligned}\mathit{MediaProfiles}(l_{salem}) = \{&(\texttt{www.../Salem183.avi}, \{(\text{``}\mathit{duration}\text{''}, \mathit{183min}), (\text{``}\mathit{format}\text{''}, \mathit{AVI})\}),\\ &(\texttt{www.../Salem112.avi}, \{(\text{``}\mathit{duration}\text{''}, \mathit{112min}), (\text{``}\mathit{format}\text{''}, \mathit{AVI})\})\}\end{aligned}$$

gives a set of two media profiles, each of them consisting of a URI locating the media data and a metadata set describing the low-level characteristics of the media data. The algebra provides further operators to extract the media data as well as the metadata from a given media profile, e.g.

$$\mathit{MediaInstance}((\texttt{www.../Salem183.avi}, \{(\text{``}\mathit{duration}\text{''}, \mathit{183min}, \ldots)\})) = \texttt{www.../Salem183.avi},$$

extracts the URI pointing to the media data from the given media profile. Similarly, the operator *Metadata* extracts the physical metadata from the profile.

Semantic Aspect

By attaching concepts of an ontology, entities get meaning. The operator *types* retrieves an entity's set of classifying ontology objects. For example, applying the operator *types* to the logical media part "Nosferatu", yields the set containing the ontology object "Movie":

$$\mathit{types}(l_{nosferatu}) = \{o_{movie}\}.$$

The operator *types* accepts only one entity as input value. If we intend to compute all types classifying not only one, but a set of entities, the operator *types* can be used in combination with the operators *Apply* and *Elements*. The operator *Apply* takes a function and a set as input values and returns a set consisting of the return values of the specified function being applied to each element in the specified set. For example, for accessing all ontology objects used for classifying logical media parts within EMMO "Dracula Studies" in Fig. 2, we execute the operator *Apply* with the operator *types* and the set of logical media parts of EMMO "Dracula Studies" specified as input values, e.g.

$$\begin{aligned}&\mathit{Apply}(\mathit{types}, \mathit{lmp}(e_{studies})) =\\ &= \mathit{Apply}(\mathit{types}, \{l_{vampyre}, l_{dracula}, l_{nosferatu}\}) =\\ &= \{\mathit{types}(l_{vampyre}), \mathit{types}(l_{dracula}), \mathit{types}(l_{nosferatu})\} =\\ &= \{\{o_{ancient\text{-}text}, o_{text}\}, \{o_{novel}\}, \{o_{movie}\}\}\end{aligned}$$

The operator *Elements* is used to flatten data returned by other operations, e.g. applying the operator *Elements* to the result set of the above query, i.e.

$$\begin{aligned}&\mathit{Elements}(\{\{o_{ancient\text{-}text}, o_{text}\}, \{o_{novel}\}, \{o_{movie}\}\}) =\\ &= \{o_{ancient\text{-}text}, o_{text}, o_{novel}, o_{movie}\},\end{aligned}$$

returns the set of all ontology objects used for classifying the logical media parts within EMMO "Dracula Studies".

For querying the attribute values of an entity, the operator *attributes* can be used. Requesting, for example, all attribute-value pairs of the logical media part "Nosferatu", i.e.

$$attributes(l_{nosferatu}) = \{(o_{director}, \text{``}Murnau\text{''})\},$$

yields the set including only one specified attribute-value pair, i.e. the ontology object "Director" with the string-value "Murnau".

EMMOs describe a graph-like knowledge structure of entities. The algebra introduces the operator *nodes* for accessing all entities contained within an EMMO, e.g. the query operation

$$\begin{aligned} nodes(e_{studies}) = \{&l_{vampyre}, l_{dracula}, l_{nosferatu}, e_{movies}, o_{miller},\\ &a_{va\to dr}, a_{dr\to no}, a_{dr\to mo}, a_{mi\to(va\to dr)}\} \end{aligned}$$

provides a set consisting of the logical media parts representing the movie "Vampyre", Stoker's novel "Dracula", and the movie "Nosferatu"; the EMMO "Dracula Movies"; the ontology object representing the researcher "Elizabeth Miller"; and additionally the associations representing the semantic relationships between those entities, i.e. the associations "Vampyre → Dracula", "Dracula → Nosferatu", "Dracula → Dracula Movies", and "Elizabeth Miller → (Vampyre → Dracula)".

The algebra also features operators for the traversal of the semantic associations between entities. These will be explained in Subsect. 5.2.

EMMOs can be nested hierarchically. The operator *AllEncEnt* can be used for accessing *all enc*apsulated *ent*ities of an EMMO, i.e. it computes all entities recursively contained within an EMMO. For example, the query expression

$$\begin{aligned} AllEncEnt(e_{studies}) &= nodes(e_{studies}) \cup nodes(e_{movies}) = \\ &= \{l_{vampire}, l_{dracula}, l_{nosferatu}, e_{movies}, o_{miller},\\ &\quad a_{va\to dr}, a_{dr\to no}, a_{dr\to mo}, a_{mi\to(va\to dr)},\\ &\quad l_{caligari}, l_{salem}, a_{ca\to no}, a_{no\to sa}\} \end{aligned}$$

unifies the nodes of EMMO "Dracula Studies" with the nodes of EMMO "Dracula Movies", because this EMMO is the only one contained within EMMO "Dracula Studies" and contains no further EMMOs.

Functional Aspect

EMMOs offer functions for dealing with their content. The operator *Designators* can be used to receive all ontology objects labeling an EMMO's functions, e.g. the result set of the query

$$Designators(e_{movies}) = \{o_{rendering}\}$$

indicates that EMMO “Dracula Movies” in Fig. 1 offers a rendering functionality, and the operator *ImpToName* allows access to the corresponding implementation represented by a mathematical function, i.e.

$$ImpToName(e_{movies}, o_{rendering}) = f_{render}$$

with f_{render} being some rendering function. For the execution of an EMMO's functionality, the query algebra EMMA specifies the operator *Execute*. Applying the operator *Execute* to EMMO “Dracula Movies”, the ontology object “rendering”, and the parameter `HTML`, i.e.

$$Execute(e_{movies}, o_{rendering}, \texttt{HTML}) = f_{render}(e_{movies}, \texttt{HTML}) = \texttt{DraculaMovies.html},$$

returns an HTML-document representing the content of EMMO “Dracula Movies”, for example, an HTML-document of a table with the rows being the EMMO's associations as illustrated in the left part of Fig. 4.

Applying the operator *Execute* to the same EMMO and the same ontology object, but the parameter `SMIL`, i.e.

$$Execute(e_{movies}, o_{rendering}, \texttt{SMIL}) = f_{render}(e_{movies}, \texttt{SMIL}) = \texttt{DraculaMovies.smil},$$

yields a SMIL-document about the EMMO's content, for example, a SMIL-document sequentially representing the EMMO's associations as illustrated in the right part of Fig. 4.

```
<html>
<body>
<h1>EMMO Dracula Movies</h1>
<table border="1">
<tr><th>Source</th>
<th>Association</th>
<th>Target</th></tr>
<tr><td>
<a href=".../Caligari.mpeg">The ..Caligari</a></td>
<td>Inspire</td>
<td><a href=".../Noseferatu.mpeg">Nosferatu</a></td></tr>
<tr><td><a href=".../Noseferatu.mpeg">Nosferatu</a></td>
<td>Inspire</td>
<td><a href=".../Salem183.avi">Salem's Lot</a><br>
<a href=".../Salem112.avi">Salem's Lot</a>
</td></tr>
</table>
</body>
</html>
```

```
<smil>
<head><layout>
<root-layout height="200" width="620"/>
<region id="l" left="0" ..../> .......
</layout></head>
<body> <seq>
<par end="60s" >
<video src="./Caligari.mpeg" type="video/mpeg" region="l"/>
<text src="./inspire.txt" type="text/plain" region="m"/>
<video src="./Nosfertatu.mpeg" type="video/mpeg" region="r"/>
</par>
<par end="60s" >
<video src="./Nosferatu.mpeg" type="video/mpeg" region="l"/>
<text src="./inspire.txt" type="text/plain" region="m"/>
<video src="./Salem183.avi" type="video/mpeg" region="r"/>
</par>
</seq></body>
</smil>
```

Fig. 4. DraculaMovies.html and DraculaMovies.smil

Versioning

Each entity describes a set of succeeding and a set of preceding versions. The operator *successors* can be used for accessing all direct successors of an entity, e.g. the query expression

$$successors(e_{movies}) = \{e_{moviesV1}\}$$

returns EMMO "Dracula Movies – V1", the one direct successor version of EMMO "Dracula Movies" (see Fig. 3). For accessing all succeeding versions, the operator *AllSuccessors* is applied, e.g.

$$\mathit{AllSuccessors}(e_{movies}) = \{e_{moviesV1}, e_{moviesV2}\}.$$

For the access of an entity's preceding versions, EMMA also provides the operators *predecessors* and *AllPredecessors*, which are defined in a similar way.

5.2 Navigational Operators

An EMMO establishes a graph-like knowledge structure of entities with associations being labeled by ontology objects describing the edges in the graph structure. The navigational operators provide means for traversing the semantic graph structure of an EMMO. Navigation through an EMMO's graph structure is controlled by a navigation path defined as a set of sequences of ontology objects. A mapping for each ontology object in the sequence to the corresponding association of an EMMO defines the traversal path of the graph structure. We have defined regular path expressions over ontology objects for describing the syntax of a navigation path. Navigational operators take a regular path expression as input and specify how this syntactic expression is applied to navigate the graph structure. For example, for a given EMMO, start entity, and regular path expression, the navigational operator *JumpRight* returns the set of all entities that can be reached by traversing the navigation path in the right direction, i.e. by following associations from source to target entities. Applying the operator *JumpRight* to EMMO "Dracula Movies – V1"(see Fig. 5), the starting entity "The Cabinet of Dr. Caligari", and the regular path expression consisting of only one single ontology object "$o_{inspire}$" yields the logical media part representing the movie "Nosferatu":

$$\mathit{JumpRight}(e_{\mathit{movies\,V1}}, l_{caligari}, o_{inspire}) = \{l_{nosferatu}\}.$$

The basic building blocks of regular path expressions are ontology objects which can be modified and combined using conventional regular expression operators. For example, adding the operator "+" to the regular path expression of the above query returns two logical media parts representing the movies "Nosferatu" and "Salem's Lot":

$$\mathit{JumpRight}(e_{\mathit{movies\,V1}}, l_{caligari}, o_{inspire}+) = \{l_{nosferatu}, l_{salem}\}.$$

Regular path expressions can also be concatenated and defined as optional. For example, applying the operator *JumpRight* to EMMO "Dracula Movies – V1", the start entity "The Cabinet of Dr. Caligari" and the regular path expression "$o_{inspire} o_{rework}?$", yields the logical media parts "Nosferatu" and "Van Helsing":

$$\mathit{JumpRight}(e_{\mathit{movies\,V1}}, l_{caligari}, o_{inspire} o_{rework}?) = \{l_{nosferatu}, l_{helsing}\}.$$

The choice operator "|" can be used to combine regular path expression as alternate versions, e.g.

$$JumpRight(e_{movies\,V1}, l_{nosferatu}, o_{inspire}\,|\,o_{rework}) = \{l_{salem}, l_{helsing}\}.$$

By adding the operator "−" to a regular path expression, the inversion of the regular path expression, i.e. the change of direction of navigation, can be expressed, e.g.

$$JumpRight(e_{movies\,V1}, l_{helsing}, o_{rework}-) = \{l_{nosferatu}\}.$$

Traversal along the opposite direction of associations can also be expressed with the navigational operator *JumpLeft*, e.g.

$$JumpLeft(e_{movies\,V1}, l_{helsing}, o_{rework}) = JumpRight(e_{movies\,V1}, l_{helsing}, o_{rework}-).$$

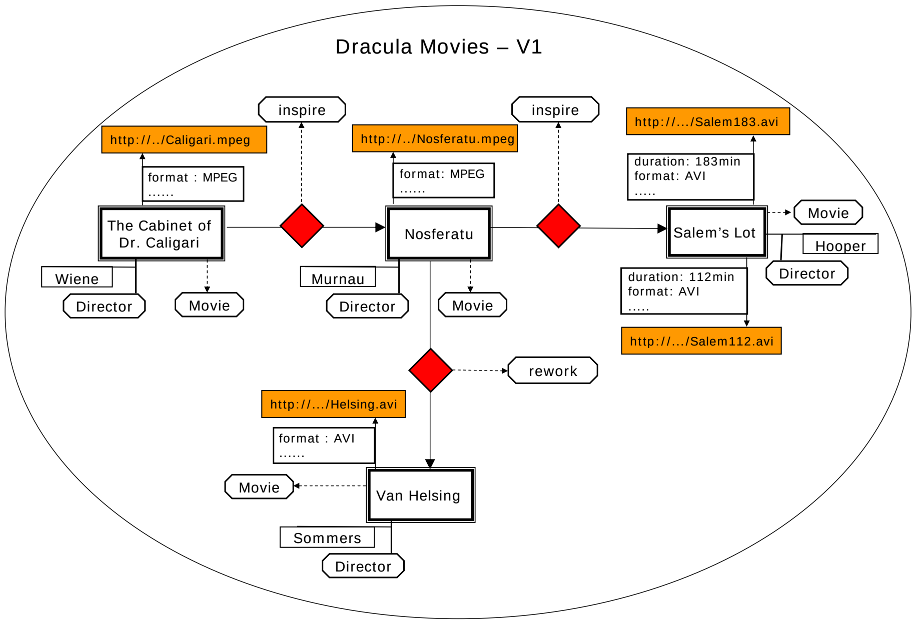

Fig. 5. EMMO "Dracula Movie – V1" ($e_{movieV1}$)

Navigational accessors provide the basis for the integration of basic ontological knowledge into queries. For example, the transitivity of association types, such as the transitivity of associations of type "inspire", can be reflected by replacing the navigation path $o_{inspire}$ by the navigation path $o_{inspire}+$ (see example above). Knowledge about inverse association types, such as the association types "rework" and "is-reworked", can be integrated within the queries as well, for instance, by replacing the navigation path $o_{is-reworked}$ by the navigation path $o_{is-reworked} | o_{rework}-$, e.g.

$$\mathit{JumpRight}(e_{moviesV1}, l_{helsing}, o_{is-reworked} | o_{rework}-) = \{l_{nosferatu}\}.$$

5.3 Selection Predicates

The selection predicates allow the selection of only those entities satisfying a specific characteristic. They basically use the result values of extraction operators to create Boolean operators. The selection predicates can be combined with the generic *Select* operator, which takes a predicate and an arbitrary set as input values, and returns all elements of the set that satisfy the condition of the specified predicate. For instance, applying the operator *IsType* to the logical media part "Dracula" and the set of the one ontology object "Book" returns false:

$$\mathit{IsType}(l_{dracula}, \{o_{book}\}) = \mathit{false}.$$

By taking a set of ontology objects as input parameter, the operator *IsType* enables the integration of supertype/subtype relationships within queries. The ontological knowledge about a subtype relationship, for example, the subtype relationship between the ontology objects "Novel" and "Book" can be reflected within the query expression, e.g.

$$\mathit{IsType}(l_{dracula}, \{o_{book}, o_{novel}\}) = \mathit{true}.$$

Assuming that ontological knowledge about supertype/subtype relationships was represented within EMMOs (e.g. in EMMO $e_{ontology}$) as well, e.g., by means of associations of type "is_a", the subtypes of "Book" in the previous query would not need to be hardwired but could also be dynamically calculated during query execution using an appropriate *JumpRight* expression:

$$\mathit{IsType}(l_{dracula}, \mathit{JumpRight}(e_{ontology}, o_{book}, o_{is\_a}*)) = \mathit{true}.$$

Although we have not developed a language yet which governs the expression of such ontology knowledge within the EMMO model, the query algebra in this manner is prepared for exploiting this knowledge once it becomes available.

If we apply the *Select* operator to the selection predicate *IsType* with the set consisting of the ontology objects "Book"and "Novel" as fixed parameter value and to the logical media parts contained within EMMO "Dracula Studies" (see Fig. 2), the result set consists of the logical media part representing Stoker's novel "Dracula":

$$\mathit{Select}(\mathit{IsType}_{[\$, \{o_{book}, o_{novel}\}]}, \mathit{lmp}(e_{studies})) = \{l_{dracula}\}.$$

By combining selection predicates with logical predicates, such as *And*, *Or*, and *Not*, we can ask, for example, for all logical media parts within EMMO "Dracula Studies" which are not of type "Novel":

$$\mathit{Select}(\mathit{Not}(\mathit{IsType}_{[\$, \{o_{novel}\}]}, \mathit{lmp}(e_{studies}))) = \{l_{vampyre}, l_{nosferatu}\}.$$

Being based on the return values of extraction operators, the list of selection predicates has the same length as the list of extraction operators. Any information which can be accessed by the extraction operators is again used for the selection of entities.

Thus, for example, selection predicates allow the selection of all logical media parts within EMMO "Dracula Movies" (see Fig. 1) associating a media profile encompassing media data in AVI format, i.e.

$$Select(HasMediaProfileValue_{[\$, \text{“}format\text{”}, \text{“}AVI\text{”}, =]}, lmp(e_{movies})) = \{l_{salem}\},$$

yields the logical media part "Salem's Lot" encompassing two media profiles which both describe the attribute "format" with value "AVI" within their sets of metadata.

5.4 Constructors

EMMA specifies five constructors for EMMOs, i.e. the operators *Union*, *Nest*, *Flatten*, *Difference*, and *Intersection*. All the constructors take at least one EMMO and possibly other parameters as input value, and return exactly one EMMO as output value. For example, the *Difference* operator takes two EMMOs and a string value. It creates a new EMMO which is denoted by the specified string value. The new EMMO's nodes encompass all entities belonging to the first, but not the second EMMO, and additionally the source and target entities of each association contained within the first EMMO. Otherwise, an EMMO constructed by the *Difference* operator could encompass incomplete associations without source or target entity. The remaining properties of the new EMMO, such as its *operations* or *predecessors* sets are specified as empty set. Applying

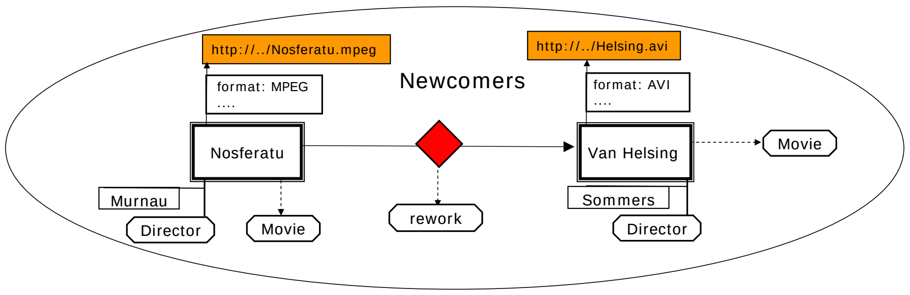

Fig. 6. EMMO "Newcomers" ($e_{newcomers}$)

the *Difference* operator to the successor EMMO "Dracula Movies – V1" and the original EMMO "Dracula Movies", generates a new EMMO "Newcomers" (see Fig. 6) consisting of the logical media parts describing the movies "Nosferatu" and "Van Helsing", as well as their connecting "reworking" association, i.e.

$$\begin{aligned}&\mathit{Difference}(e_{moviesV1}, e_{movies}, \text{“}\mathit{Newcomers}\text{”}) = e_{newcomers}\\ &\text{with} \qquad nodes(e_{newcomers}) = \{l_{nosferatu}, a_{no\rightarrow he}, l_{helsing}\}.\end{aligned}$$

The *Nest* operator maps the information stored within an association, i.e. the triple consisting of source entity, association, and target entity, into an EMMO knowledge structure. It takes an EMMO, a string value, and a set of associations as input values and creates a new EMMO encompassing a subgraph consisting of only those associations together with their source and target entities. Applying the *Nest* operator to EMMO "Dracula Studies" (see Fig. 2) and to the associations which were stated by "Elizabeth Miller", i.e. the return value of the operation $\mathit{JumpRight}(e_{studies}, o_{miller}, o_{state})$:

$$\begin{aligned}&\mathit{Nest}(e_{studies}, \text{“}\mathit{Miller's\ Statements}\text{”}, \mathit{JumpRight}(e_{studies}, o_{miller}, o_{state})) = e_{miller}\\ &\text{with} \qquad nodes(e_{miller}) = \{l_{vampyre}, a_{va\rightarrow dr}, l_{dracula}\}\end{aligned}$$

constructs a new EMMO encompassing three entities, i.e. the ancient text "Vampyre", the book "Dracula", and the connecting association of type "inspire" as illustrated in Fig. 7.

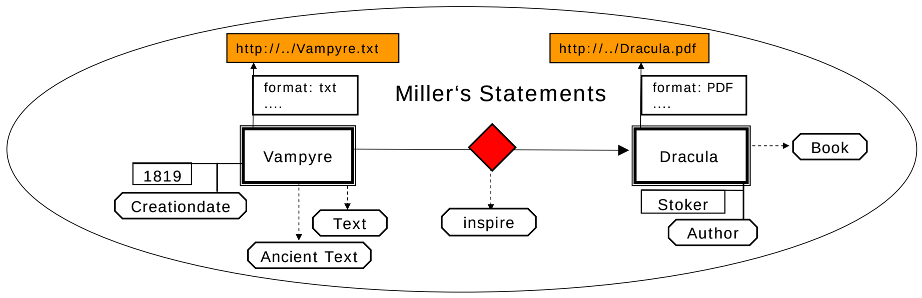

Fig. 7. EMMO "Miller's Statements" (e_{miller})

5.5 Join Operator

The *Join* operator renders it possible to extend queries across multiple EMMOs. It specifies how to relate n sets of entities, possibly originating from different EMMOS, within a query. The *join* operator takes n entity sets, n operators, and one predicate as input value. We compute the Cartesian product of the n entity sets and select only those tuples that satisfy the predicate after applying the n operators to the n entities. The result set of tuples is projected onto the first entry. For example, asking for all successors of EMMO "Dracula Movies" which constitute an extended version of the original version, i.e. asking for all succeeding EMMOs which at least encompass the entities contained within the original EMMO "Dracula Movie", corresponds to the query expression

$$Join(AllSuccessors(e_{movies}), \{e_{movies}\}, nodes, nodes, \supseteq) = \{e_{movies\,V1}\}$$

and yields EMMO "Dracula Movies – V1" (see Fig. 5) , because this succeeding EMMO encompasses – in addition to the entities already contained within EMMO "Dracula Movies" – two further entities, i.e. the "reworking" association with the logical media part "Van Helsing" as target entity.

The *join* operator is a generalization of the *Select* operator accounting for constraints defined on not only one but several entity sets. Defining the *identity* function *id*, i.e. $id(x) = x$, any select operation can be expressed by a join expression taking only one set, one operator, and one predicate p as input value, e.g.

$$Join(nodes(e_{studies}), id, p) = Select(p, nodes(e_{studies})).$$

5.6 Summary of EMMA Operators

Figure 8 summarizes the contribution of the EMMA operators introduced in the preceding subsections in satisfying the requirements of an EMMO query algebra as described in Sect. 3.

adequacy and completeness				orthogonality	joins	construction and manipulation	presentation of ontological knowledge
media aspect	semantic aspect	functional aspect	versioning				
lmp MediaProfiles	types attributes nodes AllEncEnt JumpRight JumpLeft	Designators lmpToName Execute	Successors Predecessors AllSuccessors AllPredecessors	Select Apply Elements Nest	Join	Union Intersection Difference Nest Flatten	types IsType JumpRight JumpLeft

Fig. 8. EMMA operators addressing the EMMA requirements

By offering operators to access the three aspects and the versioning information, EMMA is *adequate* and *complete* with regard to the EMMO model. The access to EMMO's *media aspect* is realized by the operator *lmp* retrieving all logical media parts, and the operator *MediaProfiles* returning all media profiles of a logical media part. For accessing the *semantic aspect*, EMMA provides the operator *types* accessing the types of an entity, the operator *attributes* returning an entity's attribute values, the operator *nodes* yielding all entities within an EMMO, the operator *AllEncEnt* attaining all recursively contained entities within an EMMO, and the operators *JumpRight* and *JumpLeft* enabling the navigation of an EMMO's graph structure. The operators *Designator*, *ImpToName*, and *Execute* address the *functional aspect*, and the operators *successors* (*predecessors*) and *AllSuccessors* (*AllPredecessors*) ensure the access to the versioning information.

The ability to arbitrarily nest and combine operators indicates the high *orthogonality* of EMMA's operators. The basic *Select* operator takes a selection predicate and an arbitrary set – possibly the return set of another EMMA operation. The operator *Apply* allows one to use a specified operator not only for a single input value, but for a set of input values. As some of the operator's output values are represented in a format which cannot be directly used as input value for other operators, EMMA provides operators to transform and prepare the data for the use by other operators: the operator *Elements* allows the flattening of data sets and the *Nest* operator facilitates the nesting of an arbitrary set of associations into an EMMO knowledge container.

By extending queries across multiple EMMOs and entities, the *join* operator allows one to correlate the information contained in different EMMOs. The construction operators establish primitive operators for the construction and manipulation of EMMOs.

Finally, EMMA allows one to capture basic ontological knowledge within a query. Within the EMMO model, ontological knowledge is represented by ontology objects. The operator *types* accesses the classification of an entity (represented by a set of ontology objects) and the operator *IsType* the entities of specific types. As the operators *JumpRight* and *JumpLeft* allow the specification of navigation along associations by means of powerful regular path expressions, they are able to consider basic ontological knowledge such as transitive and inverse association types, and supertype/subtype relationships.

6 Conclusion

In this paper, we have introduced the EMMA query algebra for EMMOs, a novel approach to semantic multimedia content modeling for collaborative and content sharing applications. EMMA is adequate and complete with regard to the EMMO model and formally defined and orthogonal, establishing a foundation for the querying of EMMOs and a formally sound basis for query rewriting and optimization. EMMA is expressive, featuring orthogonal, arbitrarily combinable operators that range from simple selection and extraction operators to more complex navigational operators and joins and even rudimentary operators for the construction and manipulation of EMMOs. Furthermore, EMMA is capable of capturing basic ontological knowledge within queries, such as supertype/subtype relationships, transitive or inverse association types.

Currently, we are implementing the algebra and developing a cost model for its operators based on the experiences with this implementation. Based on the cost model, we will derive and formally prove elementary query rewriting rules with a high potential of saving query evaluation time. Furthermore, we are in the process of providing the proof for elementary, formal properties of the algebra, such as completeness, etc. Moreover, we are developing a language for the definition of ontologies that is compatible with EMMOs to allow the seamless integration of ontological knowledge into query processing.

Acknowledgement

We would like to thank Bernhard Haslhofer for his work on the implementation of the EMMA query algebra.

References

1. Ayars, J., et al.: Synchronized Multimedia Integration Language (SMIL 2.0). W3C Recommendation, World Wide Web Consortium (W3C) (2001)
2. ISO/IEC JTC 1/SC 34/WG 3: Information Technology – Hypermedia/Time-Based Structuring Language (HyTime). International Standard 15938-5:2001, ISO/IEC (1997)
3. Ferraiolo, J., Jun, F., Jackson, D.: Scalable Vector Graphics (SVG) 1.1. W3C Recommendation, World Wide Web Consortium (W3C) (2003)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
5. Schellner, K., Westermann, U., Zillner, S., Klas, W.: CULTOS: Towards a World-Wide Digital Collection of Exchangeable Units of Multimedia Content for Intertextual Studies. In: Proceedings of the Conference on Distributed Multimedia Systems (DMS 2003), Miami, Florida (2003)
6. Westermann, U., Zillner, S., Schellner, K., Klas, W.: EMMOs: Tradeable Units of Knowledge Enriched Multimedia Content. In Srinivasan, U., Nepal, S., eds.: Managing Multimedia Semantics. IDEA Group Publishing, Hershey PA, USA (to appear)
7. Schneider, P., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, World Wide Web Consortium (W3C) (2004)
8. Brickely, D., Guha, R.: Resource Description Framework (RDF) Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, World Wide Web Consortium (W3C) (2002)
9. Lee, T., et al.: Querying Multimedia Presentations Based on Content. IEEE Transcations on Knowledge and Data Engineering **11** (1999)
10. Duda, A., Weiss, R., Gifford, D.: Content Based Access to Algebraic Video. In: Proceedings of the IEEE First International Conference on Multimedia Computing and Systems, Boston, MA,USA (1994)
11. Adali, S., Sapino, M., Subrahmanian, V.: A Multimedia Presentation Algebra. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA (1999)
12. Lassila, O., Swick, R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, World Wide Web Consortium (W3C) (1999)

13. ISO/IEC JTC 1/SC 34/WG 3: Information Technology – SGML Applications – Topic Maps. ISO/IEC International Standard 13250:2000, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) (2000)
14. ISO/IEC JTC 1/SC 29/WG 11: Information Technology – Multimedia Content Description Interface – Part 5: Multimedia Description Schemes. Final Draft International Standard 15938-5:2001, ISO/IEC (2001)
15. Karvounarakis, G., et al.: RQL: A Declarative Query Language for RDF. In: Proceedings of the 11th International World Wide Web Conference (WWW 2002), Honolulu, Hawaii (2002)
16. Miller, L., Seaborn, A., Reggiori, A.: Three Implementations of SqishQl, a Simple RDF Query Language. In: Proceedings of the first International Semantic Web Conference (ISWC2002), Sardinia, Italy (2002)
17. Frasincar, F., et al.: RAL: An Algebra for Querying RDF. In: Proceedings of the Third International Conference on Web Information Systems Engineering (WISE 2000), Singapore (2002)
18. ISO/IEC JTC1 SC34 WG3: New Work Item Proposal, Topic Map Query Language (TMQL). New Proposal, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) (2000)
19. Garshol, L.: tolog 0.1. Ontopia Technical Report, Ontopia (2003)
20. Bogachev, D.: TMPath – Revisited. Online Article, available under http://homepage.mac.com/dmitryv/TopicMaps/TMPath/TMPathRevisited.html (2004)
21. Barta, R., Gylta, J.: XTM::Path – Topic Map management, XPath like retrieval and construction facility. Online Article, available under http://cpan.uwinnipeg.ca/htdocs/XTM/XTM/Path.html (2002)
22. Widhalm, R., Mück, T.: Topic Maps (in German). Springer, Berlin Heidelberg, Germany (2002)
23. Manjunath, B., Salembier, P., Sikora, T., eds.: Introduction to MPEG-7. John Wiley & Sons, West Sussex, UK (2002)
24. Zillner, S.: The EMMA Algebra for EMMOs – Compendium. Technical Report TR 2004 301, Department of Computer Science and Business Informatics, University of Vienna (2004) available at http://www.ifs.univie.ac.at/˜sz/EMMA-Compendium.pdf.