EMMA – A Formal Basis for Querying Enhanced Multimedia Meta Objects

Sonja Zillner and Werner Winiwarter

Faculty of Computer Science,
University of Vienna,
Liebiggasse 4, A-1010 Vienna,
sonja.zillner@univie.ac.at

Abstract. Today's multimedia content formats primarily encode the presentation of content but not the information the content conveys. However, this presentation-oriented modeling only permits the inflexible, hard-wired presentation of multimedia content. For the realization of advanced operations like the retrieval and reuse of content, automatic composition, or adaptation to a user's needs, the multimedia content has to be enriched by additional semantic information, e.g. the semantic interrelationships between single multimedia content items. Enhanced Multimedia Meta Objects (EMMOs) are a novel approach to multimedia content modeling, which combines media, semantic relationships between those media, as well as functionality on the media (such as rendering) into tradeable and versionable knowledge-enriched units of multimedia content. For the processing of EMMOs and the knowledge they incorporate, suitable querying facilities are required. Based on the formal definition of the EMMO model, in this paper, we propose and formally define the EMMO Algebra EMMA, a query algebra that is adequate and complete with regard to the EMMO model. EMMA offers a rich set of orthogonal query operators, which are sufficiently expressive to provide access to all aspects of EMMOs and enable efficient query rewriting and optimization. In addition, they allow for the seamless integration of ontological knowledge within queries, such as supertype/subtype relationships, transitive and inverse associations, etc. Thus, EMMA represents a sound and adequate foundation for the realization of powerful EMMO querying facilities. We have finished the implementation of an EMMO container environment and an EMMA query execution engine, and are currently in the process of evaluating the query algebra in several case studies.

1 Introduction

For the presentation and rendering of multimedia content, there exist several multimedia content formats, such as HTML [1], SMIL [2], or SVG [3]. Those approaches have all in common that they merely focus on the encoding of the content, but neglect the information the content conveys. As those multimedia content formats are limited to the modeling of presentation-related issues of multimedia content, only the generation of inflexible hard-wired multimedia

presentations can be realized. As a prerequisite for advanced operations, such as retrieval and reuse of content, automatic composition, and adaptation of content to a user's needs, additional information about the content's semantics has to be provided. Triggered by the research in the context of the Semantic Web initiative [4], several attempts have been undertaken to integrate semantics into the modeling of multimedia content. For recent publications on multimedia semantics see [5], [6], [7], [8], [9], [10], an up-to-date overview is given in [11].

To facilitate the semantic modeling of multimedia content in content sharing and collaborative applications, we have developed *Enhanced Multimedia Meta Objects (EMMOs)* [12] in the context of the EU-funded CULTOS project[1]. An EMMO establishes a self-contained unit of multimedia content indivisibly unifying three aspects of multimedia content: The *media aspect* aggregates the basic media objects of an EMMO, the *semantic aspect* enables the specification of semantic associations between an EMMO's media objects, and finally the *functional aspect* permits EMMOs to define arbitrary, domain-specific operations that can be invoked by applications. Moreover, by providing *versioning* support, EMMOs can be modified concurrently within a distributed environment. As all three aspects of multimedia content and the versioning information can be bundled into one unit and serialized into an exchangeable format, EMMOs establish *tradeable*, semantically enriched units of multimedia content.

In contrast to common approaches for the representation of multimedia content, as well as existing standards for modeling the content's semantics, EMMOs establish a unique way for the semantic modeling of multimedia content. Popular standards for *multimedia document models*, such as HTML [1], XHTML+SMIL [13], HyTime [14], MHEG-5 [15], MPEG-4 BIFS and XMT [16], SMIL [2], or SVG [3], model the presentation of content by arranging basic media objects according to temporal, spatial, and interaction relationships. Therefore, they mainly cover the content's media aspect, but disregard the semantic and functional aspects of content, and provide no versioning support. Standards for *modeling semantics*, such as RDF [17, 18], Topic Maps [19], MPEG-7 (especially MPEG-7's `Graph` tools for the description of content semantics [20]), or Conceptual Graphs [21], clearly provide means for describing the semantic aspect of content. However, they rather neglect the media aspect and functional aspect, and also do not provide versioning support.

Within the CULTOS project, a distributed infrastructure of *EMMO containers* [22] and an *authoring tool* for the creation of EMMOs were developed. For the realization of advanced operations on EMMOs, efficient retrieval and processing of the information captured by EMMOs was still missing after the completion of the CULTOS project. Therefore, we have developed the query algebra EMMA, which provides a formal basis for querying EMMOs.

[1] CULTOS was carried out from 2001 to 2003 by partners from 11 EU countries and Israel. It 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.

The contribution of this paper is to introduce the formal foundation of the query algebra EMMA. The paper builds on, revises, and extends previous research work published in [12], [23], and [24]. By addressing an EMMO's media, semantic, and functional aspect, as well as its versioning information, EMMA is adequate and complete with regard to the EMMO model. EMMA comprises an extensive set of simple and orthogonal query operators (extraction operators, navigational operators, selection predicates, constructors, and a join operator), which allow the construction of more complex queries against EMMOs, thus providing the basis for efficient query rewriting and optimization.

The remainder of the paper is organized as follows. Section 2 introduces and formally defines the EMMO model, Sect. 3 discusses the requirements of a query algebra for EMMOs. Section 4 takes a look at related approaches and Sect. 5 introduces a representative selection of EMMA's formal foundation along with illustrative examples. Section 6 briefly introduces the EMMO and EMMA implementation and, finally, Sect. 7 concludes this paper and gives an outlook on future work.

2 The EMMO Model

As mentioned before, EMMOs establish tradable, knowledge-enriched units of multimedia content that indivisibly combine the content's media, semantic, and functional aspect, as well as its versioning information into one single object.

The formal components of the EMMO model are *entities*, which occur in four different kinds – *logical media parts* representing media objects or parts of media objects, *ontology objects* representing concepts of an ontology, *associations* modeling binary relationships between entities, and *EMMOs* establishing an aggregation of semantically related entities.

In the following subsections, we formally define entities and their four specializations and use real-world example EMMOs originating from the CULTOS project (see [25]) to illustrate the EMMO model. Moreover, to exemplify the integration of domain knowledge, we use an extended version of the Ontology of Intertextual Studies [26].

2.1 Entities

Each entity w is characterized by thirteen properties:

- Each entity w has a *global* and *unique object identifier (OID)* o_w represented as universal unique identifier (UUID) [27], which enables the unique identification of entities in distributed scenarios.
- As UUIDs are not really useful for humans, each entity w has also a human readable *name* n_w expressed as string value.
- For classifying whether an entity w is a logical media part, an ontology object, an association, or an EMMO, its *kind* k_w is specified accordingly.

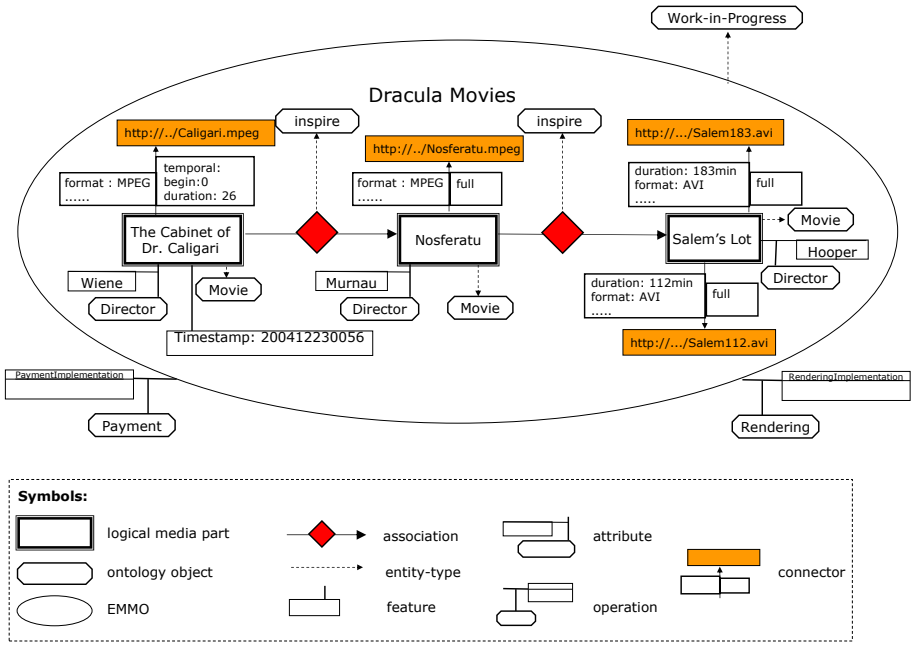

Fig. 1. EMMO "Dracula Movies" (e_{movies})

- Each entity w is described by a set of *types* T_w, i.e. a set of ontology objects, enabling the classification by concepts taken from a domain ontology, e.g. entity w might be an instantiation of the concepts "Ancient Text" and "Novel", or an instantiation of the concept "Movie". The semantics of ontology objects is specified in the underlying domain ontology, e.g. within the ontology structure represented in Fig. 2.
- Each entity possesses an arbitrary number of application-dependent *attributes* A_w. Attributes are represented as attribute-value pairs with the attribute name being a concept of a domain ontology, e.g. by attaching the value "Murnau" for the attribute "Director" to the entity representing the movie "Nosferatu" (see Fig. 1), one can express that the movie was directed by Murnau. The attribute value is per default untyped, however, typing constraints can be introduced via the domain ontology (see Fig. 2).
- For providing versioning support, a set of *preceding versions* P_w and *succeeding versions* S_w can be assigned to each entity w. Each version of w is again an entity of the same kind k_w. By also treating an entity's versions as entities, different versions of an entity can be interrelated just like any other entities, thus allowing one to establish relationships between entity versions. Figure 3 shows several versions of the EMMO "Dracula Movies" and their interrelationships.

– As it might be necessary in an implementation of the model, to augment an entity w with further low-level data, such as timestamps or status information, in a flexible, ad-hoc manner, a set of *features* F_w, represented as feature-value pairs, can be attached to the entity. In contrast to attributes, feature names are not ontology objects but simple strings.

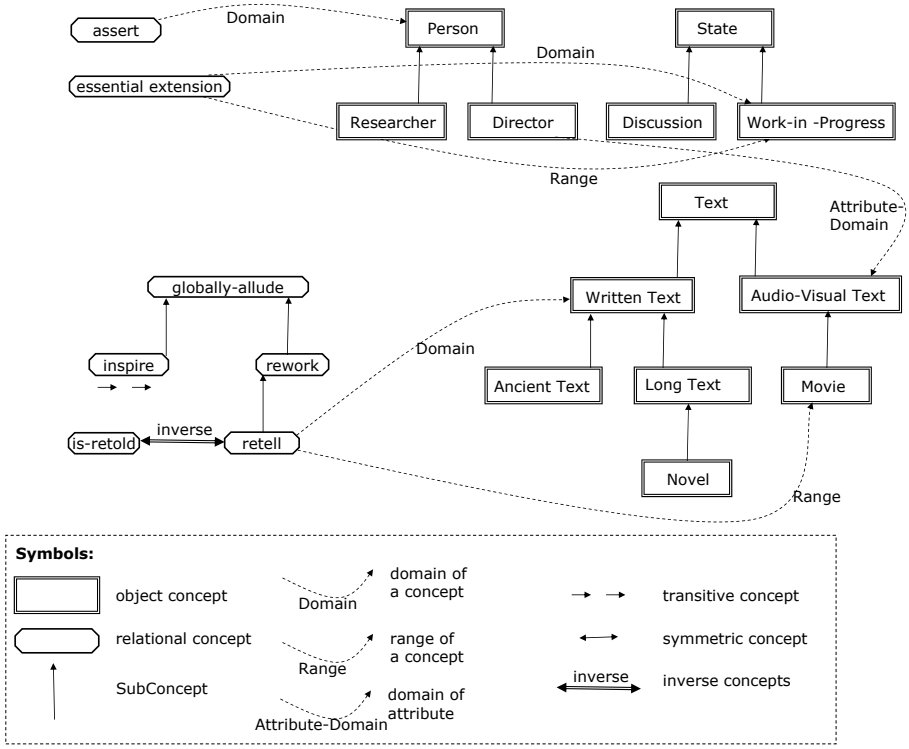

Fig. 2. Graphical representation of a part of an Ontology of Intertextual Studies

As the remaining properties of an entity w are only relevant for certain kinds of entities, at this point we will only provide a brief explanation as far as it is necessary for the understanding of the following definitions; we will provide more detailed definitions and examples in the following subsections.

– By specifying exactly one *source* and *target entity* s_w and t_w, an *association* establishes a directed binary relationship between those entities.
– The *connectors* C_w establish a connection to the physical media data of a *logical media part*. Each connector consists of a *media profile* which describes the storage location by either embedding the *raw media data* or by referencing the media data via a *URI*, and of a *media selector*, which provides means to address only selected parts of the media object.

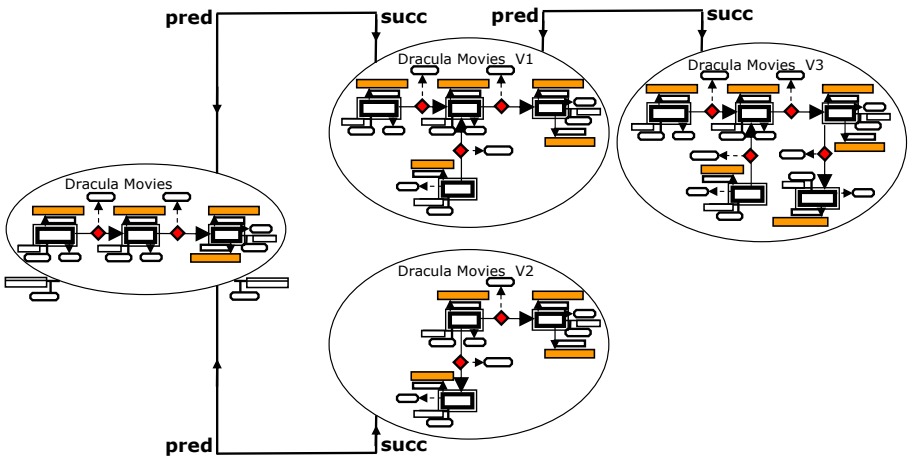

Fig. 3. The versioning information of EMMO "Dracula Movies"

- An *EMMO* constitutes a container of all entities specified in the set *nodes* N_w.
- An *EMMO* offers *operations* O_w, which can be invoked by external applications. The implementation of an operation is described by a mathematical *function*.

After this informal intuitive description, we are now ready to provide a formal definition of an entity. First we define some basic symbols we will use throughout the rest of this paper.

Definition 1. *[Symbols] Let* Γ *denote the set of all logical media parts,* Θ *the set of all ontology objects,* Λ *the set of all associations,* Σ *the set of all Emmos, and* $\Omega = \Gamma \cup \Theta \cup \Lambda \cup \Sigma$ *the set of all entities. Further, let* $\mathcal{MS}$ *be the set of all media selectors,* $\mathcal{MP}$ *the set of all media profiles,* $\mathcal{OP}$ *the set of all operations. Finally, let* $\mathbb{VAL}$ *be the set of all untyped data values,* $\mathbb{UUID} \subset \mathbb{VAL}$ *the set of all universal unique identifiers,* $\mathbb{STR} \subset \mathbb{VAL}$ *the set of all strings,* $\mathbb{URI} \subset \mathbb{STR}$ *the set of all uniform resource identifiers,* $\mathbb{RMD}$ *the set of all raw media data, and* $\mathbb{FUN}$ *the set of all functions.*

On the basis of these common symbols, we define entities as follows.

Definition 2. *[Entity] An entity* $w \in \Omega$ *is a thirteen-tuple* $w = (o_w, n_w, k_w, s_w, t_w, T_w, A_w, C_w, N_w, P_w, S_w, F_w, O_w)$*, where* $o_w \in \mathbb{UUID}$ *denotes the unique object identifier (OID) of* w*,* $n_w \in \mathbb{STR}$ *the name of* w*,* $k_w \in \{$"*lmp*", "*ont*", "*asso*", "*emm*"$\}$ *the kind of* w*,* $s_w \in \Omega \cup \{\varepsilon\}$ *the source and* $t_w \in \Omega \cup \{\varepsilon\}$ *the target entity of* w *with* $\varepsilon \notin \Omega$ *stating that such an entity is undefined,* $A_w \subseteq \Theta \times \mathbb{VAL}$ *the attributes,* $T_w \subseteq \Theta$ *the types,* $C_w \subseteq \mathcal{MS} \times \mathcal{MP}$ *the connectors,* $N_w \subseteq \Omega$ *the nodes,* $P_w \subseteq \Omega$ *the predecessors,* $S_w \subseteq \Omega$ *the*

successors, $F_w \subseteq \mathbb{STR} \times \mathbb{VAL}$ *the features, and* $O_w \subseteq \mathcal{OP}$ *the operations of* w*. The following constraints hold for all entities:*

$$\forall w_1, w_2 \in \Omega : o_{w_1} = o_{w_2} \longrightarrow w_1 = w_2 \tag{1}$$

$$\forall w, v \in \Omega : v \in P_w \vee v \in S_w \longrightarrow k_w = k_v \tag{2}$$

Constraint (1) enforces that each entity has a unique identifer and Constraint (2) assures that each version of w is again an entity of the same kind k_w.

2.2 Logical Media Parts

Logical media parts are entities which enable the representation of media objects or parts of media objects at a logical level, and thus address an EMMO's *media aspect*. By decoupling the logical media part from any existing physical representation, a person who is not owing a media object can still use it within an EMMO. To express the difference between, for example, the movie "Salem's Lot" directed by Tobe Hooper and its underlying source material, the novel "Salem's Lot" written by Stephen King, the movie and the novel are modeled as two different logical media parts.

Definition 3. *[Logical media part] A logical media part* $l \in \Gamma$ *is an entity with* $k_l = "lmp" \wedge s_l = t_l = \varepsilon \wedge N_l = O_l = \emptyset$*.*

By means of *connectors* C_w, logical media parts not only model media objects at a logical level but additionally maintain connections to physical media data representing these objects, and thus provide the media aspect of multimedia content represented within the EMMO model. Connectors (see Def. 2) consist of a *media profile* representing the physical media data and of a *media selector* addressing the parts of the media data represented by the profile according to textual, spatial, and temporal criteria.

As formally defined in Def. 4, a media profile combines the storage location, which is called – following the MPEG-7 terminology – *media instance*, with low-level *metadata*, such as storage format or file size. The *media instance* either directly embeds media data or – if embedding is not feasible, e.g. because the media data is a live stream – references media data via a URI.

Definition 4. *[Media profile] A media profile* $mp = (i_{mp}, M_{mp}) \in \mathcal{MP}$ *is described by its media instance* $i_{mp} \in \mathbb{URI} \cup \mathbb{RMD}$ *and its metadata* $M_{mp} \subseteq \mathbb{STR} \times \mathbb{VAL}$*.*

Media selectors (see Def. 5) render it possible to address only selected parts of the physical media data, such as the introductory section of a movie from the first until the 26th minute, without having to extract that part, for instance, by putting the scene into a separate file using an audio editing tool.

Definition 5. *[Media selector] A media selector* $ms = (k_{ms}, P_{ms}) \in \mathcal{MS}$ *is described by its kind* $k_{ms} \in \{"spatial", "textual", "temporal", "full"\}$ *and by its parameters* $P_{ms} \subseteq \mathbb{STR} \times \mathbb{VAL}$*.*

In Example 1 we illustrate how the three logical media parts depicted in Fig. 1 representing the media objects "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot" can be formally described within the EMMO model. The symbols $l_{caligari}$, $l_{nosferatu}$, and l_{salem} represent the three logical media parts. For example, the thirteen-tuple $l_{caligari}$ indicates that there exists an entity which is uniquely identified by the OID "l2471", is named "The Cabinet of Dr. Caligari", is of kind logical media part ("lmp"), specifies no source and target entity, is classified as "Movie", has the value "Wiene" for the attribute "Director", describes its physical media data by the connector (ms_1, mp_1), is augmented by its timestamp information, and specifies its sets of nodes, predecessors, successors, and operations as empty. The connector (ms_1, mp_1) references the temporal selection of the first 26 minutes from the MPEG-movie "`Caligari.mpeg`". (ms_2, mp_2) represents the connector of the logical media part $l_{nosferatu}$ associating the complete MPEG-movie "Nosferatu.mpeg", and, finally, (ms_3, mp_3) and (ms_4, mp_4) represent two versions of different length of the movie "Salem's Lot".

Example 1

$$
\begin{aligned}
l_{caligari} =& (\text{“}l2471\text{”}, \text{“The Cabinet of Dr. Caligari”}, \text{“lmp”}, \epsilon, \epsilon, \{o_{movie}\}, \{(o_{director}, \text{“Wiene”})\},\\
& \{(ms_1, mp_1)\}, \emptyset, \emptyset, \emptyset, \{(\text{“timestamp”}, \text{“200412230056”})\}, \emptyset),\\
l_{nosferatu} =& (\text{“}l9462\text{”}, \text{“Nosferatu”}, \text{“lmp”}, \epsilon, \epsilon, \{o_{movie}\}, \{(o_{director}, \text{“Murnau”})\},\\
& \{(ms_2, mp_2)\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
l_{salem} =& (\text{“}l6231\text{”}, \text{“Salem's Lot”}, \text{“lmp”}, \epsilon, \epsilon, \{o_{movie}\}, \{(o_{director}, \text{“Hooper”})\},\\
& \{(ms_3, mp_3), (ms_4, mp_4)\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
ms_1 =& (\text{“temporal”}, \{(\text{“begin”}, 0), (\text{“duration”}, 26)\}),\\
mp_1 =& (\text{“}\texttt{www.../Caligari.mpeg}\text{”}, \{(\text{“format”}, \text{“MPEG”})\}),\\
ms_2 =& (\text{“full”}, \emptyset),\\
mp_2 =& (\text{“}\texttt{www.../Nosferatu.mpeg}\text{”}, \{(\text{“format”}, \text{“MPEG”})\}),\\
ms_3 =& (\text{“full”}, \emptyset),\\
mp_3 =& (\text{“}\texttt{www.../Salem183.avi}\text{”}, \{(\text{“format”}, \text{“AVI”}), (\text{“duration”}, 183)\}),\\
ms_4 =& (\text{“full”}, \emptyset),\\
mp_4 =& (\text{“}\texttt{www.../Salem112.avi}\text{”}, \{(\text{“format”}, \text{“AVI”}), (\text{“duration”}, 112)\}).
\end{aligned}
$$

2.3 Ontology Objects

Ontology objects are entities that represent concepts of an ontology. By providing the basis for the description of entities and other properties by concepts taken from an ontology, ontology objects contribute to the *semantic aspect* of multimedia content modeling. Within the EMMO model, ontology objects are applied in four different ways, i.e. they are used:

- for designating the types of entities,
- for designating the attributes of attribute values,
- for designating the operations attached to EMMOs (see Def. 9),

– as nodes within the EMMO knowledge structure (see Sect. 2.5).

Definition 6. *[Ontology object] An ontology object* $o \in \Theta$ *is an entity with* $k_o = "ont" \wedge s_o = t_o = \varepsilon \wedge C_o = N_o = O_o = \emptyset$.

As can be seen from Def. 6, the types T_o of an ontology object o can be a non-empty set, i.e. ontology objects can again be classified by other ontology objects. This provides the basis for expressing ontological structures within the EMMO model. The development of a dedicated ontology engineering environment is focus of future work (see [28], [29], [30]). The final aim is the seamless integration of ontological knowledge into the EMMO model.

In Example 2 all four different ways of using ontology objects are illustrated: Within the EMMOs "Dracula Studies" and "Dracula Research"(see Fig. 4 and Fig. 5), ontology object $o_{inspire}$ represents the type of the association connecting the two logical media parts "Vampyre" and "Dracula", ontology object o_{movie} the type of the logical media part "Nosferatu"; ontology object $o_{director}$ is used as name of the attribute attached to the logical media part "Nosferatu", and ontology object $o_{payment}$ represents the designator of the operation provided by EMMO "Dracula Studies". Moreover, the ontology object o_{miller}, which represents the concept "Elizabeth Miller", is specified as node contained within EMMO "Dracula Research", and by additionally typing this ontology object with the ontology object $o_{researcher}$, "Elizabeth Miller" is classified as "Researcher".

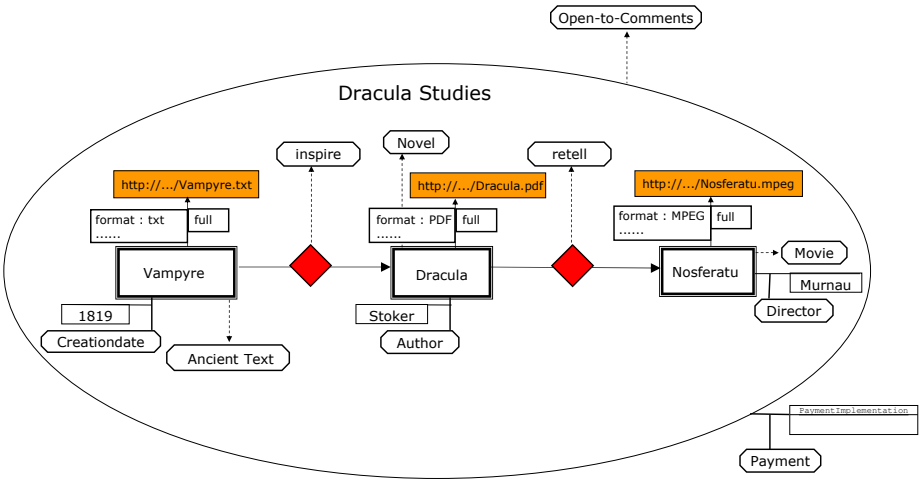

Fig. 4. EMMO "Dracula Studies"($e_{studies}$)

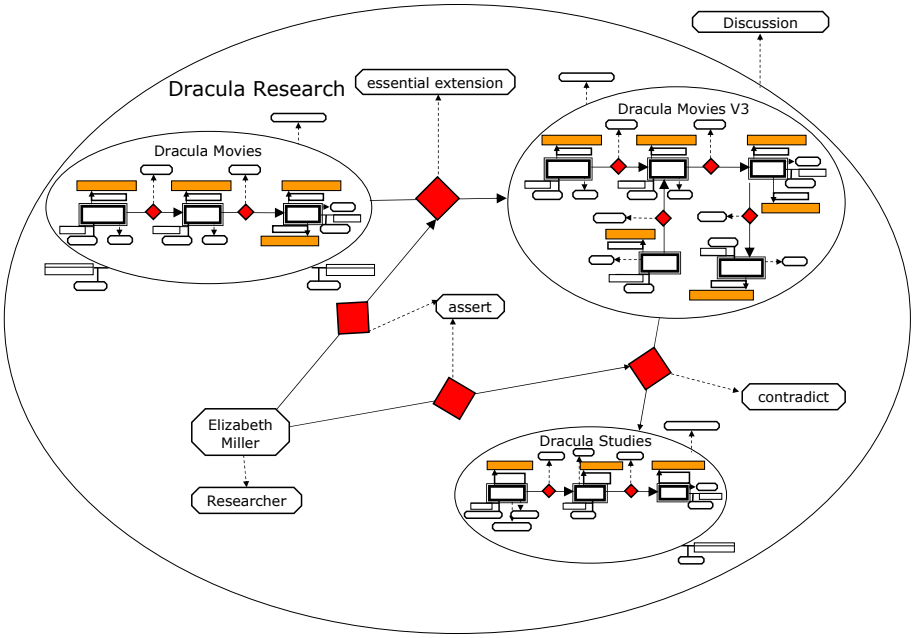

Fig. 5. EMMO "Dracula Research"($e_{research}$)

Example 2

$$
\begin{aligned}
o_{inspire} &= (\text{“}o8421\text{”}, \text{“inspire”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{movie} &= (\text{“}o4302\text{”}, \text{“Movie”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{director} &= (\text{“}o3418\text{”}, \text{“Director”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{payment} &= (\text{“}o6445\text{”}, \text{“Payment”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{miller} &= (\text{“}o3021\text{”}, \text{“Elizabeth Miller”}, \text{“ont”}, \epsilon, \epsilon, \{o_{researcher}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
o_{researcher} &= (\text{“}o2166\text{”}, \text{“Researcher”}, \text{“ont”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}
$$

2.4 Associations

Associations describe binary directed semantic relationships between entities. Thus, they contribute to the *semantic aspect* of multimedia content. By being modeled as entities, associations can take part in other associations, and thus facilitate the *reification* of statements in the EMMO model.

Definition 7. *[Association] An association $a \in \Lambda$ is an entity with $k_a = "asso" \wedge s_a \neq \varepsilon \wedge t_a \neq \varepsilon \wedge C_a = N_a = O_a = \emptyset \wedge |T_a| = 1$.*

Similar to other entities, an association's type is represented by an ontology object and determines the kind of semantic relationship. Different from other entities, however, an association can only associate one type because it is supposed to represent only a unique kind of relationship. By specifying exactly one source and one target entity s_a and t_a, each association establishes a directed binary relationship between those two entities.

Example 3 shows the formal description of the two associations $a_{ca \to no}$ and $a_{no \to sa}$ contained within EMMO "Dracula Movies"(Fig. 1) and of the four associations $a_{mo \to moV3}$, $a_{moV3 \to st}$, $a_{mi \to (mo \to moV3)}$, and $a_{mi \to (moV3 \to st)}$ contained within EMMO "Dracula Research"(Fig. 5). Association $a_{mo \to moV3}$ models that EMMO "Dracula Movies V3" is an essential extension of EMMO "Dracula Movies", and by expressing that this statement was asserted by "Elizabeth Miller", association $a_{mi \to (mo \to moV3)}$ exemplifies the reification of statements.

Example 3

$$
\begin{aligned}
a_{ca \to no} &= (\text{“}a0225\text{”}, \text{“}ca \to no\text{”}, \text{“asso”}, l_{caligari}, l_{nosferatu}, \{o_{inspire}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{no \to sa} &= (\text{“}a5461\text{”}, \text{“}no \to sa\text{”}, \text{“asso”}, l_{nosferatu}, l_{salem}, \{o_{inspire}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{mo \to moV3} &= (\text{“}a6390\text{”}, \text{“}mo \to moV3\text{”}, \text{“asso”}, e_{movies}, e_{moviesV3}, \{o_{essential\text{-}extension}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{moV3 \to st} &= (\text{“}a5461\text{”}, \text{“}moV3 \to st\text{”}, \text{“asso”}, e_{moviesV3}, e_{studies}, \{o_{contradict}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{mi \to (mo \to moV3)} &= (\text{“}a4771\text{”}, \text{“}mi \to (mo \to moV3)\text{”}, \text{“asso”}, o_{miller}, a_{mo \to moV3}, \{o_{assert}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{mi \to (moV3 \to st)} &= (\text{“}a7031\text{”}, \text{“}mi \to (moV3 \to st)\text{”}, \text{“asso”}, o_{miller}, a_{moV3 \to st}, \{o_{assert}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}
$$

2.5 EMMOs

An EMMO constitutes the core component of our model. It is a container that combines several entities into a single unit. By aggregating media data (i.e. logical media parts) and enriching this media data by semantic data (i.e. associations and ontology objects), an EMMO addresses the *media* and *semantic aspect* of multimedia content modeling. For instance, EMMO "Dracula Movies" groups the semantic descriptions of the logical media parts "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot" into one single unit. Since EMMOs are modeled as entities, EMMOs can be contained within other EMMOs, just as any other entity. Therefore, a structure of hierarchically nested EMMOs can be established: EMMO "Dracula Research" in Fig. 5, for example, contains the EMMOs "Dracula Movies", "Dracula Movies V3", and "Dracula Studies". Furthermore, an EMMO can also take part in associations, facilitating the representation of knowledge about the EMMO. For instance, within EMMO "Dracula Research" it is stated that EMMO "Dracula Movies V3" contradicts EMMO "Dracula Studies". Finally, by specifying operations that process its content, EMMOs address the *functional aspect* of multimedia content.

Definition 8. *[EMMO] An EMMO $e \in \Sigma$ is an entity with $k_e =$ "emm", and $s_e = t_e = \varepsilon \wedge C_e = \emptyset$, such that*

$$\forall x \in N_e : \quad k_x = \text{"}asso\text{"} \longrightarrow \{s_x, t_x\} \subseteq N_e \tag{3}$$

According to this definition, an EMMO e constitutes a container of other entities because its set of nodes N_e is not restricted to an empty set, as it is the case with other kinds of entities. The contained entities form a connected graph structure when they are interlinked by associations. Constraint 3 ensures that associations can specify only those entities as source or target entity which already belong to the EMMO's nodes, and thus, guarantees that any established relationship is fully contained within the EMMO.

A further difference between EMMOs and the other kinds of entities is that its set of *operations* is not necessarily empty, allowing an EMMO to associate arbitrary operations. Within the EMMO model, an operation is a tuple combining an ontology object acting as the operation's *designator* with the operation's *implementation*, which can be described by any mathematical function.

Definition 9. *[Operation] An operation* $op = (d_{op}, i_{op}) \in \mathcal{OP}$ *is described by its designator* $d_{op} \in \Theta$ *and its implementation* $i_{op} \in \mathbb{FUN}$.

In Example 4, finally, the three EMMOs "Dracula Movies", "Dracula Studies", and "Dracula Research" are formally described: EMMO "Dracula Movies" consists of five nodes, i.e. the three logical media parts "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot", and two associations; it defines EMMO "Dracula Movies V1" and EMMO "Dracula Movies V2" as its direct successor versions (see Fig. 3), and specifies the functions f_{render} implementing a "rendering" operation and $f_{payment}$ implementing a "payment transaction" operation. EMMO "Dracula Studies" aggregates five entities, i.e. the three logical media parts "Vampyre", "Dracula", and "Nosferatu", as well as two associations, and offers a payment functionality. Finally, EMMO "Dracula Research" consists of eight nodes, i.e. the EMMOs "Dracula Movies","Dracula Movies V3", and "Dracula Studies", the ontology object "Elizabeth Miller", and four associations.

Example 4

$$
\begin{aligned}
e_{movies} =& (\text{“}e7921\text{”}, \text{“Dracula Movies”}, \text{“emm”}, \epsilon, \epsilon, \{o_{work\text{-}in\text{-}progress}\}, \emptyset, \emptyset, \{l_{caligari}, l_{nosferatu}, l_{salem},\\
& a_{ca\rightarrow no}, a_{no\rightarrow sa}\}, \emptyset, \{e_{moviesV1}, e_{moviesV2}\}, \emptyset, \{(o_{render}, f_{render}), (o_{payment}, f_{payment})\}),\\
e_{studies} =& (\text{“}e3811\text{”}, \text{“Dracula Studies”}, \text{“emm”}, \epsilon, \epsilon, \{o_{open\text{-}to\text{-}discussion}\}, \emptyset, \emptyset,\\
& \{l_{vampyre}, l_{dracula}, l_{nosferatu}, a_{va\rightarrow dr}, a_{dr\rightarrow no}\}, \emptyset, \emptyset, \emptyset, \{(o_{payment}, f_{payment})\}),\\
e_{research} =& (\text{“}e1411\text{”}, \text{“Dracula Research”}, \text{“emm”}, \epsilon, \epsilon, \{o_{discussion}\}, \emptyset, \emptyset, \{e_{movies}, e_{moviesV3}, e_{studies},\\
& o_{miller}, a_{mo\rightarrow moV3}, a_{moV3\rightarrow st}, a_{mi\rightarrow(mo\rightarrow moV3)}, a_{mi\rightarrow(moV3\rightarrow st)}\}, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}
$$

The integration of ontology knowledge enables to restrict the usage of associations and attributes, i.e. to define constraints on the knowledge structures within EMMOs. For example, by specifying the ontology represented in Fig. 2 as underlying domain ontology, it is determined that associations of type "retell" describe relationships pointing from entities of type "Written Text" to entities of type "Audio-Visual Text". Within the axioms of the ontology, it can be additionally specified that integrity constraints on associations are extended to subconcepts,

i.e. specifying entities of type "Audio-Visual Text" as permitted target entities then also includes entities of type "Movie" as permitted value. Moreover, within the ontology, one can specify the permitted domain of attributes, i.e. the types of entities to which they can be attached, for instance, the concept "Director" can only be used as attribute for entities of type "Audio-Visual Content".

3 Requirements of a Query Algebra for EMMOs

For the realization of advanced operations on EMMO structures, a formal basis for querying of EMMOs, i.e. an algebra providing a set of formal query operators suitable for the EMMO model, is needed. The EMMO model has no inherent semantics, i.e. the particular semantics of an application scenario implementing the EMMO model is derived from the integrated domain ontology. Therefore, the requirements for accessing the information captured by EMMOs result from structural and syntactical issues, and have to be seen as being independent from the semantics of any particular application scenario. In the following, we introduce the essential requirements for such a query algebra.

The most important and fundamental prerequisite of such an algebra is to provide operators for accessing an EMMO's three aspects and its versioning information. Thus, the algebra has to offer operators enabling the access to:

- an EMMO's *media aspect*, i.e. operators that give access to logical media parts and their connectors;
- an EMMO's *semantic aspect*, i.e. operators that facilitate the retrieval of all kinds of entities contained in an EMMO, the querying of the types of entities and their attribute values, as well as the traversal of associations between entities; 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;
- an EMMO's *functional aspect*, i.e. operators that allow the access to and permit the execution of the operations of an EMMO;
- an EMMO's *versioning information*, i.e. operators for the querying of an entity's direct and indirect versions.

In addition, an EMMO query algebra should meet basic query algebra requirements. Its operators should be formally defined with *precise semantics* to provide the basis for query rewriting and optimization. Furthermore, the operators should be *orthogonal* and arbitrarily nestable for enabling the formulation of expressive queries.

To combine information contained within different EMMOs, the algebra should support *joins* between entities. Moreover, a suitable algebra should support some basic construction and manipulation operators, such as union, intersection, and difference. However, since we have a graphical authoring tool available, such construction and manipulation operators can be kept simple.

Finally, because the EMMO model uses 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 should be expressive enough to *integrate ontological knowledge* into a query. Thus, for example, it should be possible to consider supertype/subtype relationships, transitive or inverse associations, etc.

A query algebra which is sufficiently expressive to fulfill all these requirements is said to be *adequate and complete* with regard to the EMMO model.

4 Related Approaches

On the search for a suitable query algebra for EMMOs, we will first take a brief look at object-oriented query approaches and query approaches for semi-structured data and multimedia content, before we analyze query approaches for semantic standards and examine their adequacy and completeness with regard to the EMMO model.

Although object-oriented database systems establish a graph-based data model, the object-oriented data model and the EMMO model are very different from each other, i.e. the former operates on the schema level and gathers objects of a particular type within one class, whereas the EMMO model operates on the instance level, defines no object classes, and leaves the specification of the type semantics to the integrated domain ontology. However, for accessing the information captured by EMMOs, i.e. an EMMO's three aspects and versioning information, one needs an adequate EMMO query approach. Therefore, we did not analyze query languages for object-oriented databases, such as OQL [31], AQUA Algebra [32], or XSQL [33], in further detail.

The Object Exchange Model OEM [34] is the widely accepted data model for semi-structured data. Similar to the EMMO model, OEM is schema-less and describes graph structures. Nevertheless, there are many differences between the two data models. OEM does neither address the three aspects of multimedia content, nor provides versioning support. Therefore, query languages for semi-structured data, such as Lorel [35], UnQL [36], SAL [37], XQuery [38], or XPath [39], are inadequate for querying EMMOs. However, query languages for semi-structured data provide a profound basis for graph navigation by establishing regular path expressions. Thus, we could use those query languages as inspiration for the design of the regular path expressions and navigational operators in EMMA (see Sect. 5.3).

In the literature, several query algebras for multimedia content have been proposed, such as GCalculus/S [40], Algebraic Video [41], or the Multimedia Presentation Algebra (MPA) [42]. These algebras have in common that they mainly address the *media aspect* of multimedia content. They focus on querying the temporal and spatial 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 [17, 18], Topic Maps [19], and MPEG-7 (especially MPEG-7's `Graph` tools for the description of content semantics [20]). 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 neither provide explicit support for versioning nor a hierarchical structuring of resource descriptions; the same is generally true for RDF-based query approaches as well. There are quite a few proposals for RDF query languages, such as RQL [43], SquishQL [44], or RAL [45], which can be used for querying the semantic aspect of multimedia content, but provide no means for querying the media and functional aspect or the versioning information of multimedia content. Thus, those approaches are incomplete and inadequate with regard to the EMMO model.

The situation for Topic Maps is quite similar to RDF. The Topic Map data model focuses on the semantic aspect 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 can be hierarchically nested like EMMOs, they have no explicit versioning support. Consequently, query languages for Topic Maps are also in general incomplete and inadequate with regard to the EMMO model.

Within the context of the ongoing standardization of a Topic Maps Query Language TMQL [46], several approaches, such as Tolog [47], TMPath [48], XTMPath [49], or [50] have been introduced. However, those proposals remain again 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.

Finally, concerning the querying of semantic descriptions of multimedia content on the basis of MPEG-7's `Graph` tools, there are quite a few approaches adapting XQuery for the querying of MPEG-7 media descriptions (see [51]), but these approaches do not provide specific operators that would allow a reasonable processing of the `Graph` tools.

To summarize, looking at related work, 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, as well as for querying the semantic aspect of multimedia content, such as the RDF-based RAL, those algebras are neither adequate nor complete with regard to the EMMO model, which addresses the media, semantic, and the functional aspects of multimedia content, as well as versioning support.

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

5 EMMA Query Algebra

The design of the EMMO query algebra EMMA was in the first place driven by the requirement for accessing the complete information stored within an EMMO, i.e. the access to the three aspects of an 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.

There are five general classes of EMMA's query operators: the *extraction operators* provide the basis for querying an EMMO's three aspects, as well as its versioning information. The *navigational operators* enable the navigation along an EMMO's semantic graph structure and provide means for the integration of ontological knowledge. The *selection predicates* facilitate the selection of only those entities satisfying a specific condition, and the *constructors* enable the modification, combination, and creation of new EMMOs. Finally, the *join operator* relates several entities or EMMOs with a join condition.

Before we present the formal basis of the five operator classes in the following subsections, we will provide some definitions required for the understanding of the definitions to follow. To guarantee the readability of the paper, we will introduce the most representative EMMA operators by giving their formal definitions accompanied with illustrative real-world example queries originating from the CULTOS project. We will omit any EMMA operator which is used for accessing only some very specific aspects and information captured by the EMMO model. The complete list of formal definitions of EMMA operators can be found in [52].

To conclude this section, we will explain in a summary subsection how these operators contribute to fulfil the requirements for an EMMO query algebra.

5.1 Basic Definitions

The input and output values of EMMA operators, i.e. their signatures, are described by *sets* and *sequences*.

Definition 10. *[Set and Sequence] Let* IN *denote the set of all natural numbers,* I *an arbitrary index set,* $\mathbb{BOO} = \{true, false\}$ *the Boolean set, and* $\mathbb{SET}$ *the set of all sets. Let* A *and* B *be arbitrary sets, then* $\mathcal{P}(A) = \{x \mid x \subseteq A\}$ *denotes the powerset of* A *and* $A \times B := \{(x, y) \mid x \in A \wedge y \in B\}$ *the Cartesian product over* A *and* B*. The elements of a Cartesian product are called sequences or tuples.* $\mathbb{SEQ}$ *denotes the set of all sequences. A sequence of length* 1 *is equal to its single entry element, i.e.* $\forall x\,(x) = x$*. Let* $j \in I$ *then* $\pi_j : \prod_{i \in I} A_i \longrightarrow A_j$ *with* $\pi_j(a_1, a_2, \ldots, a_n) = a_j$ *denotes the* j^{th} *projection of* $\prod_{i \in I} A_i$*.*

EMMA operators are either *functions* or *predicates*.

Definition 11. *[Function and Predicate] Let* $A, B \in \mathbb{SET}$ *and* $f \in \mathbb{FUN}$ *with* $f : A \longrightarrow B$ *be a function, then* $\mathcal{D}(f) = A$ *denotes the domain and* $\mathcal{R}(f) = B$ *the range of function* f*,* $\mathbb{FUN}_A$ *the set of all functions with* $\mathcal{D}(f) = A$*, and*

$\mathbb{FUN}_{[A,B]}$ *the set of all functions with* $\mathcal{D}(f) = A$ *and* $\mathcal{R}(f) = B$*. Furthermore,* $p \in \mathbb{FUN}_{[A,\mathbb{BOO}]}$ *denotes a predicate,* $\mathbb{PRE}_A = \mathbb{FUN}_{[A,\mathbb{BOO}]}$ *the set of all predicates with domain* A*, and* $\mathbb{PRE} = \{\mathbb{PRE}_A \mid A \in \mathbb{SET}\}$ *the set of all predicates. Let* $f \in \mathbb{FUN}_{\prod_{i\in I} A_i}, j \in I, x \in A_j$ *and* $(a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) \in \prod_{i\in I\setminus\{j\}} A_i$*, then the function* $f_{[a_1,\ldots,a_{j-1},\$,a_{j+1},\ldots,a_n]} : A_j \longrightarrow \mathbb{SET}$ *with* $f_{[a_1,\ldots,a_{j-1},\$,a_{j+1},\ldots,a_n]}(x) = f(a_1, \ldots, a_{j-1}, x, a_{j+1}, \ldots, a_n)$ *is called* f*-projection onto* A_j*.*

EMMA operators are designed to be modular and simple. By using modular EMMA operators in combination with the operators *Apply* and *Elements*, more complex EMMA operators can be defined, and complex queries can be formulated. The operator *Apply* takes a function and a set as input values and returns the set consisting of all return values of the specified function being applied to each element in the specified set.

Definition 12. *[Apply] Let* $A \in \mathbb{SET}$ *and* $f \in \mathbb{FUN}$*, then the operator* $Apply : \mathbb{FUN} \times \mathbb{SET} \longrightarrow \mathbb{SET}$ *is defined as* $Apply(f, A) = \{f(x) \mid x \in A \cap \mathcal{D}(f)\}$.

The operator *Elements* is used to flatten data returned by other operations, e.g. for the specified input set it returns all elements being contained in at least one element of the specified set.

Definition 13. *[Elements] Let* $A \in \mathbb{SET}$*, then the operator* $Elements : \mathbb{SET} \longrightarrow \mathbb{SET}$ *is defined as* $Elements(A) = \{x \mid \exists X \in A \wedge x \in X\}$.

Additionally, for enabling the combination and nesting of EMMA operators, their signatures are always specified in the most general way, i.e. their input and output values are specified as set of entities. Thus, operators which only return valid results if applied to specific kinds of entities can still be applied to other kinds of entities yielding an empty result.

5.2 Extraction Operators

The extraction operators render it possible to access the information stored within an EMMO. In the following, we define a representative subset of the 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 maintain connections to their physical representations, i.e. to their media profiles and media selectors. For accessing the information described by a logical media part's connectors, EMMA defines several modular operators, as well as some more complex operators defined by nesting those modular operators. For example, the operator *MediaProfiles* can be used for locating media profiles, i.e. applying the operator *MediaProfiles* to a logical media part returns the union of all its associated media profiles, e.g. (see Fig. 1) the query expression

$$MediaProfiles(l_{salem}) = \{(\texttt{www.../Salem183.avi}, \{(\text{“}duration\text{”}, 183), (\text{“}format\text{”}, \text{“}AVI\text{”})\}),$$
$$(\texttt{www.../Salem112.avi}, \{(\text{“}duration\text{”}, 112), (\text{“}format\text{”}, \text{“}AVI\text{”})\})\}$$

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 operator *MediaProfiles* is defined as a combination of the operators *connectors* and *MediaProfile*. For a specified entity, the operator *connectors* returns its set of connectors, and the operator *MediaProfile* returns the media profile for a given connector. By using the operators *Apply* and *Elements* in its definition, the operator *MediaProfiles* can be used to access the union of associated media profiles of a logical media part.

Definition 14. *[connectors and MediaProfiles] Let* $w \in \Omega$, $ms \in \mathcal{MS}$, *and* $mp \in \mathcal{MP}$, *then the operator* $connectors : \Omega \longrightarrow \mathcal{P}(\mathcal{MS} \times \mathcal{MP})$ *is defined as* $connectors(w) = C_w$, *the operator* $MediaProfile : \mathcal{MS} \times \mathcal{MP} \longrightarrow \mathcal{MP}$ *as* $MediaProfile(ms, mp) = mp$, *and* $MediaProfiles : \Omega \longrightarrow \mathcal{P}(\mathcal{MP})$ *as* $MediaProfiles(w) = Elements(Apply(MediaProfile, connectors(w)))$.

Semantic Aspect By attaching concepts of an ontology to entities, entities get meaning. The operator *types* accesses 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" (see Fig. 1):

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

Definition 15. *[types] Let* $w \in \Omega$, *then the operator* $types : \Omega \longrightarrow \mathcal{P}(\Theta)$ *is defined as* $types(w) = T_w$.

For retrieving the attributes 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 attribute-value pair, i.e. the ontology object "Director" with the string value "Murnau". The operator *attributes* returns the set of associated attribute-value pairs for a given entity.

Definition 16. *[attributes] Let* $w \in \Omega$, *then the operator* $attributes : \Omega \longrightarrow \mathcal{P}(\Theta \times \mathbb{VAL})$ *is defined as* $attributes(w) = A_w$.

EMMOs encapsulate a graph-like knowledge structure of entities. The algebra provides the operator *asso* for accessing all associations representing binary directed semantic relationships between other entities, e.g. the query expression

$$asso(e_{research}) = \{a_{mo \rightarrow moV3}, a_{moV3 \rightarrow st}, a_{mi \rightarrow (mo \rightarrow moV3)}, a_{mi \rightarrow (moV3 \rightarrow st)}\}$$

returns the associations within EMMO "Dracula Research" (Fig. 5).

Definition 17. *[asso] Let* $w \in \Omega$, *then the operator* $asso : \Omega \longrightarrow \mathcal{P}(\Lambda)$ *is defined as* $asso(w) = \{x \mid x \in N_w \cap \Lambda\}$.

As associations are modeled as entities, they belong to an EMMO's set of nodes. The algebra provides the operator *nodes* for accessing all entities contained within an EMMO, e.g. the query expression

$$\begin{aligned} \mathit{nodes}(e_{research}) = \{&e_{movies}, e_{moviesV3}, e_{studies}, o_{miller}, a_{mo\rightarrow moV3}, \\ &a_{moV3\rightarrow st}, a_{mi\rightarrow(mo\rightarrow moV3)}, a_{mi\rightarrow(moV3\rightarrow st)}\} \end{aligned}$$

yields a set consisting of all entities in EMMO "Dracula Research".

Definition 18. *[nodes] Let $w \in \Omega$, then the operator nodes : $\Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $\mathit{nodes}(w) = N_w$.*

As EMMOs are also entities, 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

$$\begin{aligned} \mathit{AllEncEnt}(e_{research}) &= \mathit{nodes}(e_{research}) \cup \mathit{nodes}(e_{movies}) \cup \mathit{nodes}(e_{moviesV3}) \\ &\quad \cup \mathit{nodes}(e_{studies}) = \\ &= \{e_{movies}, e_{moviesV3}, e_{studies}, o_{miller}, a_{mo\rightarrow moV3}, a_{moV3\rightarrow st}, \\ &\quad a_{mi\rightarrow(mo\rightarrow moV3)}, a_{mi\rightarrow(moV3\rightarrow st)}, l_{caligari}, l_{nosferatu}, \\ &\quad l_{salem}, a_{ca\rightarrow no}, a_{no\rightarrow sa}, l_{dracula}, l_{return}, \\ &\quad a_{dr\rightarrow no}, a_{sa\rightarrow re}, l_{vampire}, a_{va\rightarrow dr}\} \end{aligned}$$

unifies the nodes of EMMO "Dracula Research" with the nodes of the EMMOs "Dracula Movies" (Fig. 1), "Dracula Movies V3" (see Fig. 6), and "Dracula Studies" (Fig. 4), because these EMMOs are contained within EMMO "Dracula Research" and contain no further EMMOs themselves.

The operator *AllEncEnt* is defined by means of induction over the natural numbers *IN* and is based on the operator *EncEnt*. We say

- "entity w_1 is *contained* in EMMO w_0 *at first level*", if w_1 belongs to EMMO w_0's nodes,
- "entity w_{n+1} is *contained* in EMMO w_0 at $n{+}1^{th}$*-level*", if there exists a sequence of n EMMOs, i.e. $w_1, \ldots, w_n$, such that for all $k \in \{1, \ldots, n+1\}$ entity w_k belongs to EMMO w_{k-1}'s nodes,
- "w is *recursively contained* or *encapsulated* in EMMO w_0", if there exists a natural number n, such that w is contained in EMMO w_0 at n^{th}-level.

To provide a basis for the combination with other EMMA operators, the operators *AllEncEnt* and *EncEnt* both take entities as input value, but only return a reasonable result, if the input entity is of kind EMMO. In all other cases, the empty set is returned. In this way, the operator *EncEnt* takes an EMMO e and a natural number n as input, and returns the nodes of EMMO e at n^{th} level. By defining a unification over the operator *EncEnt*, the operator *AllEncEnt* returns, for a specified EMMO, the set of all its recursively contained entities.

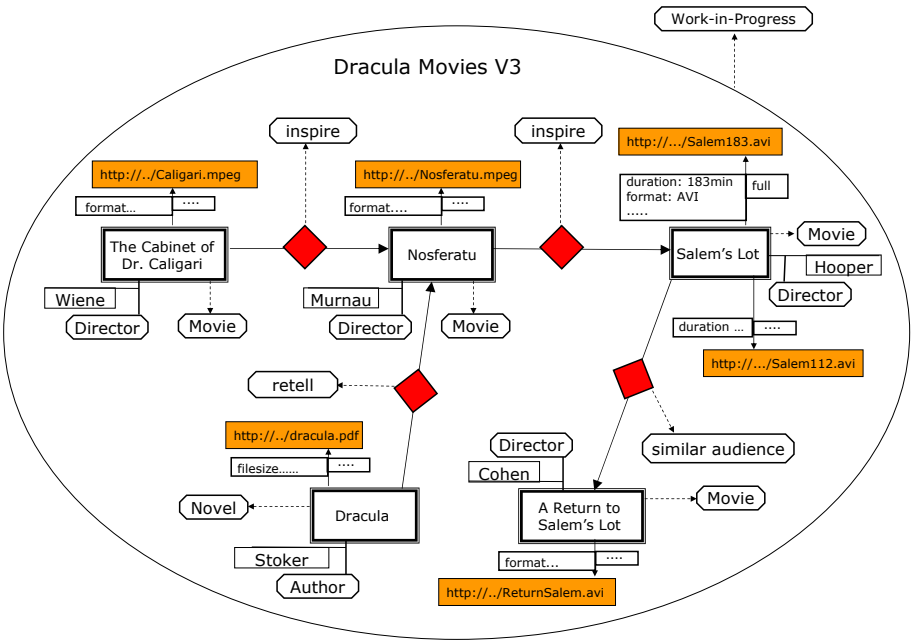

Fig. 6. EMMO "Dracula Movies V3" ($e_{moviesV3}$)

Definition 19. *[AllEncEnt] Let $e \in \Omega$, then the operator $EncEnt : \Omega \times IN \longrightarrow \mathcal{P}(\Omega)$ is defined inductively over IN as follows: $EncEnt(e,1) = N_e$, and by assuming $EncEnt(e,n)$ is defined, one defines $EncEnt(e,n+1) = \{x \in \Omega \mid \exists y \in EncEnt(e,n) \cap \Sigma \ \wedge \ x \in N_y\}$. The operator $AllEncEnt : \Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $AllEncEnt(e) = \bigcup_{i \geq 1} EncEnt(e,i)$.*

Functional Aspect EMMOs offer functions for dealing with their content. For the execution of an EMMO's functionality, the query algebra EMMA specifies the operator *Execute*. Applying the operator *Execute* to EMMO "Dracula Movies" (Fig. 1), the ontology object "rendering", and the parameter `HTML`, i.e.

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

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. 7.

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

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

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. 7.

```
<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. 7. DraculaMovies.html and DraculaMovies.smil

The operator *Execute* takes an EMMO, a function, and a sequence of parameters as input values and returns the result value of the execution of the function with the specified EMMO and sequence of parameters as input values. If the operator *Execute* is applied to an entity which is not of kind EMMO, or the specified operation does not belong to the operations of the specified EMMO, or the sequence of parameters constitutes no valid input value for the specified operation, the empty set is returned.

Definition 20. *[Execute] Let* $e \in \Omega$ *,* $op \in \mathcal{O}P$*, and* $s \in \mathbb{SEQ}$ *, then the operator* $Execute : \Omega \times \mathcal{O}P \times \mathbb{SEQ} \longrightarrow \mathbb{SET}$ *is defined as*

$$Execute(e, op, s) = \begin{cases} \pi_2(op)(e,s) & if \quad op \in O_e \wedge (e,s) \in D(\pi_2(op)) \\ \emptyset & else \end{cases}$$

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

$$successors(e_{movies}) = \{e_{moviesV1}, e_{moviesV2}\}$$

returns EMMO "Dracula Movies V1" and "Dracula Movies V2", the two direct successor versions of EMMO "Dracula Movies" (see Fig. 3). For accessing all succeeding versions, the operator *AllSuccessors* is applied, e.g.

$$AllSuccessors(e_{movies}) = \{e_{moviesV1}, e_{moviesV2}, e_{moviesV3}\}.$$

The operator *successors* retrieves all direct successors of the specified entity.

Definition 21. *[successors] Let* $w \in \Omega$ *, then the operator* $successors : \Omega \longrightarrow \mathcal{P}(\Omega)$ *is defined as* $successors(w) = S_w$.

The operator *AllSuccessors* is defined by means of induction over the natural numbers *IN* and returns the set of all successors for a specified entity. The operator's definition is based on the operator *successors* serving as initial step of the induction and on the operator *Successors* which returns the set of all n^{th}-successors for a specified entity and natural number n. An entity w' is called n^{th} successor of entity w, if there exists a sequence of *n-1* entities, with each entity in the sequence representing the direct successor of its subsequent entity.

Definition 22. *[AllSuccessors] Let* $w \in \Omega$ *, then the operator* $Successors : \Omega \times IN \longrightarrow \mathcal{P}(\Omega)$ *is defined by induction over IN as follows:* $Successors(w, 1) = successors(w)$*, and by assuming* $Successors(w, n)$ *is defined, one defines* $Successors(w, n+1) = \{x \in \Omega \mid \exists y \in Successors(w, n) \wedge x \in successors(y)\}$*, and the operator* $AllSuccessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ *as* $AllSuccessors(w) = \bigcup_{i \geq 1} Successors(w, i)$*.*

For the access to an entity's preceding versions, EMMA also provides the operators *predecessors*, *Predecessors*, and *AllPredecessors*, which are defined analogously.

5.3 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. The basic building blocks of regular path expressions are ontology objects which can be modified and combined using conventional regular expression operators.

Definition 23. *[Regular path expression] Given a symbol set* $S = \{\varepsilon, \_, +, *, ?, |, -, (,)\}$*, an alphabet* $\Psi = \Theta \cup S$*, and* Ψ^**, the set of words over* Ψ *(finite strings over elements of* Ψ*). Then, we define* $\mathbb{REG} \subseteq \Psi^*$ *as the smallest set with the following properties:*

(1) $\forall o \in \Theta : o \in \mathbb{REG}$,
(2) $\varepsilon \in \mathbb{REG}$,
(3) $\_ \in \mathbb{REG}$,
(4) $\forall r_1, r_2 \in \mathbb{REG} : \quad r_1 \,|\, r_2 \in \mathbb{REG}$,
(5) $\forall r_1, r_2 \in \mathbb{REG} : \quad r_1 r_2 \in \mathbb{REG}$,
(6) $\forall r \in \mathbb{REG} : \quad r? \in \mathbb{REG}$,
(7) $\forall r \in \mathbb{REG} : \quad r+ \in \mathbb{REG}$,
(8) $\forall r \in \mathbb{REG} : \quad r* \in \mathbb{REG}$,
(9) $\forall o \in \Theta : \quad o\text{–} \in \mathbb{REG}$,
(10) $\forall r \in \mathbb{REG} : \quad (r) = r$,

and denote $\mathbb{REG}$ *as the set of regular path expressions over ontology objects.*

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, starting 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 V3"(see Fig. 6), 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":

$$\mathit{JumpRight}(e_{moviesV3}, l_{caligari}, o_{inspire}) = \{l_{nosferatu}\}.$$

As already mentioned, 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 a regular path expression specifies an iteration of path expressions, which is interpreted as navigation along the same regular path expression any number of times. Applying the operator *JumpRight* to the same EMMO and starting entity as in the above query, as well as the regular path expression "$o_{inspire}*$" returns three logical media parts representing the movies "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot":

$$\mathit{JumpRight}(e_{moviesV3}, l_{caligari}, o_{inspire}*) = \{l_{caligari}, l_{nosferatu}, l_{salem}\}.$$

Regular path expressions can also be concatenated or defined as optional. For example, applying the operator *JumpRight* to EMMO "Dracula Movies V3", the starting entity "Nosferatu", and the regular path expression "$o_{inspire}o_{similar}?$", yields the logical media parts "Salem's Lot" and "A Return to Salem's Lot":

$$\mathit{JumpRight}(e_{moviesV3}, l_{nosferatu}, o_{inspire}o_{similar}?) = \{l_{salem}, l_{return}\}.$$

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

$$\mathit{JumpRight}(e_{moviesV3}, l_{nosferatu}, o_{inspire} \,|\, o_{retell}) = \{l_{salem}\}.$$

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.

$$\mathit{JumpRight}(e_{moviesV3}, l_{nosferatu}, o_{retell}-) = \{l_{dracula}\}.$$

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

$$\mathit{JumpLeft}(e_{moviesV3}, l_{nosferatu}, o_{retell}) = \mathit{JumpRight}(e_{moviesV3}, l_{nosferatu}, o_{retell}-).$$

Navigational operators provide the basis for the integration of 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 $o_{inspire}$ with the navigation path $o_{inspire}*$ (see example above). Knowledge about inverse association types, such as the association

types "retell" and "is-retold", can be integrated within the queries as well, for instance, by replacing the navigation path $o_{is-retold}$ with the navigation path $o_{is-retold}\,|\,o_{retell}-$, e.g.

$$\mathit{JumpRight}(e_{moviesV3}, l_{nosferatu}, o_{is-retold}\,|\,o_{retell}-) = \{l_{dracula}\}.$$

The operator *JumpRight*, which is formally defined below, takes two entities and one regular path expression as input values. The first input entity – which has to be of type EMMO for the operator *JumpRight* to return a non-empty result value – determines the navigation space, the second entity specifies the starting point of navigation, and the regular path expression describes the set of all possible navigation paths.

Definition 24. *[JumpRight] For $e, w \in \Omega$, and a regular path expression $r \in \mathbb{REG}$, the operator $JumpRight : \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega)$ is defined as follows:*

(1) $\forall r \in \Theta$: $JumpRight(e, w, r) = \{x \in N_e \,|\, \exists y\, y \in asso(e) \wedge r \in types(y) \wedge w = s_y \wedge x = t_y\}$

(2) $r = \varepsilon$: $JumpRight(e, w, \varepsilon) = \{w \,|\, w \in N_e\}$

(3) $r = \_$: $JumpRight(e, w, \_) = \{x \in N_e \,|\, \exists y \in asso(e) \wedge w = s_y \wedge x = t_y\}$

(4) $\forall r_1, r_2 \in \mathbb{REG}$: $JumpRight(e, w, r_1\,|\,r_2) = \bigcup_{x \in \{r_1, r_2\}} JumpRight(e, w, x)$

(5) $\forall r_1, r_2 \in \mathbb{REG}$: $JumpRight(e, w, r_1 r_2) = \bigcup_{x \in JumpRight(e,w,r_1)} JumpRight(e, x, r_2)$

(6) $\forall r \in \mathbb{REG}$: $JumpRight(e, w, r?) = \bigcup_{x \in \{r, \varepsilon\}} JumpRight(e, w, x)$

(7) $\forall r \in \mathbb{REG}$: $JumpRight(e, w, r+) = \bigcup_{n \geq 1} JR_n(e, w, r)$ *with*
$JR_n(e, w, r)$ *defined by induction over* IN :
$JR_1(e, w, r) = JumpRight(e, w, r)$
$JR_n(e, w, r) = \bigcup_{x \in JR_{n-1}(e,w,r)} JumpRight(e, x, r)$

(8) $\forall r \in \mathbb{REG}$: $JumpRight(e, w, r*) = \bigcup_{x \in \{r+, \varepsilon\}} JumpRight(e, w, x)$

(9) $\forall o \in \Theta$: $JumpRight(e, w, o\text{–}) = \{x \in N_e \,|\, \exists y\, y \in asso(e) \wedge o \in types(y) \wedge x = s_y \wedge w = t_y\}$.

The navigational operator *JumpLeft* is defined analogously.

5.4 Selection Predicates

The selection predicates allow the selection of only those entities that satisfy a specific condition. They basically use the result values of extraction operators to create Boolean operators. For instance, applying the operator *IsType* to the logical media part "Dracula" (Fig. 6) and the set containing one ontology object "Book" returns false:

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

By taking a set of ontology objects as second input parameter, the operator *IsType* enables the integration of supertype/subtype relationships within queries.

The ontological knowledge about a subtype relationship, e.g. the subtype relationship between the ontology objects "Novel" and "Book", can be reflected within the query expression, e.g.

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

Assuming that ontological knowledge about supertype/subtype relationships is also represented within EMMOs (e.g. in an EMMO $e_{ontology}$), e.g. by means of associations of type "is_a", the subtypes of "Book" in the previous query could also be calculated dynamically during query execution by using an appropriate *JumpLeft* expression:

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

Although we have not yet developed a language which governs the expression of such ontological knowledge within the EMMO model, the query algebra is sufficiently expressive to be ready for exploiting this knowledge once it becomes available.

Definition 25. *[IsType] Let* $w \in \Omega$ *, and* $O \subseteq \Omega$ *, then the operator* $IsType : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ *is defined as*

$$IsType(w, O) = \begin{cases} true & if \quad \exists o \in O \quad o \in types(w) \\ false & else \end{cases}$$

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, 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 set of all logical media parts contained within EMMO "Dracula Studies" (see Fig. 4), the result set consists of the logical media part representing Stoker's novel "Dracula":

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

Definition 26. *[Select] Let* $A \in \mathbb{SET}$ *and* $p \in \mathbb{PRE}$ *, then let the operator* $Select : \mathbb{PRE} \times \mathbb{SET} \longrightarrow \mathbb{SET}$ *be* $Select(p, A) = \{x \mid x \in A \cap \mathcal{D}(p) \wedge p(x)\}$.

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) associated with a media profile describing media data in AVI format, i.e.

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

yields the logical media part "Salem's Lot" specified by two media profiles which both contain the attribute "format" with value "AVI" in their sets of metadata.

The operator $HasMediaProfileValue$ takes three input parameters, i.e. an entity w, a string value s, and a predicate p, and returns true, if the entity w contains a media profile with a set of metadata including a name-value pair, with the name being equal to s and the value satisfying the condition described by the specified predicate p, e.g. in the above example the predicate *Equal* returns true if its two specified parameters are equal, otherwise false.

Definition 27. *[HasMediaProfileValue] Let* $w \in \Omega$, $s \in \mathbb{STR}$, *and* $p \in \mathbb{PRE}$, *then* $HasMediaProfileValue : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *is defined as*

$$HasMediaProfileValue(w, s, p) = \begin{cases} true & if \quad \exists c \in C_w \\ & \exists k \in Metadata(MediaProfile(c)) \\ & (\pi_1(k) = s \wedge p(\pi_2(k))) \\ false & else \end{cases}$$

5.5 Constructors

EMMA specifies five constructors for EMMOs, i.e. the operators *Difference*, *Union*, *Intersection*, *Nest*, and *Flatten*. All the constructors take at least one EMMO and possibly other parameters as input values, and return exactly one EMMO as output value. 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 to the second EMMO, and additionally, the source and target entities of each association contained within the first EMMO.

Applying the *Difference* operator to the successor EMMO "Dracula Movies V3"(Fig. 6) and the original EMMO "Dracula Movies"(Fig. 1), generates a new EMMO "Newcomers" (see Fig. 8) consisting of the logical media parts describing the movies "Nosferatu", "Salem's Lot", and "A Return to Salem's Lot", and the novel "Dracula", as well as two connecting associations, i.e.

$$Difference(e_{moviesV3}, e_{movies}, \text{“}Newcomers\text{”}) = e_{newcomers}$$
$$\text{with} \qquad nodes(e_{newcomers}) = \{l_{dracula}, a_{dr \rightarrow no}, l_{nosferatu}, l_{salem}, a_{sa \rightarrow re}, l_{return}\}.$$

Definition 28. *[Difference] Let* $e_1, e_2 \in \Sigma$ *and* $s \in \mathbb{STR}$ *then the operator*
Difference : $\Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *is defined as*
Difference$(e_1, e_2, s) = (o_{e_s}$, *"s"*, *"emm"*, $\epsilon, \epsilon, \emptyset, \emptyset, \emptyset, N_{e_s}, \emptyset, \emptyset, \emptyset, \emptyset)$ *with* $o_{e_s} \in \mathbb{UUID}$
and $N_{e_s} = \text{nodes}(e_1) \backslash \text{nodes}(e_2) \cup \{x \mid \exists y \in \text{asso}(e_1) \backslash \text{asso}(e_2) \quad x = t_y \vee x = s_y\}$.

The operators *Union* and *Intersection* are defined in a similar way, the operator *Nest* extracts the information stored within a set of associations from an EMMO, i.e. triples consisting of source entity, association, and target entity, into a new EMMO knowledge structure, and the operator *Flatten* generates a flattened EMMO, i.e. all recursively contained higher level entities are added as first level entities to the nodes of the EMMO. Due to space restriction, we omit the formal definitions.

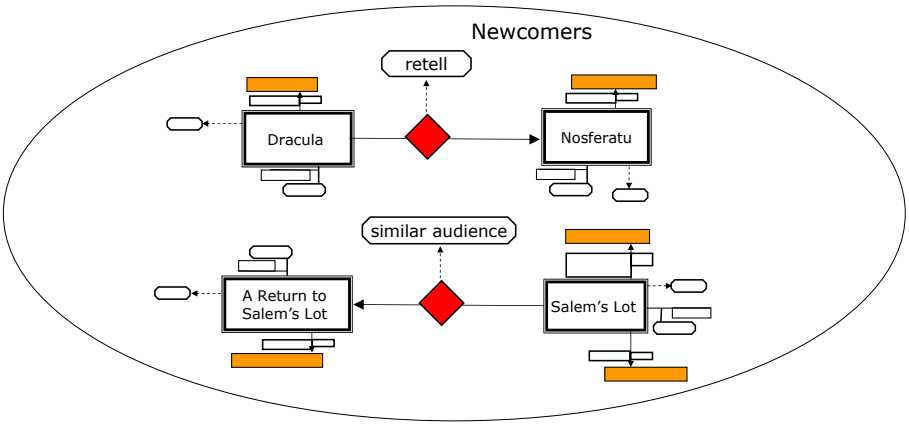

Fig. 8. EMMO "Newcomers" ($e_{newcomers}$)

5.6 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 values. 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 entities within EMMO "Zoa's Research" (see Fig. 9) which contain within their nodes the logical media part "Icarus' Fall" corresponds to the query expression

$$Join(nodes(e_{zoa}), \{l_{icarus}\}, nodes, id, \supseteq) = \{e_{studiesfall}\}$$

and yields EMMO "Studies about the Fall", because this EMMO includes the logical media part "Icarus' Fall".

Definition 29. *[Join] Let* $i \in I = \{1, \ldots n\}, n \in IN, W_i \subseteq \Omega, f_i \in \mathbb{FUN}$ *and* $p \in \mathbb{PRE}$*, then the operator* $Join : \prod_{i \in I} \mathcal{P}(\Omega) \times \prod_{i \in I} \mathbb{FUN} \times \mathbb{PRE} \longrightarrow \mathbb{SET}$ *is defined as* $Join(W_1, \ldots, W_n, f_1, \ldots, f_n, p) = \{\pi_1(w_1, \ldots, w_n) \,|\, \forall i \in I$ $(w_i \in W_i \wedge f_i \in \mathbb{FUN}_{W_i} \wedge p \in \mathbb{PRE}_{\prod_{i \in I} \mathcal{R}(f_i)} \wedge p(f_1(w_1), \ldots, f_n(w_n)))\}$.

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 values, e.g.

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

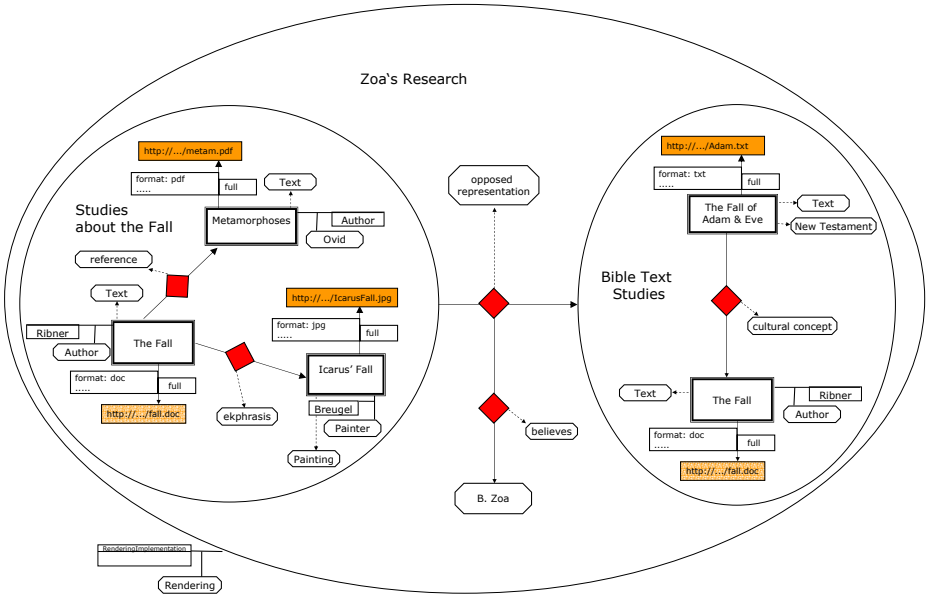

Fig. 9. EMMO "Zoa's Research"(e_{zoa})

5.7 Summary of EMMA Operators

EMMA provides operators to access the three aspects and the versioning information. The access to an EMMO's *media aspect* is realized by the operator *connectors* returning all connectors of a logical media part 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 *asso* retrieving all associations within an EMMO, 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 operator *Execute* addresses 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 guarantees 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 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* selects 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 enable the inclusion of ontological knowledge such as transitive and inverse association types, and supertype/subtype relationships.

By fulfilling all the requirements described in Sect. 3, EMMA can be said to be *adequate* and *complete* with regard to the EMMO model.

6 Implementation

For enabling content sharing and collaborative authoring of EMMOs, the implementation had to be realized on a distributed infrastructure. Thus, we have established *EMMO containers* constituting a management component for EMMOs, i.e. the space where EMMOs "live". The EMMO containers are not intended as a centralized infrastructure realized by one single Root EMMO container running at one server. Instead we establish a decentralized infrastructure with EMMO containers of different scale and sizes running at different, distributed servers. To realize a decentralized EMMO management infrastructure two requirements need to be fulfilled:

- The users of EMMO containers are manifold, i.e. ranging from individual users running a home PC to multimedia content publishers. In other words, the systems running the EMMO containers are very heterogeneous servers with different sizes, operating systems, capabilities, and requirements. Therefore, the implementation of the EMMO container infrastructure needed to be *platform independent* and *scalable*. We have implemented the EMMO containers in *Java* and used the object-oriented DBMS *ObjectStore* for their persistent storage. By using Java we achieved platform independency and by using ObjectStore for the persistent storage of EMMOs the scalability of EMMO containers could be realized, i.e. besides a full-fledged database server implementation suitable for larger content providers, ObjectStore also provides a code-compatible file-based in-process variant *PSEPro* that better suits the limited capabilities and needs of home users.
- For enabling the sharing and collaborative authoring of multimedia content, EMMOs must be *transferable* between the different EMMO containers. Therefore, *export* and *import* facilities for EMMO containers, reflecting an EMMO's content, i.e. its three aspects and versioning information, are required. As EMMOs can describe quite complex structures, it is important for the users to specify the parts of the EMMO they want to export. They

can choose between four *export options* indicating whether the EMMO is transferred with or without media objects, with or without versioning information, with or without encapsulated entities, and with or without the attached operations. EMMO containers export their EMMOs to a bundle structure, i.e. a ZIP archive that captures an EMMO's three aspects and versioning information, and indicates the specified export option.

We have implemented the *EMMA query processing architecture* with query optimization in mind, however, the realization of a query optimizer is subject of future work. The EMMA query processing architecture, which is depicted in Fig. 10, is based on the implementation of the EMMO container infrastructure described above. Its focus is the extraction and navigation of information stored within the EMMO containers. The EMMA query processing architecture takes syntactically well-defined query expressions as input. The processing of the query expressions reflects the definition of the EMMA query operators and produces a set of EMMO objects in a pre-defined output format.

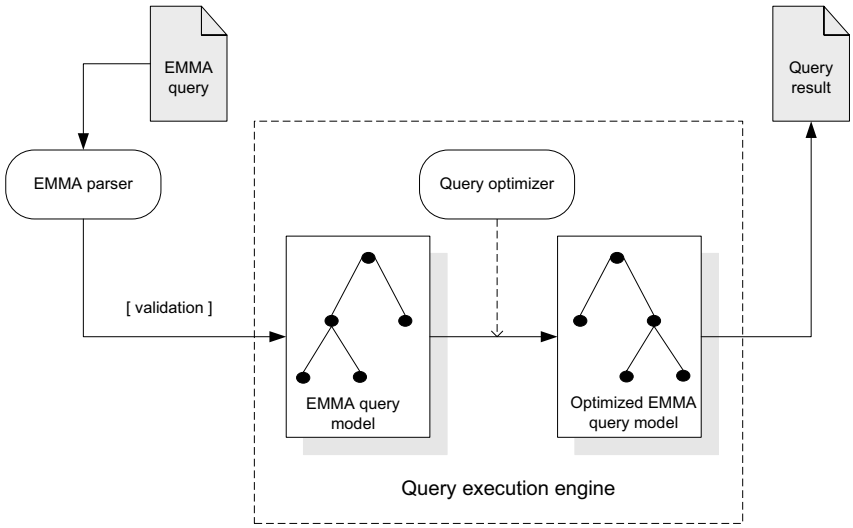

Fig. 10. The EMMA query processing architecture

For the implementation of the EMMA operators, we have chosen a functional approach, i.e. each operator has a corresponding function that evaluates according to its implementation-specific algorithm. For enabling consistency and integrity checking, each function has a signature that defines the number and types of input arguments, and, additionally, the types of the expected output values. By typing all EMMO and EMMA model constructs according to an in-

ternal hierarchy, those constructs can be used for specifying the signature of functions.

For realizing complex queries, i.e. the nesting of modular EMMA operators, the *EMMA query model* is built up. The EMMA query model is a tree consisting of nodes and leaves. Nodes represent algebraic operators, and leaves correspond to EMMO and EMMA model constructs, i.e. values of the underlying EMMO container. The EMMA query model is supplied with a built-in validation mechanism, ensuring that operators in the query tree contain only valid references to subsequent nodes, i.e. before evaluating the complex structure of the query model tree, a consistency and integrity check concerning the signature of the functions implementing the EMMA operators is performed.

By applying a *bottom-up evaluation* technique, the execution computes the final query result. This evaluation technique runs through several steps. First, any EMMO or EMMA model construct captured by the EMMO containers that represents a valid input value for the query expression is fetched. Then, all possible output values – represented as tuples – that can be derived when applying the function's algorithm reflecting the definition of the corresponding EMMA operator of the fetched input values, are computed. Going up the tree hierarchy, this process is repeated by applying functions to the set of objects in the EMMO store together with those tuples which were inferred in the previous step. This process is repeated until the root of the query tree is reached and the final result set is delivered.

Query optimization is realized by the EMMA query optimizer, which takes a query model and transforms it into an equivalent model that can evaluate more efficiently. The design of the transformation algorithm is based on the evaluation of the response time of query expressions and is subject of future work.

7 Conclusion

In this paper, we have introduced the formal basis of the query algebra EMMA, which enables the efficient retrieval of the knowledge represented by EMMOs, a novel approach to semantic multimedia meta modeling. EMMA's operators provide the access to all information and aspects stored within EMMOs and are based on precise semantics, thus offering a formal basis for query rewriting and optimization. EMMA features orthogonal, arbitrarily combinable operators that range from simple selection and extraction operators to more complex navigational operators, joins, and even rudimentary operators for the construction and manipulation of EMMOs. Moreover, EMMA renders it possible to integrate ontological knowledge within queries, such as supertype/subtype relationships, transitive or inverse association types. We have briefly sketched the implementation of the EMMO container infrastructure and the EMMA query processing architecture.

In our future work, we will focus on the realization of an eLearning scenario by means of the EMMO infrastructure. Based on real-world data gathered from this use case, we will carry out experiments for performance evaluation, in partic-

ular to achieve a detailed analysis and understanding of the effects of the various factors on query performance. We will use the application-specific data as starting point for the development of an EMMA query optimizer. Furthermore, we are in the process of developing an ontology engineering environment, consisting of an ontology description language compatible with the EMMO model, and tools that enable the seamless integration of ontological knowledge into query processing.

References

[1] Raggett, D., Hors, A.L., Jacobs, I.: HTML 4.01 Specification. W3C Recommendation, World Wide Web Consortium (W3C) (1999)

[2] Ayars, J., et al.: Synchronized Multimedia Integration Language (SMIL 2.0). W3C Recommendation, World Wide Web Consortium (W3C) (2001)

[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] Cruz, I., Sayenko, O.: Semantically Driven Multimedia Querying and Presentation. In Srinivasan, U., Nepal, S., eds.: Managing Multimedia Semantics. IDEA Group Publishing, Hershey PA, USA (2005)

[6] Hunter, J.: Enhancing the Semantic Interoperability of Multimedia Through a Core Ontology. IEEE Transaction on Circuits and Systems for Video Technology **13** (2003)

[7] van Ossenbruggen, J., Nack, F., Hardman, L.: That Obscure Object of Desire: Multimedia Metadata on the Web (Part I). IEEE MultiMedia **11** (2004)

[8] Nack, F., van Ossenbruggen, J., Hardman, L.: That Obscure Object of Desire: Multimedia Metadata on the Web (Part II). IEEE MultiMedia **12** (2005)

[9] Nack, F., Hardman, L.: Towards a Syntax for Multimedia Semantics. CWI Report INS-RO204, Centrum voor Wiskunde en Informatica (2002)

[10] Hammiche, S., et al.: Semantic Retrieval of Multimedia Data. In: Proc. of the Second ACM International Workshop on Multimedia Databases, Washington, DC, USA (2004)

[11] Srinivasan, U., Nepall, S., eds.: Managing Multimedia Semantics. IDEA Group Publishing, Hershey PA, USA (2005)

[12] 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: Proc. of the Conference on Distributed Multimedia Systems (DMS 2003), Miami, Florida (2003)

[13] Newman, D., Patterson, A., Schmitz, P.: XHTML+SMIL Profile. W3C Note, World Wide Web Consortium (W3C) (2002)

[14] ISO/IEC JTC 1/SC 34/WG 3: Information Technology – Hypermedia/Time-Based Structuring Language (HyTime). International Standard 15938-5:2001, ISO/IEC (1997)

[15] ISO/IEC IS 13522-5: Information Technology – Coding of Hypermedia Information – Part 5: Support for Base-Level Interactive Applications. International Standard, ISO/IEC (1996)

[16] Pereira, F., Ebrahimi, T., eds.: The MPEG-4 Book. Pearson Education, California (2002)

[17] Beckett, D.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, World Wide Web Consortium (W3C) (2004)
[18] Brickely, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, World Wide Web Consortium (W3C) (2004)
[19] 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)
[20] 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)
[21] ISO/JTC 1/SC 32/WG 2: Conceptual Graphs. ISO/IEC International Standard, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) (2001)
[22] 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 (2005)
[23] Zillner, S., Westermann, U., Winiwarter, W.: EMMA – Towards a Query Algebra for Enhanced Multimedia Meta Objects. In: Proc. of the Fourth International Conference on Computer and Information Technology (CIT 2004), Wuhan, China (2004)
[24] Zillner, S., Westermann, U., Winiwarter, W.: EMMA – A Query Algebra for Enhanced Multimedia Meta Objects. In: Proc. of the Third International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2004), Larnaca, Cyprus (2004)
[25] Billiani, F., et al.: Demonstrator of Intertextual Cultural Threads – Standard Ontology-Extended Ontology. Public Deliverable Version 2.0, CULTOS Consortium and Project Planning (2003)
[26] Benari, M., et al.: Proposal for a Standard Ontology of Intertextuality. Public Deliverable Version 2.0, CULTOS Consortium and Project Planning (2003)
[27] Leach, P.: UUIDs and GUIDs. Network Working Group Internet-Draft, The Internet Engineering Task Force (IETF) (1998)
[28] Zillner, S., Winiwarter, W.: Ontology-Based Query Refinement for Multimedia Meta Objects. In: Proc. of the Sixth International Conference on Information Integration and Web Based Applications & Services (iiWAS 2004), Jakarta, Indonesia (2004)
[29] Zillner, S., Winiwarter, W.: Integrating Ontology Knowledge into a Query Algebra for Multimedia Meta Objects. In: Proc. of the Fifth International Conference on Web Information Systems Engineering (WISE 2004), Brisbane, Australia (2004)
[30] Zillner, S., Winiwarter, W.: Integration of Ontological Knowledge within the Authoring and Retrieval of Multimedia Meta Objects. International Journal of Web and Grid Services (IJWGS) **1** (2005)
[31] Cattell, R., ed.: The Object Database Standard: ODMG-93. Morgan, Kaufmann, San Francisco, CA (1994)
[32] Leung, T., et al.: The Aqua Data Model and Algebra. In: Proceedings of the Fourth International Workshop on Database Programming Languages – Object Models and Languages, Manhattan, New York City (1993)
[33] Kifer, M., Kim, W., Sagiv, Y.: Querying Object-Oriented Databases. In: Proc. of the ACM SIGMOD Conference on Management of Data, San Diego, CA (1992)

[34] Papakonstantinou, Y., Garcia-Molina, H., Widom, J.: Object Exchange Across Heterogeneous Information Sources. In: Proc. of the Eleventh International Conference on Data Engineering, Taipei (1995)
[35] Abiteboul, S., et al.: The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries **1** (1997) 68–88
[36] Bruneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. The VLDB Journal – The International Journal on Very Large Data Bases **9** (2000)
[37] Beeri, C., Tzaban, Y.: SAL: An Algebra for Semistructured Data and XML. In: Proc. of the Second International Workshop on the Web and Databases (WebDB 99), Philadelphia, Pennsylvania, USA (1999)
[38] Boag, S., et al.: XQuery 1.0: An XML Query Language. W3C Working Draft, World Wide Web Consortium (W3C) (2005)
[39] Berglund, A., et al.: XML Path Language (XPath). W3C Working Draft Version 2.0, World Wide Web Consortium (W3C) (2005)
[40] Lee, T., et al.: Querying Multimedia Presentations Based on Content. IEEE Transactions on Knowledge and Data Engineering **11** (1999)
[41] Duda, A., Weiss, R., Gifford, D.: Content Based Access to Algebraic Video. In: Proc. of the IEEE First International Conference on Multimedia Computing and Systems, Boston, MA, USA (1994)
[42] Adali, S., Sapino, M., Subrahmanian, V.: A Multimedia Presentation Algebra. In: Proc. of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA (1999)
[43] Karvounarakis, G., et al.: RQL: A Functional Query Language for RDF. In Gray, P.M.D., et al., eds.: The Functional Approach to Data Management. Springer, Heidelberg, Germany (2003)
[44] Miller, L., Seaborn, A., Reggiori, A.: Three Implementations of SquishQL, a Simple RDF Query Language. In: Proc. of the First International Semantic Web Conference (ISWC2002), Sardinia, Italy (2002)
[45] Frasincar, F., et al.: RAL: An Algebra for Querying RDF. In: Proc. of the Third International Conference on Web Information Systems Engineering (WISE 2000), Singapore (2002)
[46] 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)
[47] Garshol, L.: Tolog 0.1. Ontopia Technical Report, Ontopia (2003)
[48] Bogachev, D.: TMPath – Revisited. Online Article, available under http://homepage.mac.com/dmitryv/TopicMaps/TMPath/TMPath Revisited.html (2004)
[49] 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)
[50] Widhalm, R., Mück, T.: Topic Maps (in German). Springer, Berlin Heidelberg, Germany (2002)
[51] Manjunath, B., Salembier, P., Sikora, T., eds.: Introduction to MPEG-7. John Wiley & Sons, West Sussex, UK (2002)
[52] Zillner, S.: A Query Algebra for Ontology-enhanced Management of Multimedia Meta Objects. PhD thesis, Vienna University of Technology (2005)