Technical Report, TR-2004301 **, Dept. of Computer Science and Business Informatics, University of Vienna,** June **,** 2004

The EMMA Algebra for EMMOs - Compendium

Sonja Zillner

Dept. of Computer Science and Business Informatics
University of Vienna, Austria

sonja.zillner@univie.ac.at

Contents

Chapter 1

Introduction

Enhanced Multimedia Meta Objects (EMMOs) are a novel approach of semantic multimedia content modeling in content sharing and collaborative applications. EMMOs were developed within CULTOS, an EU-funded project, which was going from September 2001 until October 2003 and carried by 11 partners from different EU countries, Israel and Estonia.

CULTOS addresses the needs of researchers in the domain of intertextual studies for an integrated view on individual and culture-dependent perceptions of interrelationships between artefacts. This knowledge about the interrelationships between artefacts is gathered within so-called *InterTextual Threads (ITTs)*, i.e. complex knowledge structures that semantically interrelate and compare cultural artefacts such as literature, artworks movies, etc. EMMOs provide an adequate foundation for the representation of multimedia enriched ITTSs, thus paving the way for an Internet-based multimedia platform for the collaborative authoring, managing, retrieving, exchanging, and presenting of ITTs.

For the processing of EMMOs suitable querying facilities are required. For that purpose, EMMA, an expressive query algebra that is adequate and complete with regard to the EMMO model, was developed. EMMA offers a rich set of formally defined, orthogonal query operators providing an adequate foundation for the realization of powerful EMMO querying services.

The remainder of the paper is organized as follows. Chapter 2 gives an introduction to ITTs. Chapter 3 explains the basic ideas if EMMOs and provides its formal definition. Chapter 4 provides a formal definition of the EMMA query algebra.

Chapter 2

What are InterTextual Threads (ITTs)?

A central task of researchers in intertextual studies is to discover the relationships between pieces of literature and other works of art thereby elaborating InterTexual Threads (ITTs). ITTs can be represented with graphical structures that may take a variety of forms, ranging from spiders over centipedes to associative maps as shown in Fig. 2.1.

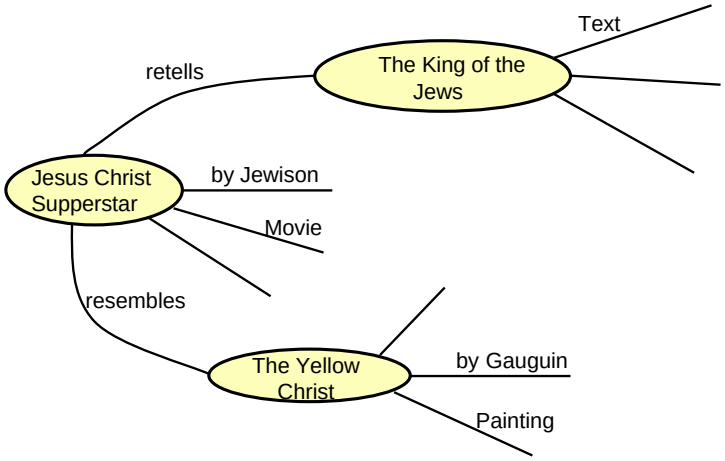

Figure 2.1: Simple InterTextual Thread

The example ITT depicted in the figure highlights several relationships of the movie “Jesus Chris Superstar” by Norman Jewison to other works of art. It states that the movie retells the text “The King of the Jews” and that the movie resembles the painting “The Yellow Christ” of the famous painter Paul Gauguin.

When looking at the ITT, well-known techniques from the domain of knowledge engineering like conceptual graphs and semantic nets immediately come to mind. Indeed, the depicted graphical representation of the ITT bears a strong resemblance to such techniques, though it lacks their formal rigidity. However, the complexity of ITTs should not be underestimated. ITTs commonly make use of constructs that are very

challenging from the perspective of knowledge representation, such as *encapsulation* and *reification of statements*.

Encapsulation is intrinsic to ITTs because intertextual studies are not exact sciences. Certainly, the cultural and personal context of a researcher will affect the kinds of relationships between pieces of literature and works of art that are discovered and of importance to the researcher. This inevitably results in differences and even contradictions between different ITTs created by two different researchers on the same subject. As there thus cannot be a global "truth", every ITT is a "truth" in its own right that has to be protected by an encapsulating impenetrable boundary. Moreover, differences on a certain subject are highly interesting facts for researchers in intertextual studies. Consequently, ITTs themselves can be relevant subjects of discourse and thus be contained as first-class artefacts within other ITTs.

Reification of statements is yet another demanding construct frequently occurring within ITTs. Since experts in intertextual studies extensively base their position on the position of other researchers, statements about statements are common practice within ITTs. Typically, reification is not just a one-step process: statements about already reified statements are no rarity.

Chapter 3

The EMMO Model

To create a suitable foundation for the representation of ITTs, we have developed Enhanced Multimedia Meta Objects (EMMOs).

An EMMO is a self-contained unit of multimedia content that encompasses three aspects, which we would like to illustrate using Fig. 3.1 that depicts a sketch of an EMMO representing the ITT of Fig. 2.1.

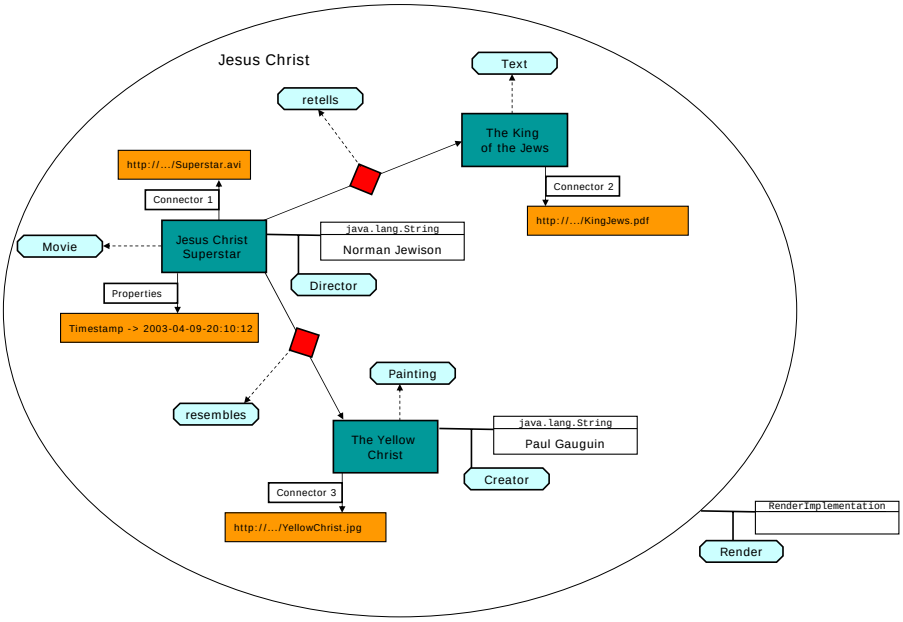

Figure 3.1: EMMO "Jesus Christ"(e_{jesus})

1. *The media aspect:* An EMMO aggregates the media objects of which the multimedia content consists. In the figure, we see that the depicted EMMO contains the avi video "Superstar.avi", the pdf document "KingJews.pdf", and the JPEG image "YellowChrist.jpg". Containment of media objects can be realized either by inclusion, i.e., the raw media data is embedded within an EMMO, or by reference via an URI, in cases where embedding media data is not feasible.

2. *The semantic aspect:* An EMMO further encapsulates semantic associations between its contained media objects by means of a graph-based model similar to conceptual graphs. Hence, an EMMO constitutes a unit of expert knowledge concerning the multimedia content. In the example figure, it is stated that the media objects contained within the EMMO are digital manifestations of Jewison's movie "Jesus Christ Superstar", the text "The King of the Jews", and Gauguin's painting "The Yellow Christ". Also, the interpretation of the author of the original ITT has been remodeled: "Jesus Christ Superstar" retells "The King of the Jews" and resembles "The Yellow Christ". The model used for semantic associations is expressive: it is, e.g., possible to establish references to other EMMOs and to reify associations.

3. *The functional aspect:* An EMMO offers operations for dealing with its content which applications can invoke. In the figure, the depicted EMMO is associated with one operation for rendering the EMMO. The operation might return a presentation of the EMMO in different formats, such as SMIL and SVG.

EMMOs have further desirable characteristics. They can be *serialized* into a bundle that completely encompasses all three aspects. Thus, an EMMO is *transferable* in its entirety between different EMMO providers, including its contained media objects, semantic associations between these objects, and functionality. Moreover, *versioning support* has been a central design objective: all the constituents of an EMMO can be versioned, thereby paving the way for the distributed, *collaborative construction* of EMMOs.

In the following, we describe and formally define the EMMO model and illustrate how the model can be used to build and represent multimedia-enhanced ITTs. We begin by introducing the concept of *entities* which constitute an abstract notion subsuming the different constituents of the EMMO model (3.1). We then define the four concrete specializations of entities, namely *logical media parts* representing media objects, *ontology objects* representing concepts of an ontology, *associations* modeling binary relationships between entities, and *EMMOs* themselves which are aggregations of semantically related entities (3.2 – 3.5).

3.1 Entities

Before we start with a formal definition of the abstract notion of entities, we clarify some basic symbols required for the definitions to follow:

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.*

Furthermore, let $\mathcal{MS}$ *be the set of all media selectors,* $\mathcal{MP}$ *the set of all media profiles, and* $\mathcal{OP}$ *be the set of all operations.*

Finally, let $\mathbb{OID}$ *be the set of all universal unique identifiers,* $\mathbb{STR}$ *the set of all strings,* $\mathbb{OBJ}$ *the set of all objects,* $\mathbb{URI}$ *the set of all uniform resource identifiers,* $\mathbb{RMD}$ *the set of all raw media data, and* $\mathbb{FUN}$ *the set of all functions.*

Based on these common symbols, the definition of entities is formulated below:

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* $o_w \in \mathbb{OID}$ *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,* $A_w \subseteq \Theta \times \mathbb{OBJ}$ *the attribute values,* $T_w \subseteq \Theta$ *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,* $F_w \subseteq \mathbb{STR} \times \mathbb{OBJ}$ *the features, and* $O_w \subseteq \mathcal{OP}$ *the operations of* w*. The following constraints hold for all entities:*

$$\forall w_1, w_2 \in \Omega : o_{w_1} = o_{w_2} \longrightarrow w_1 = w_2 \tag{3.1}$$

$$\forall w, v \in \Omega : v \in P_w \vee v \in S_w \longrightarrow k_w = k_v \tag{3.2}$$

According to the definition, an entity w is globally and uniquely identified by its OID o_w as ensured by Constraint (3.1). Since we have chosen o_w to be a universal unique identifier (UUID) [3], OIDs can easily be generated even in a distributed scenario like the CULTOS project. As UUIDs are not really useful to humans, an entity can be augmented with a human readable name n_w which is a string. The kind k_w serves to identify whether an entity is either a logical media part, an ontology object, an association, or an EMMO.

An entity w may further have an arbitrary number of types T_w. Types are concepts taken from an ontology, so for instance, an entity might be an instantiation of the concepts ”text” and ”movie”; another might instantiate the concept ”painting”, etc. By attaching types, an entity gets meaning and is classified in an application-dependent ontology. In the EMMO model, types are represented as ontology objects and thus constitute entities themselves.

An entity can additionally be described by an arbitrary number of attribute values A_w. Attribute values are simple attribute-value pairs with the attribute being a concept of an application-dependent ontology (similar to types represented by an ontology object in the EMMO model) and the value being an arbitrary object. With attribute values, it is for instance possible to state that a movie has been directed by Norman Jewison by attaching the attribute value ”director=Norman Jewison” to the entity representing that movie in the EMMO model. The attribute ”director” would be an ontology object and the value ”Norman Jewison” would probably be a string value. The rationale behind representing attributes as concepts of an ontology and not just as simple string identifiers is that this allows to express constraints on the usage of attributes within the ontology, e.g., which entity types attributes are applicable to.

As already mentioned, the CULTOS project intends to develop a distributed platform allowing researchers in intertextual studies to work collaboratively on ITTs. In such an environment, different versions of their work will accrue not only due to the temporal evolution of a researcher’s viewpoints but also due to concurrent work of different reseachers on the same ITTs. Since different versions of ITTs are highly interesting facts to researchers, it is important to be able to trace these versions and to interrelate them within other ITTs. The EMMO model takes account of this need for versioning by allowing any entity w to have an arbitrary number of direct preceding versions P_w and direct succeeding versions S_w. A version of w is again an entity of the same kind k_w, as expressed by Constraint (3.2). Treating an entity’s versions as

entities on their own has several benefits: on the one hand, entities constituting versions of other entities have their own globally unique OID. Hence, different versions concurrently derived from one and the same entity at different sites can easily be distinguished without synchronization effort. On the other hand, different versions of an entity can be interrelated just like any other entities allowing to establish comparative relationships between entity versions as desired in intertextual studies.

The features F_w of an entity w represent a fixed set of primitive attribute-value pairs. They have been included in the EMMO model as it might be necessary to augment entities with further attributes, e.g., time-stamps or status information, in an implementation of the model.

The remaining elements and sets given by the definition – the source and target entities s_w and t_w, the connectors C_w, the nodes N_w, the operations O_w – are only relevant for certain kinds of entities. Therefore, we defer their explanation to the sections to follow as they become relevant.

3.2 Logical Media Parts

Logical media parts are entities serving to represent the media objects or parts of media objects of which multimedia content consists at a logical level within the EMMO model. When modeling a multimedia-enhanced ITT as an EMMO, logical media parts address the cultural artefacts that are subject of discourse within the ITT, for example, pieces of literature, movies, paintings, etc. In order to relief authors from the burden of having digital representations of the artefacts to be treated at hand before they can start building an ITT, special care has been taken to decouple logical media parts from any existing physical representation. In fact, one can talk about Gauguin's painting "The Yellow Christ" and find intertextual relationships to other (art)works without owning a JPG image showing that painting.

However, if an author focuses on the difference between, e.g., a movie of "Jesus Christ Superstar" as seen on television and the corresponding cd recording, the television broadcast and the cd recording will become two distinct media objects on a logical level and thus have to be represented by two different logical media parts. If nothing of this kind has to be expressed, a single logical media part will suffice for representing "Jesus Christ Superstar".

Definition 3 formally introduces logical media parts:

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

It is important that the definition does not restrict the set of connectors C_l of a logical media part l, which has been defined to exist for all entities in Definition 2, to an empty set: logical media parts not only model media objects at a logical level but are additionally able to maintain connections to media data representing these objects. Thereby, logical media parts provide the media aspect of multimedia content represented with the EMMO model.

Connectors (see Definition 2) consist of a media profile and a media selector. Media profiles, in accordance to the media tool set of MPEG-7 [2], represent media data. A media profile combines low-level metadata describing the media data, e.g., the storage format, with its storage location – a media instance in MPEG-7 terminology. A media instance can either address the location of media data by means of an URI or it may directly embed the media data. The ability to embed media data allows to combine

media data and multimedia content described with the EMMO model based on these media into single, indivisible units.

Definition 4 formally captures media profiles and media instances in the EMMO model.

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* $M_{mp} \subseteq \mathbb{STR} \times \mathbb{OBJ}$.

Media selectors contained in connectors along with media profiles can address parts of the media data represented by the profile according to textual, spatial, and temporal criteria. For example, it should be possible to address a scene in a digital video starting from second 10 and lasting until second 30 without having to extract that scene and to put it into a separate file using a video editing tool.

Definition 5 introduces media selectors.

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

Example 1 shows how the three cultural artefacts occurring in the sketch of the EMMO named "Jesus Christ" in Figure 3.1 can be represented as individual logical media parts in the EMMO model. In the example, the logical media parts have been labeled l_0, l_1, and l_2. The connector of the logical media part l_2 references the upper left corner of the JPG image file located at the URI "`http://.../YellowChrist.jpg`" which is expressed by the media profile mp_2 in combination with the spatial selector ms_2.

Example 1

$$
\begin{aligned}
l_0 =& ("a3564", "Jesus\,Christ\,Superstar", "lmp", \epsilon, \epsilon, \{o_{movie}\},\\
& \{(o_{director}, Norman\,Jewison)\}, \{(ms_0, mp_0)\}, \emptyset, \emptyset, \emptyset,\\
& \{("Timestamp", 2003\text{-}04\text{-}09\text{-}20{:}10{:}12), \ldots\}, \emptyset)\\
l_1 =& ("a7655", "The\,King\,of\,the\,Jews", "lmp", \epsilon, \epsilon, \{o_{text}\},\\
& \emptyset, \{(ms_1, mp_1)\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)\\
l_2 =& ("b4567", "The\,Yellow\,Christ", "lmp", \epsilon, \epsilon, \{o_{painting}\},\\
& \{(o_{creator}, Paul\,Gauguin)\}, \{(ms_2, mp_2)\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)\\
mp_0 =& ("\texttt{http://.../Superstar.avi}", \{(\textit{format}, \textit{avi}), \ldots\})\\
ms_0 =& ("full", \emptyset)\\
mp_1 =& ("\texttt{http://.../KingJews.pdf}", \{(\textit{format}, \textit{pdf}), \ldots\})\\
ms_1 =& ("full", \emptyset)\\
mp_2 =& ("\texttt{http://.../YellowChrist.jpg}", \{(\textit{format}, \textit{jpg}), \ldots\})\\
ms_2 =& ("spatial", \{(startpoint, (0,0)), (endpoint, (50,50))\})
\end{aligned}
$$

3.3 Ontology Objects

Ontology objects are the kind of entities that represent concepts of an ontology. As already explained, ontology objects among others serve to designate the types of entities or the attributes of attribute values attached to entities.

In the CULTOS project, the experts in intertextual studies have defined an ontology featuring the concepts necessary to represent ITTs within the EMMO model. As we have not developed an ontology language for the EMMO model yet, we follow the pragmatic approach of defining the concepts of the CULTOS ontology in an external ontology language such as RDF Schema [1] and letting the ontology objects reference these concepts.

Definition 6 formally introduces ontology objects:

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

Example 2 illustrates ontology objects again using the sketch of an EMMO of Figure 3.1. The ontology objects $o_{resembles}$ and $o_{retells}$ represent the types of the two associations contained in Fig. 3.1, i.e., "resembles" and "retells". The ontology objects o_{movie}, o_{text} and $o_{painting}$ model the types of the three logical media parts depicted, and the ontology objects $o_{director}$ and $o_{creator}$ label the attribute-value pairs of the logical media part l_0 and l_1. The ontology object o_{render} finally represent the designator of the operation the sketched EMMO offers (these will be explained later in conjunction with EMMOs).

Example 2

$$
\begin{aligned}
o_{movie} &= (\text{"}c3456\text{"}, \text{"}Movie\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
o_{director} &= (\text{"}c4516\text{"}, \text{"}Director\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
o_{text} &= (\text{"}c1162\text{"}, \text{"}Text\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
o_{painting} &= (\text{"}c2356\text{"}, \text{"}Painting\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
o_{creator} &= (\text{"}c2333\text{"}, \text{"}Creator\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
o_{resembles} &= (\text{"}c5627\text{"}, \text{"}resembles\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
o_{retells} &= (\text{"}c4111\text{"}, \text{"}retells\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \\
o_{render} &= (\text{"}c3336\text{"}, \text{"}Render\text{"}, \text{"}ont\text{"}, \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)
\end{aligned}
$$

3.4 Associations

Associations represent binary directed semantic relationships between entities. Thus, they provide the semantic aspect of multimedia content represented on the basis of the EMMO model. In the CULTOS project, they are in particular used to model the intertextual relationships between cultural artefacts within ITTs. Since associations are "first-class" entities, they can take part in associations as well facilitating the reification of statements in the EMMO model. As we have explained before, expressing

statements about statements is a very essential part of the work of experts in intertextual studies when analyzing literature and building their ITTs.

Definition 7 formally describes associations:

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

According to the definition, the kind of semantic relationship represented by an association is defined by the association's type which is – like the types of other entities – an ontology object representing a concept taken from an ontology. Different from other entities, however, an association is only allowed to have one type as it can represent only a single kind of relationship. Each association specifies exactly one source and one target entity s_a and t_a, and thus establishes a directed binary relationship between those two entities.

Example 3 shows the representation of the two associations given in the example EMMO of Figure 3.1.

Example 3

$$a_0 = ("g7490", "a_0", "asso", l_0, l_1, \{o_{retells}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$$
$$a_1 = ("w4399", "a_1", "asso", l_0, l_2, \{o_{resembles}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$$

3.5 EMMOs

The EMMO is the core component of our model. It is a container that groups arbitrary entities into a single unit. An EMMO can thus address the media and semantic aspects of multimedia content by aggregating media data (i.e., logical media parts) with semantic data (i.e., associations and ontology objects). The functional aspect of multimedia content can be addressed as well by augmenting the EMMO with arbitrary operations that process the content.

In CULTOS, EMMOs act as the containers carrying multimedia-enhanced ITTs. In such a container, the cultural artefacts covered by an ITT are captured as logical media parts, media data digitally representing these artefacts are attached to the media parts via connectors, the intertextual relationships are modeled by means of associations, and concepts from the domain of intertextual studies are covered by ontology objects. The possibility to attach operations to an EMMO is exploited, among others, to provide ITTs with the ability to render themselves as SMIL and SVG presentations.

Since EMMOs are "first-class" entities, EMMOs can be contained within other EMMOs just like any other entity. As a consequence, a structure of hierarchically-nested EMMOs can be established. With regard to the representation of ITTs by the means of EMMOs, this is of particular advantage: it is possible to interrelate two different EMMOs representing two different ITTs with two different points of view on a subject within an EMMO representing a third ITT. In that manner, contradictions and relevant differences between both viewpoints can be expressed which is important for intertextual studies.

Definition 8 formally captures EMMOs:

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*

$$\forall x \in N_e : \quad k_x = "asso" \longrightarrow \{s_x, t_x\} \subseteq N_e \tag{3.3}$$

As shown in the formal description above, 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 the other kinds of entities in the EMMO model. These contained entities form a connected graph structure when they become interlinked by associations within the EMMO e.

Only entities belonging to the EMMO's nodes can be specified as source or target entity of an association (Constraint (3.3)). In this way, it is guaranteed that established relationships are fully contained in an EMMO.

Definition 8 unveils a further difference between EMMOs and other kinds of entites: an EMMO is more powerful in that it can have operations attached, because its set of operations O_e is not necessarily empty. In the EMMO model, an operation is basically a tuple combining an ontology object acting as the operation's designator with the operation's implementation, which can be any mathematical function. It is the intention of this modeling to achieve a high flexibility by allowing to attach arbitrarily complex operations to EMMOs. In a concrete implementation of the model, an operation could be realized as a function in the underlying programming language with the full expressiveness of that language at disposal. An operation could then reference this function by means of a function pointer. We have modeled operation designators as ontology objects to be able to express constraints on operations within ontologies, e.g., for which types of EMMOs an operation is available.

Definition 9 formally defines the notion of operations:

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

To conclude the formal definition of the EMMO model, Example 4 assembles the EMMO "Jesus Christ" (e_{jesus}) sketched in Figure 3.1 from the entities of the other examples. The function `RenderImplementation` implement the "Render" operation that is attached to EMMO "Jesus Christ". For example, `RenderImplementation` could be a mathematical function that takes an EMMO as its input and transforms it to an appropriate SMIL presentation, i.e., to a string that follows the SMIL syntax.

Example 4

$$\begin{aligned} e_{jesus} = (&"f4672", "Jesus\,Christ", "emm", \varepsilon, \varepsilon, \emptyset, \emptyset, \emptyset, \emptyset, \\ &\{l_0, l_1, l_2, a_0, a_1\}, \emptyset, \emptyset, \emptyset, \{(o_{render}, \texttt{RenderImplementation})\}) \end{aligned}$$

Chapter 4

The EMMO Algebra (EMMA)

In the following sections, we will first provide some preliminary definitions. Subsequently, we will introduce the five classes of EMMA operators.

4.1 Preliminary Definitions

The central constructs in the EMMO model and algebra are described by sets and sequences. In this section, we introduce all relevant mathematical definitions that will be used in the following.

4.1.1 Sets and Sequences

A set is a collection of distinct objects and can either be described by enumerating its members, e.g. $\{x_1, x_2, \ldots, x_n\}$ or by specifying a condition, e.g. $\{x \,|\, x \subseteq A\}$. We denote sets with capital letters, and elements with lower case letters, e.g $A = \{x, y, z\}$. $|A|$ stands for the cardinality of set *A* and is equal to the number of elements in *A*, and $\mathcal{P}(A)$ with $\mathcal{P}(A) = \{x \,|\, x \subseteq A\}$ defines the powerset of set A. Finally, $\emptyset$ denotes the empty set, i.e. a set with no members, and $\mathbb{SET}$ the set of all sets.

Further, we denote the set of all Strings by $\mathbb{STR}$, the set of all natural numbers by $\mathbb{N}$, the set of all real numbers by $\mathbb{R}$, and the Boolean set, i.e. $\{true, false\}$ by $\mathbb{BOO}$.

We use the well-known operations for computing the cardinality, union, intersection and difference of sets.

Definition 10 *[Set Operations] Let A, B, and $A_i, i \in I$ be arbitrary sets, then*
$card(A) = |A|$ *defines the cardinality of set A,*
$\bigcup(A, B) = A \cup B := \{x \,|\, x \in A \vee x \in B\}$ *defines the union of A and B, and* $\bigcup_{i \in I} A_i := \{x \,|\, \exists i \in I \, x \in A_i\}$ *defines the union of the sets $A_i, i \in I$.*
$\bigcap(A, B) = A \cap B := \{x \,|\, x \in A \wedge x \in B\}$ *defines the Intersection of A and B,* $\bigcap_{i \in I} A_i := \{x \,|\, \exists i \in I \, x \in A_i\}$ *defines the Intersection of the sets $A_i, i \in I$, and*
$\setminus(A, B) = A \setminus B := \{x \,|\, x \in A \wedge x \notin B\}$ *defines the Difference of A and B.*

Example 5 For all $i \in I = \{1, 2, \ldots, n\}$, let $A_i = \{0, i\}$, then
$\bigcup(A_1, A_2) = A_1 \cup A_2 = \{0, 1, 2\}$ and $\bigcup_{i \in I} A_i = \{0\} \cup I$,
$\bigcap(A_1, A_2) = A_1 \cap A_2 = \{0\}$ and $\bigcap_{i \in I} A_i = \{0\}$, and
$\setminus(A_1, A_2) = A_1 \setminus A_2 = \{1\}$.

A sequence is an ordered collection of objects and is defined as follows:

Definition 11 *[Cartesian Product and Sequence] Let A, B, and $A_i, i \in I$ be arbitrary sets, then $A \times B := \{(x, y) \mid x \in A \wedge y \in B\}$ denotes the cartesian product over A and B, and $\prod_{i\in I} A_i := \{x : I \longrightarrow \bigcup_{i\in I} A_i \mid \forall i \in I : x(i) \in A_i\} \subset (\bigcup_{i\in I} A_i)^I$ denotes the cartesian product over the sets $A_i, i \in I$. If for all $i \in I \quad A_i = A$, we write $\prod_{i\in I} A_i := A^I$ or $\prod_{i\in I} A_i := A^{|I|}$*

The elements of a cartesian product are called or tuples, and a sequence or tuple of length n is called n-sequence or n-tuple. The length of a sequence (tuple) is determined by the number of its contained elements, i.e. $\forall x \in \prod_{i\in I} A_i \quad length(x) = |I|$. The set of all sequences of length n (n-sequences or n-tuples) is denoted by $\mathbb{SEQ}_n$, and the set of all sequences (tuples) by $\mathbb{SEQ}$.

Example 6 For all $i \in I = \{1, 2, \ldots, n\}$ let $A = \{0, 1\}$, then
$A^1 = \{0, 1\}$,
$A^2 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$,
$A^3 = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}$,
and so on.
Then, the set $\bigcup_{i\in I} A^i$ consists all sequences (tuples) of arbitrary length over elements of the set $\{0, 1\}$. The length of each sequence (tuple) $x \in A^k$ is k.

In order to retrieve some specific information of sequences, we need to define the operation projection.

Definition 12 *[Projection] Let $I = \{1, 2, \ldots, n\}$ and $\prod_{i\in I} A_i$ be the cartesian product over the sets A_i.*

- *Let $j \in I$ then*

$$\begin{aligned} \pi_j : \prod_{i\in I} A_i &\longrightarrow A_j \quad \textit{with} \\ \pi_j(a_1, a_2, \ldots, a_n) &= a_j \end{aligned}$$

  denotes the j^{th} projection of the cartesian product $\prod_{i\in I} A_i$, and the return value a_j denotes projected sequence.

- *Let J= $\{j_1, \ldots j_k\} \subseteq I$ and $\forall l < k \quad j_l < j_{l+1}$ then*

$$\begin{aligned} \pi_J : \prod_{i\in I} A_i &\longrightarrow \prod_{i\in J} A_i \quad \textit{with} \\ \pi_J(a_1, a_2, \ldots, a_n) &= (a_{j_1}, a_{j_2}, \ldots, a_{j_k}) \end{aligned}$$

  denotes the projection of the cartesian product $\prod_{i\in I} A_i$ onto the cartesian product $\prod_{i\in J} A_i$, and the return value $(a_{j_1}, a_{j_2}, \ldots, a_{j_k})$ denotes projected sequence.

Example 7 *Let* $A_1 = \{1, 2, 3\}$,$A_2 = \{11, 12, 13\}$, *and* $A_3 = \{21, 22, 23\}$ *then*

$$\begin{aligned}\prod_{i=1,2,3} A_i \quad = \quad & \{(1,11,21),(1,11,22),(1,11,23),(1,12,21),(1,12,22),(1,12,23),\\ & (1,13,21),(1,13,22),(1,13,23),(2,11,21),(2,11,22),(2,11,23),\\ & (2,12,21),(2,12,22),(2,12,23),(2,13,21),(2,13,22),(2,13,23),\\ & (3,11,21),(3,11,22),(3,11,23),(3,12,21),(3,12,22),(3,12,23),\\ & (3,13,21),(3,13,22),(3,13,23),\}\end{aligned}$$

with $\pi_1((2,11,23)) = 2$ *and* $\pi_{\{1,2\}}((2,11,22) = (2,11)$.

4.1.2 Functions and Predicates

Functions and predicates will be of importance in the following definitions. A function basically describes a mapping from a so-called domain set to a so-called range set. Predicates are functions with boolean return type.

Definition 13 *[Functions] Let* $A, B \in \mathbb{SET}$ *and* $f : A \longrightarrow B$ *be a function, then we call* $D(f) = A$ *the domain (set) and* $R(f) = B$ *the range (set) of function f.* $\mathbb{FUN}_A$ *denotes the set of all functions with* $D(f) = A$, $\mathbb{FUN}_{[A,B]}$ *the set of all functions with* $D(f) = A$ *and* $R(f) = B$, *and* $\mathbb{FUN} = \{\mathbb{FUN}_A \mid A \in \mathbb{SET}\}$ *the set of all functions.*

Example 8 The function $f : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{R}$ with $f(x, y) = x + y$ has the domain $D(f) = \mathbb{R} \times \mathbb{R}$, the range $R(f) = \mathbb{R}$, and belongs to the set $\mathbb{FUN}_{[\mathbb{R}\times\mathbb{R},\mathbb{R}]}$.

Definition 14 *[Predicates] Let* $A \in \mathbb{SET}$ *, then we call* $p \in \mathbb{FUN}_{[A,\mathbb{BOO}]}$ *a predicate.* $\mathbb{PRE}_A = \mathbb{FUN}_{[A,\mathbb{BOO}]}$ *denotes the set of all predicates with domain A, and* $\mathbb{PRE} = \{\mathbb{PRE}_A \mid A \in \mathbb{SET}\}$ *the set of all predicates.*

Example 9 The predicate $p : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{BOO}$ with
$p(x, y) = \begin{cases} true & if \quad x + y < 0 \\ false & else \end{cases}$
has range $R(p) = \mathbb{R} \times \mathbb{R}$ and belongs to the set $\mathbb{PRE}_{\mathbb{R}\times\mathbb{R}}$.

Assuming the domain of the function is specified as cartesian product, i.e. the function's input values are represented by tuples. By restricting the domain of the function onto a projection of the cartesian product (see Definition 12), we define the projection of functions.

Definition 15 *[Projection of Functions] Let* $I = \{1, \ldots, n\}$, $f \in \mathbb{FUN}_{\prod_{i\in I} X_i}$ *and*

- *let* $j \in I$, $x \in X_j$ *and* $(a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) \in \prod_{i\in I\setminus\{j\}} X_i$.

Then the function

$$\begin{array}{rcll} f_{[a_1,\ldots,a_{j-1},\$,a_{j+1},\ldots,a_n]} : X_j & \longrightarrow & \mathbb{SET} & with \\ f_{[a_1,\ldots,a_{j-1},\$,a_{j+1},\ldots,a_n]}(x) & = & f(a_1,\ldots,a_{j-1},x,a_{j+1},\ldots,a_n) & \end{array}$$

is called projection of f onto X_j.

- *let* $k \leq n$, $\{j_1,\ldots j_k\} \subseteq I$ *and* $\forall l < k \quad j_l < j_{l+1}$ *with* $(x_{j_1},\ldots,x_{j_k}) \in \prod_{i\in\{j_1,\ldots j_k\}} X_i$.
  and $(a_1,\ldots,a_{j_1-1},a_{j_1+1},\ldots,a_{j_k-1},a_{j_k+1},\ldots,a_n) \in \prod_{i\in I\setminus\{j_1,\ldots j_k\}} X_i$.
  Then the function

$$f_{[a_1,\ldots,a_{j_1-1},\$_1,a_{j_1+1},\ldots,a_{j_k-1},\$_k,a_{j_k+1},\ldots,a_n]} : \prod_{i\in\{j_1,\ldots j_k\}} X_i \longrightarrow \mathbb{SET} \qquad with$$

$$f_{[a_1,\ldots,a_{j_1-1},\$_1,a_{j_1+1},\ldots,a_{j_k-1},\$_k,a_{j_k+1},\ldots,a_n]}(x_{j_1},\ldots,x_{j_k}) =$$
$$f(a_1,\ldots,a_{j_1-1},x_{j_1},a_{j_1+1},\ldots,a_{j_k-1},x_{j_k},a_{j_k+1},\ldots,a_n)$$

is called projection of f onto $\prod_{i\in\{j_1,\ldots j_k\}} X_i$.

Example 10 Let $f \in \mathbb{FUN}_{\mathbb{R}^4}$ with $f(v,w,x,y) = v \cdot w - x \cdot y^2$ then the function

$$f_{[4,\$,7,2]} \in \mathbb{FUN}_{\mathbb{R}} \quad \text{with}$$
$$f_{[4,\$,7,2]}(x) = 4 \cdot x - 7 \cdot 2^2 = 4x - 28$$

defines a projection of f onto $\mathbb{R}$, and the function

$$f_{[4,\$_1,7,\$_2]} \in \mathbb{FUN}_{\mathbb{R}^2} \quad \text{with}$$
$$f_{[4,\$_1,7,\$_2]}(x,y) = 4x - 7y^2$$

defines a projection of f onto $\mathbb{R}^2$.

As predicates are a specific type of function, the projection of predicates are defined in the same way.

In the following definitions, two ways of joining functions will be of importance: Firstly, the product and secondly, the composition of two functions. The product of two functions is based on the product of the two functions's domain values, and returns the product of the corresponding output values.

Definition 16 *[Product of Functions] Let* $x \in X$, $y \in Y$, *and* $f \in \mathbb{FUN}_X$, $g \in \mathbb{FUN}_Y$, *then*

$$\begin{array}{c} f \otimes g \ : \ X \times Y \longrightarrow \mathbb{SET} \times \mathbb{SET} \qquad with \\ f \otimes g \quad (x,y) = f(x) \times g(y) \end{array}$$

the Product of the function f *and* g.

The composition of two functions determines that two functions are applied sequentially, such that the output values of the first function serve as input value for the second function.

Definition 17 *[Composition of Functions] Let* $f, g \in \mathbb{FUN}$, $A, B \in \mathbb{SET}$ *with* $f : A \longrightarrow B$ *and* $g : B \longrightarrow C$*, then*

$$g \circ f \; : \; A \longrightarrow C \qquad with$$
$$g \circ f(a) = g(f(a))$$

the Composition of function f and g.

Example 11 Let $f \in \mathbb{FUN}_{[\mathbb{R}\times\mathbb{R},\mathbb{R}]}$ with $f(x,y) = x + y$ and $g \in \mathbb{FUN}_{[\mathbb{R},\mathbb{R}]}$ with $g(x) = 2x$, then

$$\begin{array}{rcll} f \times g & \in & \mathbb{FUN}_{[\mathbb{R}^3,\mathbb{R}^2]} & \text{with} \\ f \times g(x,y,z) & = & (x+y, 2\cdot z) & \text{such that} \\ f \times g(3,5,7) & = & (8,14) & \end{array}$$

defines the product of the functions f and g, and

$$\begin{array}{rcll} g \circ f & \in & \mathbb{FUN}_{[\mathbb{R}^2,\mathbb{R}]} & \text{with} \\ g \circ f(x,y) & = & g(f(x,y)) = g(x+y) = 2\cdot(x+y) & \text{such that} \\ g \circ f(3,4) & = & 2\cdot(3+4) = 14 & \end{array}$$

defines the composition of function g and f,

As predicates are specific functions, the composition of predicates, i.e. the composition of function and predicates is defined in the same way.

4.1.3 Select and Apply Operator

The $Select$, and $Apply$ operators are very important for the definition of the EMMO Algebra, as they allow to combine already defined operators and predicates of the algebra, and thus allow to build more complex ones.

The operation $Select$ returns to a specified predicate and a specified set, all elements of the specified set, which fulfill the condition of the specified predicate.

Definition 18 *[Select] Let* A *and* $p \in \mathbb{PRE}$*, then we define the operation* $Select : \mathbb{PRE} \times \mathbb{SET} \longrightarrow \mathbb{SET}$ *with*

$$Select(p, A) = \begin{cases} not\ defined & if \quad p \notin \mathbb{PRE}_A \\ \{x \,|\, x \in A \wedge p(x)\} & if \quad p \in \mathbb{PRE}_A \end{cases}$$

Example 12 Let $p \in \mathbb{PRE}_{\mathbb{N}}$ with $p(x) = \begin{cases} true & if \quad x > 7 \\ false & else \end{cases}$
then $Select(p, \mathbb{N}) = \{8, 9, 10, \ldots\}$.

The operation $Apply$ takes a function and a set as input value, and returns a set, which encompasses all return values when applying the function to each element contained in the specified set.

Definition 19 *[Apply] Let* A *and* $f \in \mathbb{FUN}$*, then we define the operation* $Apply : \mathbb{FUN} \times \mathbb{SET} \longrightarrow \mathbb{SET}$ *with*

$$Apply(f, A) = \begin{cases} not\ defined & if \quad f \notin \mathbb{FUN}_A \\ \{f(x) \,|\, x \in A\} & if \quad f \in \mathbb{FUN}_A \end{cases}$$

Example 13 Let $f \in \mathbb{FUN}_{\mathbb{N}}$ with $f(x) = 3x$
then $Apply(f, \mathbb{N}) = \{3, 6, 9, 12, \ldots\}$

The operation $Elements$ can either be based on a set or a sequence, and returns a set. If the operation is based on a set, it returns the set of all elements, that are contained in any of the specified set's elements. If the operation is based on a sequence, it returns a set encompassing all tuples of the specified sequence.

Definition 20 *[Elements] Let A and $A_i, i \geq 1$ be arbitrary sets,*

$Elements : \mathbb{SET} \longrightarrow \mathbb{SET}$ *with*
$Elements(A) = \bigcup_{X \in A} X = \{x \mid \exists X \in A \wedge x \in X\}$

$Elements : \mathbb{SEQ} \longrightarrow \mathbb{SET}$ *with*
$Elements((a_1, a_2, ...)) = \bigcup_{i \geq 1} \pi_i((a_1, a_2, ...)) = \{a_1, a_2, ...\}$

Example 14 Let $A = \mathbb{N}$, $B = \{\{0, 1\}, \{0\}, \{1, 2\}, \{3\}\}$, and $(3, 6, 3, 8) \in \mathbb{SEQ}$ then

$$\begin{aligned} Elements(A) &= A, \\ Elements(B) &= \{0, 1, 2, 3\} \quad \text{and} \\ Elements((3, 6, 3, 8)) &= \{3, 6, 8\} \end{aligned}$$

4.1.4 Basic Predicates

In the following, some basic predicates will be defined. First, we will specify some logical predicates, then some generic comparison and testing predicates.

The logical predicate Not takes one boolean value as input value and returns true, if its input value is false, otherwise true.

Definition 21 *Let $x \in \mathbb{BOO}$, then*

$$Not : \mathbb{BOO} \longrightarrow \mathbb{BOO} \quad \textit{with} \quad Not(x) = \begin{cases} true & if \quad x = false \\ false & else \end{cases}$$

The logical predicate And takes two boolean values as input values and returns true, if both specified input values are true, otherwise false is returned.

Definition 22 *Let $x \in \mathbb{BOO}$, then*

$$And : \mathbb{BOO} \times \mathbb{BOO} \longrightarrow \mathbb{BOO} \quad \textit{with} \quad And(x, y) = \begin{cases} true & if \quad x \wedge y \\ false & else \end{cases}$$

The logical predicate Or takes two boolean values as input values and returns true, if at least one of the specified input values is true, and returns false if both input values are false.

Definition 23 *Let $x \in \mathbb{BOO}$, then*

$$Or : \mathbb{BOO} \times \mathbb{BOO} \longrightarrow \mathbb{BOO} \quad \textit{with} \quad Or(x, y) = \begin{cases} true & if \quad x \vee y \\ false & else \end{cases}$$

The predicate $Exists$ returns true, if the specified set contains an element which satisfies the condition described by the specified predicate, otherwise false is returned.

Definition 24 *Let $A \in \mathbb{SET}$, and $p \in \mathbb{PRE}$ then*
$Exists : \mathbb{SET} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *is defined as*
$$Exists(A,p) = \begin{cases} true & if \quad \exists x \in A \quad p(x) \\ false & else \end{cases}$$

The predicate $Forall$ returns true, if all elements of the specified set satisfy the condition described by the specified predicate, otherwise false is returned.

Definition 25 *Let $A \in \mathbb{SET}$, and $p \in \mathbb{PRE}$ then*
$Forall : \mathbb{SET} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *is defined as*
$$Forall(A,p) = \begin{cases} true & if \quad \forall x \in A \quad p(x) \\ false & else \end{cases}$$

In the following, we will introduce a set of generic predicates - $Contains$, $Equal$, $Empty$, $Subset$ and $SubsetEq$, $Superset$ and $SupersetEq$, $Smaller$ and $SmallerEq$, and finally $Bigger$ and $BiggerEq$, - which allow to compare and examine arbitrary sets and objects.

The predicate $Contains$ investigates whether a specified object is contained in a specified set, in which case the boolean value true is returned. Otherwise, false is returned.

Definition 26 *Let $x \in \mathbb{OBJ}$, and $A \in \mathbb{SET}$ then*
$Contains : \mathbb{OBJ} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ *is defined as*
$$Contains(x,A) = \begin{cases} true & if \quad x \in A \\ false & else \end{cases}$$

The predicate $Equals$ returns true, if two specified objects are equal, otherwise, false is returned.

Definition 27 *Let $a, b \in \mathbb{OBJ}$,then*
$$Equal : \mathbb{OBJ} \times \mathbb{OBJ} \longrightarrow \mathbb{BOO} \quad with \quad Equal(a,b) = \begin{cases} true & if \quad a = b \\ false & else \end{cases}$$

The predicate $Empty$ returns true, if the specified set is an empty set, i.e. contains no elements. Otherwise false is returned.

Definition 28 *Let $A \in \mathbb{SET}$ then*
$$Empty : \mathbb{SET} \longrightarrow \mathbb{BOO} \quad with \quad Empty(A) = \begin{cases} true & if \quad A = \emptyset \\ false & else \end{cases}$$

The predicates $Subset$ and $SubsetEq$ take two sets as input values and return true, if the first set is subset of the second set, or respectively, the first set is equal or subset of the second set. Otherwise, false is returned.

Definition 29 *Let $A, B \in \mathbb{SET}$*

- $Subset : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$Subset(A,B) = \begin{cases} true & if \quad A \subset B \\ false & else \end{cases}$$
- $SubsetEq : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$SubsetEq(A,B) = \begin{cases} true & if \quad A \subseteq B \\ false & else \end{cases}$$

The predicates $Superset$ and $SupersetEq$ are the counterparts to the predicate $Subset$, and $SubsetEq$. Each of it takes as well two sets as input values and returns true, if the first set is superset of the second set, or respectively, the first set is equal or superset of the second set. Otherwise, false is returned.

Definition 30 *Let* $A, B \in \mathbb{SET}$

- $Superset : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$Superset(A,B) = \begin{cases} true & if \quad A \supset B \\ false & else \end{cases}$$
- $SupersetEq : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$SupersetEq(A,B) = \begin{cases} true & if \quad A \supseteq B \\ false & else \end{cases}$$

The predicate $Smaller$ takes two real numbers as input values and returns true, if the first number is smaller than the second number. Otherwise, false is returned. Similar, the predicate $SmallerEq$ takes two real numbers as input values and returns true, if the first number is equal or smaller than the second number. Otherwise, false is returned.

Definition 31 *Let* $x, y \in \mathbb{R}$

- $Smaller : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$Smaller(x,y) = \begin{cases} true & if \quad x < y \\ false & else \end{cases}$$
- $SmallerEq : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$SmallerEq(x,y) = \begin{cases} true & if \quad x \leq y \\ false & else \end{cases}$$

The predicates $Bigger$ and $BiggerEq$ are the counterparts to the predicates $Smaller$ and $SmallerEq$. Each of them takes again two real numbers as input values and returns true, if the first number is bigger than the second number, or respectively is equal or bigger than the second number. Otherwise, false is returned.

Definition 32 *Let* $x, y \in \mathbb{R}$ *then*

- $Bigger : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$Bigger(x,y) = \begin{cases} true & if \quad x > y \\ false & else \end{cases}$$
- $BiggerEq : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{BOO}$ *is defined as*
  $$BiggerEq(x,y) = \begin{cases} true & if \quad x \geq y \\ false & else \end{cases}$$

4.2 EMMA's Operators

In the following subsections, we define the operators of the query algebra EMMA. EMMA basically consists of five different kinds of operators: Extraction Operators (see Sect. 4.2.1) allow the access to each entity's attribute information, Navigational Operators (see Sect. 4.2.2) enable the navigation along the graph structure of an EMMO, Selection Predicates (see Sect. 4.2.3) permit to choose entities with specific characteristics, Constructors (see Sect. 4.2.4) facilitate the construction of new EMMOs, and the Join Operator(see Sect. 4.2.5) relates several entities or EMMOs with a join condition.

The algebra's operators are kept very simple and orthogonal. Through the combination of modular operators, complex queries can be formulated. The combination of operators can be either realized by sequentially applying the operators or by combining the operation's return value via basic set operations as defined in Sect. 4.1. In this way the orthogonality of the algebra can be realized.

4.2.1 Extraction Operators

An entity represents a thirteen-tuple. The Extraction Operators provide means to query all attributes of the thirteen-tuple. The attributes of an entity are specified as elements and as set values. The operators for extracting an entity's element attributes are very simple and straightforward, whereas the operators for accessing an entity's set attributes can become quite elaborate. This is due to the fact, that some of an entity's set attributes describe quite complex structures, e.g. the set of nodes allows to nest entities in arbitrary depth, and the set of successors or predecessors describe a branching structure of succeeding and preceding entity versions. The extraction operators defined in the following provide means to access the hierarchically nested structure established by the encapsulation of EMMOs, as well as the branching structure conveying an entity's different versions.

The following six definitions describe very basic operators for accessing the attributes oid, name, kind, target and source, and types of an entity. As the operators are realized by a simple projection, the definition is very straightforward and does not need further explanation.

An entity can be identified by its oid, which is an universal unique identifier. The operator *oid* returns to a given entity its universal unique identifier ($\mathbb{OID}$).

Definition 33 *[Accessor for entity's oid] Let $w \in \Omega$, then*
$oid : \Omega \longrightarrow \mathbb{OID}$ *is defined as* $oid(w) = o_w$.

An entity can be identified by its name, which is a human readable name. The operator *name* takes an entity and returns the string value representing the entity's name.

Definition 34 *[Accessor for entity's name] Let $w \in \Omega$, then*
$name : \Omega \longrightarrow \mathbb{STR}$ *is defined as* $name(w) = n_w$

Entity's kind information specifies whether an entity is a logical media part, an ontology object, an association, or an EMMO. The operator $kind$ takes an entity and returns the string "lmp" in case of the entity being a logical media object, the string "emm" in case of an EMMO, "asso" in the case of the entity being an association, and "ont" in the case of an ontology object.

Definition 35 *[Accessor for entity's kind] Let $w \in \Omega$, then*
$kind : \Omega \longrightarrow \{"lmp", "emm", "asso", "ont"\}$ *is defined as* $kind(w) = k_w$

Only associations specify its source and target entity, and thus describe the semantic relationship between those two entities. All entities except the association entity, specify their source and target entity as a empty entity, represented by the symbol ϵ. The following two operators $source$ and $target$ access entity's source and target entity, and either return the specified or the empty entity.

Definition 36 *[Accessor for entity's source entity] Let $w \in \Omega$, then*
$source : \Omega \longrightarrow \Omega \cup \{\epsilon\}$ *is defined as* $source(w) = s_w$

Definition 37 *[Accessor for entity's target entity] Let $w \in \Omega$, then*
$target : \Omega \longrightarrow \Omega \cup \{\epsilon\}$ *is defined as* $target(w) = t_w$

Through the Types attribute, each entity associates an arbitrary number of ontology objects describing its real world concept. The operator $types$ takes an entity and retrieves the set of all its attached ontology objects:

Definition 38 *[Accessor for entity's Types] Let $w \in \Omega$, then*
$types : \Omega \longrightarrow \mathcal{P}(\Theta)$ *is defined as* $types(w) = T_w$

Each entity can additionally be qualified by an arbitrary number of attribute values, that are simple attribute-value pairs with the attribute being an ontology object and the value being an arbitrary object. To a given entity the operator $attributes$ returns the set of associated attribute-value pairs:

Definition 39 *[Accessor for entity's Attributes] Let $w \in \Omega$, then*
$attributes : \Omega \longrightarrow \mathcal{P}(\Theta \times \mathbb{OBJ})$ *is defined as* $attributes(w) = A_w$

Logical media parts are the only entities that model media objects on a logical level as well as maintain connections to the media data representing these objects. Thus, all other entities, i.e. EMMOs, ontology objects and associations, specify the connectors as empty set. The connectors link a logical media part to its set of existing physical representations. Each physical representation, the so-called connector, is manifested by a media profile and a media selector. A media profile is defined by its media instance and its set of metadata. Hereby, each media instance either describes the location of, or directly embeds the media data, and the metadata constitutes a set of string-object pairs. A media selector specifies a kind value and a parameter set. The kind attribute can take the values "spatial","textual", "temporal" etc., and the parameter set encompasses any arbitrary string-object pair.

The following operators allow the access to the information described by connectors. Firstly, the operation $connectors$ takes an entity and returns its set of connectors, which is a set of media selector and media profile pairs. The two operators $MediaSelector$ and $MediaProfile$ return to a given connector, the corresponding media selector, and respectively the corresponding media profile. The two operators $MediaInstance$ and $Metadata$ both take a media profile as input value, and return the corresponding media instance, respectively the corresponding set of metadata. And finally, the two operators $kind$ and $Parameter$ return to a given media selector the corresponding kind value, respectively the corresponding set of parameters.

Definition 40 *[Accessor for entity's Connectors] Let $w \in \Omega$, $ms \in \mathcal{MS}$, and $mp \in \mathcal{MP}$, then*
$connectors : \Omega \longrightarrow \mathcal{P}(\mathcal{MS} \times \mathcal{MP})$ *is defined as*
$connectors(w) = C_w$

$MediaSelector : \mathcal{MS} \times \mathcal{MP} \longrightarrow \mathcal{MS}$ *with*
$MediaSelector(ms, mp) = \pi_1(ms, mp) = ms$

$MediaProfile : \mathcal{MS} \times \mathcal{MP} \longrightarrow \mathcal{MP}$ *with*
$MediaProfile(ms, mp) = \pi_2(ms, mp) = mp$

$MediaInstance : \mathcal{MP} \longrightarrow \mathbb{URI} \cup \mathbb{RMD}$ *with*
$MediaInstance(mp) = \pi_1(mp) = i_{mp}$

$Metadata : \mathcal{MP} \longrightarrow \mathcal{P}(\mathbb{STR} \times \mathbb{OBJ})$ *with*
$Metadata(mp) = \pi_2(mp) = M_{mp}$

$kind : \mathcal{MS} \longrightarrow \{"spatial", "textual", "temporal", \dots\}$ *with*
$kind(ms) = \pi_1(ms) = k_{ms}$

$Parameter : \mathcal{MS} \longrightarrow \mathcal{P}(\mathbb{STR} \times \mathbb{OBJ})$ *with*
$Parameter(ms) = \pi_2(ms) = P_{ms}$

As described in Section 3, EMMOs are the only entities that can contain other entities. EMMO's contained entities are described by the attribute nodes. The operator $nodes$ enable the access to the set of all entities contained in the specified EMMO, whereas the operators lmp, emm, $asso$, and ont return all contained entities of a specific kind, i.e. lmp returns the set of all contained logical media parts, emm the set of all contained EMMOs, $asso$ the set of all associations, and ont the set of all contained ontology objects. All five described operators return the empty set, if the specified input entity is not of kind EMMO.

Definition 41 *[Accessor for entity's Nodes] Let $w \in \Omega$, then*

$nodes : \Omega \longrightarrow \mathcal{P}(\Omega)$ *is defined as* $nodes(w) = N_w$
$lmp : \Omega \longrightarrow \mathcal{P}(\Gamma)$ *is defined as* $lmp(w) = N_w \cap \Gamma$
$emm : \Omega \longrightarrow \mathcal{P}(\Sigma)$ *is defined as* $emm(w) = N_w \cap \Sigma$
$asso : \Omega \longrightarrow \mathcal{P}(\Lambda)$ *is defined as* $asso(w) = N_w \cap \Lambda$
$ont : \Omega \longrightarrow \mathcal{P}(\Theta)$ *is defined as* $ont(w) = N_w \cap \Theta$

An EMMO serves as container establishing a graph structure of entities. The contained entities are either interconnected via binary directed semantic relationships described by associations or represent isolated entities. Thus, beside the isolated entities, the smallest building block of the described graph structure is the semantic relationship established by an association pointing to its source and target entity. To a specified entity (which is supposed to be an EMMO), the following two operators $Sources$ and $Targets$ allow the access to the set of all source and the set of all target entities being involved in an relationship contained within a specified EMMO.

Definition 42 *[Accessor for the source and target entities] Let $w \in \Omega$, then*

$Sources : \Omega \longrightarrow \mathcal{P}(\Omega)$ *with* $Sources(w) = \{s \mid \exists x \in asso(w) : \quad s = s_x\}$
$Targets : \Omega \longrightarrow \mathcal{P}(\Omega)$ *with* $Targets(w) = \{t \mid \exists x \in asso(w) : \quad t = t_x\}$

For accessing the directed graph structure of an EMMO, which is described by its contained and possibly connected entities, a navigational access is required. The operators for navigating an EMMO's graph structure use an entity within an EMMO as input value and return all entities that are semantically linked to this selected entity:

Although specifying two entities as input value, the operator $GetSourceToAsso$ only returns a reasonable result, if the first entity constitutes an EMMO and the second entity an association within this EMMO. In such a case the association's source entity, otherwise the empty set is returned. Respectively, the operator $GetTargetToAsso$ returns to a specified EMMO and a specified association within this EMMO, the association's target entity. If the first input entity is no EMMO, or second entity does not belong to the associations of the EMMO, the empty set is returned.

Definition 43 *Let $e, a \in \Omega$, then*

$GetSourceToAsso : \Omega \times \Omega \longrightarrow \Omega \cup \{\emptyset\}$ *with*
$GetSourceToAsso(e, a) = \{s_a \mid a \in asso(e)\}$ *and*

$GetTargetToAsso : \Omega \times \Omega \longrightarrow \Omega \cup \{\emptyset\}$ *with*
$GetTargetToAsso(e, a) = \{t_a \mid a \in asso(e)\}$

The following two accessors facilitate the navigation of the graph structure by returning the connecting associations: Applied to an EMMO, the operator $GetAssoToSource$ locates all associations, whose source entity is equal the second specified input entity. Similarly, the operator $GetAssoToTarget$ finds in an EMMO all associations, whose target entity is equal the second specified input entity. Again, if the first specified input entity is not of kind EMMO and the second specified input entity is no association, then the empty set is returned.

Definition 44 *Let $e, w \in \Omega$, then*

$GetAssoToSource : \Omega \times \Omega \longrightarrow \mathcal{P}(\Lambda)$ *with*
$GetAssoToSource(e, w) = \{x \in asso(e) \mid w = s_x\}$ *and*

$GetAssoToTarget : \Omega \times \Omega \longrightarrow \mathcal{P}(\Lambda)$ *with*
$GetAssoToTarget(e, w) = \{x \in asso(e) \mid w = t_x\}$

EMMOs constitute a structure of hierarchically nested EMMOs, i.e. an EMMO can contain entities and other EMMOs, and those EMMOS can again contain entities and other EMMOs, and so on. We use the following manner-of-speaking:

- If entity w_1 is contained in EMMO w's nodes, then we say w_1 *is contained in EMMO w at first level*, and w is parent EMMO of w_1.
- If and only if there is a sequence of k entities starting with EMMO w and going until entity w_k , such that each entity in sequence is the parent EMMO of the subsequent entity, then we say that *entity w_k is contained in EMMO w at k^{th} level*.
- If entity w is contained in EMMO e at arbitrary level, then we say that *entity w is recursively contained or is encapsulated in EMMO e*

The following operators are defined by means of induction over $\mathbb{N}$ and allow to navigate the encapsulation hierarchy of nested EMMOs. Although their input value is defined on the level of entities, they only return a reasonable result, if they are applied to EMMOs, otherwise the empty set is returned. Thus, the operator *EncEnt* returns to a specified EMMO e and a natural number n the set of EMMO e's nodes at n^{th} level. The operator *AllEncEnt*, which defines a unification of the operator *EncEnt* returns to a specified EMMO all its recursively contained or encapsulated entities.

Definition 45 *Let $e \in \Omega$, then*

$EncEnt : \Omega \times \mathbb{N} \longrightarrow \mathcal{P}(\Omega)$ *with* $EncEnt(e,1) = N_e$

Let $EncEnt(e,n)$ be defined, then
$EncEnt(e,n+1) = \{x \in \Omega \mid \exists y \in EncEnt(e,n) \cap \Sigma \wedge x \in N_y\}$
and

$AllEncEnt : \Omega \longrightarrow \mathcal{P}(\Omega)$ *with* $AllEncEnt(e) = \bigcup_{i \geq 1} EncEnt(e,i)$

The following operators *EncEmm* and *AllEncEmm* describe a special application of the above operators, allowing the access of the set of encapsulated EMMOs. As you can see from the definition, the operators are more or less an application of already existing operators. In this way, one could argue that those two operators are redundant. Nevertheless, we decided to include the two operators in the EMMO Algebra, as they seem to be used very frequently in the user queries.

In the same way, the operator *EncEmm* returns to a specified EMMO e and natural number n the set of EMMOs contained in EMMO e at n^{th} level, and the operator *AllEncEmm* returns to a specified EMMO e all its recursively contained or encapsulated EMMOs.

Definition 46 *Let $e \in \Omega$, then*

$EncEmm : \Omega \times \mathbb{N} \longrightarrow \mathcal{P}(\Omega)$ *with* $EncEmm(e,1) = emm(e)$

Let $EncEmm(e,n)$ be defined, then
$EncEmm(e,n+1) = \{x \in \Omega \mid \exists y \in EncEmm(e,n) \wedge x \in emm(y)\}$
and

$AllEncEmm : \Omega \longrightarrow \mathcal{P}(\Omega)$ *with* $AllEncEmm(e) = \bigcup_{i \geq 1} EncEmm(e,i)$

As you can see from the above definitions, for all entities w and all natural numbers n the operation $EncEmm(w,n)$ is equal to the operation $Select(Contains_{[(\$, EncEnt(e,n))]}, \Sigma)$, and the operation $AllEncEmm(w)$ is equal to the operation $Select(Contains_{[(\$, AllEncEnt(w))]}, \Sigma)$.

Each entity w points to an arbitrary number of preceding versions P_w and to an arbitrary number of succeeding versions S_w. As an entity w's preceding or succeeding version can again specify an arbitrary number of preceding and succeeding versions, all those specified versions are called predecessor, or respectively successor of w. Only the entities directly belonging to P_w, and S_w are called the direct predecessors, respectively direct successors of w. The operators $predecessors$ and $successors$ retrieve the direct predecessors, and respectively direct successors of the specified entity:

Definition 47 *[Accessor for entity's Predecessor and Successor] Let $w \in \Omega$, then*

$predecessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ *is defined as* $predecessors(w) = P_w$

$successors : \Omega \longrightarrow \mathcal{P}(\Omega)$ *is defined as* $successors(w) = S_w$

Based on these operators, the operators for accessing all predecessors will be defined inductively. But beforehand, we will introduce some manner-of-speaking:

- Entity w' is called direct predecessor of entity w, if $w' \in predecessors(w)$.
- Entity w' is called n^{th} predecessor of entity w, if there is a sequence of *n-1* entities, with each entity in the sequence representing the direct predecessor of its subsequent entity.

- Entity w' is called predecessor of entity w, if there exists a natural number n such that w' is n^{th} predecessor of entity w.

The operator $Predecessors$ takes an entity w and a natural number n as input, and returns entity w's set of n^{th} predecessors, and the operator $AllPredecessors$, which unifies operator $Predecessors$'s return values, returns to a specified entity the set of all its predecessors:

Definition 48 *Let $w \in \Omega$, then*

$Predecessors : \Omega \times \mathbb{N} \longrightarrow \mathcal{P}(\Omega)$ *with*
$Predecessors(w, 1) = predecessors(w)$

Let $Predecessors(w, n)$ be defined, then
$Predecessors(w, n + 1) =$
$\{x \in \Omega \mid \exists y \in Predecessors(w, n) \quad x \in predecessors(y)\}$
and

$AllPredecessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ *with*
$AllPredecessors(w) = \bigcup_{i \geq 1} Predecessors(w, i)$

In the same way, the operator for accessing entity's successor versions will be defined: The operator $Successors$ takes an entity w and a natural number n as input, and returns entity w's set of n^{th} successors, and the operator $AllSuccessors$ returns to a specified entity the set of all its successors:

Definition 49 *Let $w \in \Omega$, then*

$Successors : \Omega \times \mathbb{N} \longrightarrow \mathcal{P}(\Omega)$ *with*
$Successors(w, 1) = successors(w)$

Let $Successors(w, n)$ be defined, then
$Successors(w, n + 1) =$
$\{x \in \Omega \mid \exists y \in Successors(w, n) \quad x \in successors(y)\}$
and

$AllSuccessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ *with*
$AllSuccessors(w) = \bigcup_{i \geq 1} Successors(w, i)$

Any entity can be augmented by additional attributes, which are joined in the features F_w of an entity w. The operator $features$ allows the access to this set of primitive attribute-value pairs:

Definition 50 *[Accessor for entity's Features] Let $w \in \Omega$, then*
$features : \Omega \longrightarrow \mathcal{P}(\mathbb{STR} \times \mathbb{OBJ})$ *with* $features(w) = F_w$

EMMOs also can have operations attached. As described before, an operation consists of a designator and an implementation. The designator is specified as ontology object, whereas the implementation can be any mathematical function. The operator $operations$ retrieves the set of all operations, i.e. the set O_w, attached to the entity w, whereas the operator $Designators$ returns the set of designators to entity w's set of operations, the operator $Implementations$ the set of implementations to entity w's set of operations, and the operator $ImpToName$ retrieves to a specified entity and specified designator represented as ontology object, the mathematical function describing the corresponding implementation.

Definition 51 *[Accessor for entity's Operations] Let $w \in \Omega$, then*
$operations : \Omega \longrightarrow \mathcal{P}(\mathcal{O}P)$ *with* $operations(w) = O_w$
$Designators : \Omega \longrightarrow \mathcal{P}(\Theta)$ *with*
$Designators(w) = \{o \in \Theta \,|\, \exists x \in O_w \quad o = \pi_1(x)\}$
$Implementations : \Omega \longrightarrow \mathcal{P}(\mathbb{FUN}_\Sigma)$ *with*
$Implementations(w) = \{f \in \mathbb{FUN}_\Sigma \,|\, \exists x \in O_w \quad f = \pi_2(x)\}$ *and*
$ImpToName : \Omega \times \Theta \longrightarrow \mathbb{FUN}_\Sigma$ *with*
$ImpToName(w, o) = \{f \in \mathbb{FUN}_\Sigma \,|\, \exists x \in O_w \quad o = \pi_1(x) \wedge f = \pi_2(x)\}$

So far, we have specified that an EMMO can carry any kind of functionality. The next step is to define an operation, which allows to execute the functions being attached to an EMMO. The operator $Execute$ takes an EMMO, a function and a sequence of parameters as input value 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 entity, respectively EMMO, or the sequence of parameters constitutes no adequate input value for the specified operation, the empty set is returned.

Definition 52 *[Execute] Let $e \in \Omega$, $op \in \mathcal{O}P$, and $s \in \mathbb{SEQ}$*
$Execute : \Omega \times \mathcal{O}P \times \mathbb{SEQ} \longrightarrow \mathbb{SET}$

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

Sometimes, we might only be interested in some very specific relationships contained in an EMMO. For example, we might be interested in the relationship which is described by an association of a specific type. The operator $AssoOfType$ allows the access to such specific associations. The operator takes two entities e and o as input value, and returns all associations of type o that are contained in e. Through the specification of association's source and target entity, an association incorporates the complete information of the requested relationship. Note, that if the first specified entity e is not of kind EMMO and the second specified entity o is not of kind ontology object, the empty set is returned.

Definition 53 *Let $e, o \in \Omega$, then*
$AssoOfType : \Omega \times \Omega \longrightarrow \mathcal{P}(\Lambda)$ *with*
$AssoOfType(e, o) = \{x \in asso(e) \,|\, o \in types(x)\}$

4.2.2 Navigational Operators

In the following section, we first introduce the *One-Step Navigational Operators*. Then, we provide the definition for the *Regular Path Expressions* over ontology objects, and, finally, we specify the *Regular Navigational Operators*.

One-Step Navigational Operators

EMMOs represent containers of entities. Those entities can be connected by associations establishing a graph structure. By attaching ontology objects, associations are classified by concepts of the ontology. Thus, EMMOs describe a graph structure with edges being classified by ontology objects. In order to allow a

single-step navigation along the graph structure of EMMOs, we define the One-step Navigators *TypedAssoToSource*, *TypedAssoToTarget*, and *TypedAsso*. The operator *TypedAssoToSource* takes three entities e, w, and o as input value and returns the set of all associations of type o contained in EMMO e, whose source entity is equal entity w. The operator *TypedAssoToTarget* takes again three entities e, w, and o as input value and returns the set of all associations of type o contained in EMMO e, whose target entity is equal entity w. Similar, the operator *TypedAsso* takes three entities e, w, and o as input value and returns the set of all associations of type o contained in EMMO e, whose source or target entity is equal entity w. For providing the orthogonality of operators, similar to the other EMMA operators, the three One-Step Navigator are defined on entity level. But nevertheless, they only return a reasonable result, if the first specified entity is of kind EMMO and the third specified entity is of kind ontology object; otherwise, the empty set is returned.

Definition 54 *[One-Step Navigator]For $e, w, o \in \Omega$, the operation*

$TypedAssoToSource : \Omega \times \Omega \times \Omega \longrightarrow \mathcal{P}(\Lambda)$ *with*
$TypedAssoToSource(e, w, o) = \{x \in AssoOfType(e, o) \mid w = s_x\}$

$TypedAssoToTarget : \Omega \times \Omega \times \Omega \longrightarrow \mathcal{P}(\Lambda)$ *with*
$TypedAssoToTarget(e, w, o) = \{x \in AssoOfType(e, o) \mid w = t_s\}$

$TypedAsso : \Omega \times \Omega \times \Omega \longrightarrow \mathcal{P}(\Lambda)$ *with*
$TypedAsso(e, w, o) = TypedAssoToTarget(e, w, o) \cup TypedAssoToTarget(e, w, o)$

Regular Navigational Operators

Regular Navigational Operators allow the navigation of any number of steps within an EMMO's graph structure. An EMMO encompasses a network of entities being connected via associations. The syntax for describing a navigation path is provided by *Regular Path Expressions*, which are specified in Definition 55. By defining the mapping of regular path expressions onto to sequences of ontology objects, the operators *JumpRight* (Def. 56), *JumpLeft* (Def. 57), and *AnchorNodes* (Def. 58) determine the semantic meaning of those syntactic expressions. As different associations can be labelled with the same ontology object, and additionally each navigation path can be matched onto a set of sequences of ontology objects, the navigation path in general describes multiple ways of navigation within the same specified EMMO.

Important to note is the fact, that the *Regular Path Expressions* only provide the syntax for describing an navigation path; its semantic interpretation concerning the navigation within an EMMO, is specified by the operators $JumpRight$, *JumpLeft*, and *AnchorNodes*.

The expressive power of ontology languages strongly influenced the design of regular path expressions. As most ontology languages allow the description of a hierarchical structure of relationships, as well as allow the declaration of properties of relationships, e.g. inverse, symmetric, or transitive relationships, regular path expressions provide means to reflect those constructs:

The simplest kind of regular path expressions is a single ontology object, whose semantic meaning is the ontology object itself, the symbol "ε" referring to the empty set, and the symbol "_" referring to any arbitrary ontology object. The union operator "|" allows the combination of two regular path expressions reflecting that the corresponding sequences of ontology objects can be treated as alternative versions. This is important to cope with the hierarchical structure of relationships within an given

ontology, such that a relationship's subclass can be simply described as alternative version. Furthermore, regular path expressions can be concatenated and be described as optional. Moreover, both unary operators "$+$" and "$*$" defines an iteration of path expressions, which will be interpreted as iteration of the corresponding sequences of ontology objects. This is important for the handling of transitive relationships. Assuming one specifies the retelling relationship to e transitive, i.e. one specifies that a retelling of a retelling of a story, is again a retelling of the original story. Thus, a researcher asking for all objects constituting a retelling of another object, receives all pairs of objects which can be connected by not only one, but any arbitrary sequence of retelling relationships. Finally, with the unary operation "$-$" the inversion of regular path expressions can be expressed. This is substantial for the processing of symmetric relationships. Assuming the relationship "is-parodied-by" constitutes the symmetric version of the relationship "parody-of", and the two objects A and B are connected by an "is-parodied-by" relationship, then one can derive that object B and A are also connected by a "parody-of" relationship. The regular path expressions $o_{parody-of}{}^{-}$ and $o_{is-parodied-by}$ can be labelled as equivalent, such that researchers asking for objects being connected by a "is-parodied-by" relationship, receive all objects being connected by a "is-parodied-by" relationship, as well as a "parody-of" relationship.

Definition 55 *[Regular Path Expression over ontology objects] 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. $\forall r_1, r_2 \in \mathbb{REG} : (r_1 \,|\, r_2) \in \mathbb{REG}$
5. $\forall r_1, r_2 \in \mathbb{REG} : r_1 r_2 \in \mathbb{REG}$
6. $\forall r \in \mathbb{REG} : (r)? \in \mathbb{REG}$
7. $\forall r \in \mathbb{REG} : (r)+ \in \mathbb{REG}$
8. $\forall r \in \mathbb{REG} : (r)* \in \mathbb{REG}$
9. $\forall o \in \Theta : (o)^- \in \mathbb{REG}$

$\mathbb{REG}$ *denotes the set of regular path expression over ontology objects.*

By defining the semantic interpretation of regular path expressions, the operators $JumpRight$ and $JumpLeft$ specify the navigation along an EMMO's graph structure. The navigation within an EMMO reflects the traversing of a sequences of edges within the EMMO's graph structure. Those edges are described by associations which are classified by ontology objects. Thus, each sequence of ontology objects defines uniquely one or more navigation paths within an EMMO. The operators $JumpRight$ and $JumpLeft$ are basically the same and differ only in their direction of navigation. They both specify a start node w within an EMMO e, and a regular path expression r determining the navigation. By induction over the construction of regular path expression, the two operators define a mapping from regular path expressions to sets of sequences of ontology objects. As the names suggest, the operation $JumpRight$ returns

the set of all nodes in the specified EMMO e, that can be reached when starting from entity w by traversing the sequence of labelled associations determined by the regular path expression r in the right direction. In this context, right direction expresses that the the association are traversed from their source to their target entities.

This is appropriate in cases, the requesting user has specified the source entity of an association as starting node. But assuming, the user has selected one particular text document, and is now interested in all documents constituting a retelling of the selected text. By specifying the selected document as start point of the navigation and the regular path expression $o_{retells}*$ determining the navigation path, in this case the traversing has to be accomplished in the left direction (as the selected document establishes the target entity of a retelling association). The operator ***JumpLeft*** returns the set of all nodes in an EMMO e, that can be reached when starting from entity w by traversing the regular path expression r in the left direction, meaning pointing from target to source entity. Note, that again, in order to facilitate the combination of operators, the following operators are defined on entity level, but only return a reasonable value, when applied to EMMOs.

Definition 56 *[JumpRight]For $e, w \in \Omega$, and a regular path expression $r \in \mathbb{REG}$, the operation $JumpRight : \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega)$ is defined as follows:*

1. $\forall o \in \Theta$:

$$JumpRight(e, w, o) = \{x \in N_e \,|\, \exists y \in AssoOfType(e, o) \wedge w = s_y \wedge x = t_y\}$$

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

3. $r = \_$:

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

4. $\forall r_1, r_2 \in \mathbb{REG}$ *with* $r = (r_1 \,|\, r_2)$:

$$JumpRight(e, w, (r_1 \,|\, r_2)) = \bigcup_{x \in \{r_1, r_2\}} JumpRight(e, w, x)$$

5. $\forall r_1, r_2 \in \mathbb{REG}$ *with* $r = r_1 r_2$:

$$JumpRight(e, w, r_1 r_2) = \bigcup_{x \in JumpRight(e, w, r_1)} JumpRight(e, x, r_2)$$

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 means of induction over $\mathbb{N}$ as follows:

$$JR_1(e,w,r) = JumpRight(e,w,r)$$
$$JR_n(e,w,r) = \bigcup_{x \in JR_{n-1}(e,w,r)} JumpRight(e,x,r)$$

8. $\forall r \in \mathbb{REG}$:

$$JumpRight(e,w,(r)*) = \bigcup_{x \in \{(r)+,\varepsilon\}} JumpRight(e,w,x)$$
$$= JumpRight(e,w,(r)+) \cup \{w\}$$

9. $\forall o \in \Theta$:

$$JumpRight(e,w,(o)\text{–}) = \{x \in N_e \,|\, \exists y \in AssoOfType(e,o) \wedge x = s_y \wedge w = t_y\}$$

Definition 57 *[JumpLeft]For $e, w \in \Omega$, and a regular path expression $r \in \mathbb{REG}$, the operation $JumpLeft : \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega)$ is defined as follows:*

1. $\forall o \in \Theta$:

$$JumpLeft(e,w,o) = \{x \in N_e \,|\, \exists y \in AssoOfType(e,o) \wedge x = s_y \wedge w = t_y\}$$

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

3. $r = \_$:

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

4. $\forall r_1, r_2 \in \mathbb{REG}$ *with* $r = (r_1 \,|\, r_2)$:

$$JumpLeft(e,w,(r_1 \,|\, r_2)) = \bigcup_{x \in \{r_1, r_2\}} JumpLeft(e,w,x)$$

5. $\forall r_1, r_2 \in \mathbb{REG}$ *with* $r = r_1 r_2$:

$$JumpLeft(e,w,r_1 r_2) = \bigcup_{x \in JumpLeft(e,w,r_2)} JumpLeft(e,x,r_1)$$

6. $\forall r \in \mathbb{REG}$:

$$JumpLeft(e,w,(r)?) = \bigcup_{x \in \{r,\varepsilon\}} JumpLeft(e,w,x)$$

7. $\forall r \in \mathbb{REG}$:

$$JumpLeft(e,w,(r)+) = \bigcup_{n \geq 1} JL_n(e,w,r)$$

with $JL_n(e,w,r)$ defined by means of induction over $\mathbb{N}$ as follows:

$$JL_1(e,w,r) = JumpLeft(e,w,r)$$
$$JL_n(e,w,r) = \bigcup_{x \in JL_{n-1}(e,w,r)} JumpLeft(e,x,r)$$

8. $\forall r \in \mathbb{REG}$:

$$JumpLeft(e,w,(r)*) = \bigcup_{x \in \{(r)+,\varepsilon\}} JumpLeft(e,w,x)$$
$$= JumpLeft(e,w,(r)+) \cup \{w\}$$

9. $\forall o \in \Theta$:

$$JumpLeft(e,w,(o)\text{–}) = \{x \in N_e \,|\, \exists y \in AssoOfType(e,o) \wedge x = t_y \wedge w = s_y\}$$

As you can see from the last two definition, there is a clear connection between the direction of navigation, and the inversion of ontology objects:
Given the ontology objects o and its symmetric counterpart (o)–, then

$$JumpRight(e,w,o) = JumpLeft(e,w,(o)\text{–})$$

expressing, that the navigation in the right direction determined by one ontology objects, corresponds to the navigation in the left direction determined by its symmetric counterpart.

The idea and design of the operator $AnchorNode$ is very similar to the operators $JumpRight$, and $JumpLeft$. To a given EMMO e and a specified regular path expression r, the operator retrieves all pairs of entities contained in EMMO e, that can be connected by the regular path expression r. Two entities can be connected by a regular path expression, if the sequences of ontology objects determined by the regular path expressions, refers to a sequence of associations connecting the two entities. A sequence of associations connects two entities, if the first entity represent the starting point from which the second entity can be reached by traversing the specified regular path expression in the right direction.

Definition 58 *[AnchorNodes]For $e \in \Omega$ and a regular path expression $r \in \mathbb{REG}$ the operation $AnchorNodes : \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega \times \Omega)$ is defined as follows:*

1. $\forall o \in \Theta$ *with* $r = o$:

$$AnchorNodes(e,o) = \{(x,y) \in \Omega \times \Omega \,|\, \exists a \in AssoOfType(e,o) \wedge x = s_a \wedge y = t_a\}$$

2. $r = \varepsilon$:

$$AnchorNodes(e,\varepsilon) = \{(x,x) \,|\, x \in N_e\}$$

3. $r = \_$:

$$AnchorNodes(e,\_) = \{(x,y) \in \Omega \times \Omega \,|\, \exists a \in asso(e) \wedge x = s_a \wedge y = t_a\}$$

4. $\forall r_1, r_2 \in \mathbb{REG}$ *with* $r = (r_1 \,|\, r_2)$:

$$AnchorNodes(e, (r_1 \,|\, r_2)) = \bigcup_{x \in \{r_1, r_2\}} AnchorNodes(e, x)$$

5. $\forall r_1, r_2 \in \mathbb{REG}$ *with* $r = r_1 r_2$:

$$AnchorNodes(e, r_1 r_2) = \{(x, y) \,|\, \exists z \in N_e \quad ((x, z) \in AnchorNodes(e, r_1) \wedge (z, y) \in AnchorNodes(e, r_2))\}$$

6. $\forall r \in \mathbb{REG}$:

$$AnchorNodes(e, (r)?) = \bigcup_{x \in \{r, \varepsilon\}} AnchorNodes(e, x)$$

7. $\forall r \in \mathbb{REG}$:

$$AnchorNodes(e, (r)+) = \bigcup_{n \geq 1} AN_n(e, r)$$

with $AN_n(e, r)$ *defined by means of induction over* $\mathbb{N}$ *as follows:*

$$\begin{aligned} AN_1(e, r) &= AnchorNodes(e, r) \\ AN_n(e, r) &= \{(x, y) \,|\, \exists z \in N_e \quad ((x, z) \in AN_{n-1}(e, r) \\ &\qquad \wedge (z, y) \in AnchorNodes(e, r))\} \end{aligned}$$

8. $\forall e \in \mathbb{REG}$:

$$AnchorNodes(e, (r)*) = \bigcup_{x \in \{(r)+, \varepsilon\}} AnchorNodes(e, x)$$

9. $\forall r \in \mathbb{REG}$:

$$AnchorNodes(e, (r)\text{–}) = \{(\pi_2(x), \pi_1(x)) \,|\, x \in AnchorNodes(e, r))\}$$

4.2.3 Selection Predicates

The selection predicates allow the selection of only those entities fulfilling a specific characteristic. If an entity fulfils a specific characteristic, the corresponding selection predicates returns the boolean value true. Thus, selection predicates can be joined with the $Select$ operator (Def. 18), which again decides on the basis of the selection predicate's returned boolean value which entities of a given set to select.

We distinguish two different kinds of selection predicates: The *Basic Selection Predicates*, which are based on Extraction Operators (Sect. 4.2.1), and the *Navigational Selection Predicates* which are based on the Navigational Operators (Sect. 4.2.2).

In the following section, we will first introduce Basic Selection Predicates, succeeded by the description of Navigational Selection Predicates.

Basic Selection Predicates

Basic Selection Predicates are dealing with an entity's attribute values (accessible by Extraction Operators).

The selection predicate *HasOid* takes an entity w and an oid s as input value, and returns true if entity w's oid is equal s, otherwise false is returned.

Definition 59 *Let $w \in \Omega$, and $s \in \mathbb{OID}$, then*
$HasOid : \Omega \times \mathbb{OID} \longrightarrow \mathbb{BOO}$ *is defined as*
$HasOid(w, s) = Equal(oid(w), s)$

To a specified entity w and a specified string s, the selection predicate *HasName* returns true, if entity w's name is equal the string s.

Definition 60 *Let $w \in \Omega$, and $s \in \mathbb{STR}$, then*
$HasName : \Omega \times \mathbb{STR} \longrightarrow \mathbb{BOO}$ *with*
$HasName(w, s) = Equal(name(w), s)$

The predicate *IsOfKind* returns true, if the specified entity w is of the kind described by the specified string parameter.

Definition 61 *Let $w \in \Omega$, and $s \in \{"emm", "asso", "ont", "lmp"\}$, then*
$IsOfKind : \Omega \times \{"emm", "asso", "ont", "lmp"\} \longrightarrow \mathbb{BOO}$ *with*
$IsOfKind(w, s) = Equal(kind(w), s)$

There is one selection predicates concerning the source and one selection predicate concerning the target information of an entity. The operator *IsSourceToAsso* returns true, if both, the specified entity w and the specified association a are contained within the nodes of entity e, and additionally, w is source entity of a. Thus, for the operator to return true, it is necessary that entity e is an EMMO, and entity a is an association within EMMO e.

Definition 62 *Let $e, a, w \in \Omega$, then*
$IsSourceToAsso : \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*

$$IsSourceToAsso(e, a, w) = \begin{cases} true & if \quad a \in asso(e) \ \wedge \ w = s_a \\ false & else \end{cases}$$

As you can derive from the above definition, the expression $IsSourceToAsso(e, a, w)$ is equivalent to the expression $Contains(a, GetAssoToSource(e, w))$ and equivalent to the expression $Equal(GetSourceToAsso(e, a), w)$.

Correspondingly, the operator *IsTargetToAsso* returns true, if the specified entity w, as well as the specified association a are contained in EMMO e, and additionally, entity w is target entity of association a. Again the expression $IsTargetToAsso(e, a, w)$ is equivalent to the expression $Contains(a, GetAssoToTarget(e, w))$ and equivalent to the expression $Equal(GetTargetToAsso(e, a), w)$.

Definition 63 *Let $e, a, w \in \Omega$, then*
$IsTargetToAsso : \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*

$$IsTargetToAsso(e, a, w) = \begin{cases} true & if \quad a \in asso(e) \ \wedge \ w = t_a \\ false & else \end{cases}$$

Furthermore, there are two selection predicates concerning an entity's types information. The operator $IsType$ returns true, if the specified entity w associates the specified entity o within its types set (assuming that entity o is of kind ontology object). The operator $EntityContains$ returns true, if the nodes of the specified entity w contain at least one entity which encompasses entity o within its types set (assuming that entity o is an ontology object).

Definition 64 *Let $w \in \Omega$, and $o \in \Omega$, then*
$IsType : \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*
$$IsType(w,o) = \begin{cases} true & if \quad o \in types(w) \\ false & else \end{cases} \qquad \textit{and}$$
$EntityContains : \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*
$$EntityContains(w,o) = \begin{cases} true & if \quad \exists x \in N_w \ \wedge \ o \in types(x) \\ false & else \end{cases}$$

As you can see from the definition above, the expression $EntityContains(w,o)$ is equivalent to the expression $Not(Empty(Select(IsType_{[\$,o]}, N_w)))$.

For applying the predicate *IsType* onto a set of entities, for example asking whether an entity $w \in \Omega$ associates within its types set at least one ontology object of set $O \subseteq \Theta$, the following query expression has to be evaluated:

$$Exists(O, IsType_{[w,\$]}).$$

Moreover, the EMMO Algebra defines two selection predicate concerning an entity's attribute values. The operator $HasAttribute$ takes two entities as input values, and returns true, if the set of the attribute values of the first specified entity contains an attribute-value-pair such that its attribute (represented by an ontology object) is equal the second specified input entity.

Definition 65 *Let $w \in \Omega$, and $o \in \Omega$, then*
$HasAttribute : \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *with*
$$HasAttribute(w,o) = \begin{cases} true & \textit{if} \quad \exists x \in attributes(w) \quad o = \pi_1(x) \\ false & else \end{cases}$$

The operator $HasAttValue$ is more specific than the operator $HasAttribute$. It takes two entities w and o, and a predicate p as input values, and returns true, if entity w contains within its attributes set an attribute-value pair whose attribute (represented by an ontology object) is equal entity o and, additionally, whose value satisfy the predicate p.

Definition 66 *Let $w, o \in \Omega$, and $p \in \mathbb{PRE}$, then*
$HasAttValue : \Omega \times \Omega \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *is defined as*
$$HasAttValue(w,o,p) = \begin{cases} true & \textit{if} \quad \exists x \in attributes(w) \\ & \qquad (o = \pi_1(x) \wedge p(\pi_2(x))) \\ false & else \end{cases}$$

Connectors consist of a media profile and a media selector. Media profiles represent media data. A media profile encompasses low-level metadata describing the media data along with its storage location being described by its media instance. The following three operators define predicates related to an entity's media profile. The operator $HasMediaProfile$ returns true, if the specified media profile is described by a connector of the specified entity.

Definition 67 *Let* $w \in \Omega$, *and* $mp \in \mathcal{MP}$, *then*
$HasMediaProfile : \Omega \times \mathcal{MP} \longrightarrow \mathbb{BOO}$ *with*

$$HasMediaProfile(w, mp) = \begin{cases} true & \textit{if} \quad \exists c \in C_w \quad mp = MediaProfile(c) \\ false & else \end{cases}$$

The operator $HasMediaProfileValue$ is more specific concerning the condition described by the specified predicate. The operator $HasMediaProfileValue$ defines three input parameter, i.e. an entity w, a string value s, and a predicate p, and returns true, if the entity w associates a media profile, whose set of metadata encompasses a name-value pair, with the name being equal s and the value satisfying the condition described by the specified predicate p.

Definition 68 *Let* $w \in \Omega$,$s \in \mathbb{STR}$, *and* $p \in \mathbb{PRE}$ *then*
$HasMediaProfileValue : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *with*

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

The operator $HasMediaInstance$ takes two input parameter - the entity w and the string d. It returns true, if entity w associates at least one media instance that is equal d.

Definition 69 *Let* $w \in \Omega$, *and* $d \in \mathbb{URI} \cup \mathbb{RMD}$, *then*
$HasMediaInstance : \Omega \times \mathbb{URI} \cup \mathbb{RMD} \longrightarrow \mathbb{BOO}$ *with*

$$HasMediaInstance(w, d) = \begin{cases} true & \textit{if} \quad \exists c \in C_w \\ & \qquad d = MediaInstance(MediaProfile(c)) \\ false & else \end{cases}$$

Media selectors described within an entity's connector specify the media data represented by the media profile according to their textual, spatial, and temporal selection. In this way, a media selector delineates a value describing the kind of selection, and further specifies a set of parameters specifying the kind of selection in more detail. The following two operators define selection predicates related to entity's media selectors. The operator $HasMediaSelector$ returns true, if the specified media selector is described by a connector of the specified entity.

Definition 70 *Let* $w \in \Omega$, *and* $ms \in \mathcal{MS}$, *then*
$HasMediaSelector : \Omega \times \mathcal{MS} \longrightarrow \mathbb{BOO}$ *with*

$$HasMediaSelector(w, ms) = \begin{cases} true & \textit{if} \quad \exists c \in C_w \quad ms = MediaSelector(c) \\ false & else \end{cases}$$

The operator $HasMediaSelectorValue$ is more concerned about media selector's specified parameters. Similar to the operator $HasMediaProfileValue$, the operator $HasMediaSelectorValue$ takes three input values - an entity w, a string value s, and a predicate p -, and returns true if entity w associates a media selector, whose set of parameters encompasses a name-value pair, with a name being equal s and a value satisfying the condition described by the specified predicate p.

Definition 71 *Let* $w \in \Omega$,$s \in \mathbb{STR}$, *and* $p \in \mathbb{PRE}$ *then*
$HasMediaSelectorValue : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *with*

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

The operator $HasMediaWithValue$ is a combination of the two operators $HasMediaProfileValue$ and $HasMediaSelectorValue$. Assuming a user is only interested in some information about the attributes specifying the media data, but the user does not necessarily know, whether this information is stored within the media profile or the media selectors, then the operator $HasMediaWithValue$ allows him to query the attribute information without knowing such model specific information.

Definition 72 *Let* $w \in \Omega$,$s \in \mathbb{STR}$, *and* $p \in \mathbb{PRE}$ *then*
$HasMediaWithValue : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *with*
$HasMediaWithValue(w,s,p) =$
$HasMediaSelectorValue(w,s,p) \vee HasMediaProfileValue(w,s,p)$

The operator $ContainsNode$ is concerned about the nodes of an EMMO. It returns true, if at least one entity belonging to the specified set of entities is contained in the nodes of the specified entity.

Definition 73 *Let* $w \in \Omega$,$W \in \mathcal{P}(\Omega)$,*then*
$ContainsNode : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsNode(w,W) = \begin{cases} true & \textit{if} \quad \exists x \in (W \cap N_w) \\ false & else \end{cases}$$

The following four operators incorporate the predecessor and successor information of an entity. Thus, the operator $ContainsDirectPredecessor$ takes an entity w and a set of entities W as input values and returns true if at least one entity of W is a direct predecessor of entity w.

Definition 74 *Let* $w \in \Omega$, $W \in \mathcal{P}(\Omega)$,*then*
$ContainsDirectPredecessor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsDirectPredecessor(w,W) = \begin{cases} true & \textit{if} \quad \exists x \in (W \cap P_w) \\ false & else \end{cases}$$

Again, from the definition can be derived that the expressions $ContainsDirectPredecessor(w,W)$ and $Not(Empty(W \cap P_w))$ are equivalent.

The operator $ContainsPredecessor$ takes the same input values as the operator $ContainsDirectPredecessor$, and returns true if at least one entity of W belongs to the predecessors of entity w.

Definition 75 *Let* $w \in \Omega$, $W \in \mathcal{P}(\Omega)$,*then*
$ContainsPredecessor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsPredecessor(w,W) = \begin{cases} true & \textit{if} \quad \exists x \in (W \cap AllPredecessors(w)) \\ false & else \end{cases}$$

Again, the equivalence of the two expressions $ContainsPredecessor(w,W)$ and $Not(Empty(W \cap AllPrecessors(w)))$ can be followed.

The operator $ContainsDirectSuccesssor$ takes an entity w and a set of entities W as input values, and returns true if at least one entity of W is direct successor of w.

Definition 76 *Let* $w \in \Omega$, $W \in \mathcal{P}(\Omega)$,*then*
$ContainsDirectSuccesssor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsDirectSuccessor(w, W) = \begin{cases} true & \textit{if} \quad \exists x \in (W \cap S_w) \\ false & else \end{cases}$$

The operator $ContainsSuccessor$ takes the same input values as the operator $ContainsDirectSuccessor$, and returns true if at least one entity of W is successor of entity w.

Definition 77 *Let* $w \in \Omega$,$W \in \mathcal{P}(\Omega)$,*then*
$ContainsSuccessor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsSuccessor(w, W) = \begin{cases} true & \textit{if} \quad \exists x \in (W \cap AllSuccessors(w)) \\ false & else \end{cases}$$

Again, the last two definitions show that the two expressions $ContainsDirectSuccessor(w, W)$ and $Not(Empty(W \cap S_w))$, as well as the two expressions $ContainsSuccessor(w, W)$ and $Not(Empty(W \cap AllSuccessors(w)))$ are equivalent.

Entity's attribute features represent a fixed set of primitive attribute-value pairs. The operator $HasFeature$ is based on the extraction operator $features$ which realizes the access to this set of primitive attribute-value pairs, and returns true, if there exists an attribute-value pair with the specified name and a value fulfilling the condition described by the specified predicate.

Definition 78 *Let* $w \in \Omega$,$s \in \mathbb{STR}$, *and* $p \in \mathbb{PRE}$ *then*
$HasFeature : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *is defined as*

$$HasFeature(w, s, p) = \begin{cases} true & \textit{if} \quad \exists k \in features(w) \\ & \qquad (\pi_1(k) = s \wedge p(\pi_2(k))) \\ false & else \end{cases}$$

In the EMMO model, an operation is described as tuple combining an ontology object with an arbitrary mathematical function. The ontology object is acting as the operation's designator, whereas the mathematical function represents the operation's implementation. The operator $HasDesignator$ returns true, if the first specified entity describes an operation, whose designator (represented as ontology object) is equal the second specified entity.

Definition 79 *Let* $w, o \in \Omega$, *then*
$HasDesignator : \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*

$$HasDesignator(w, o) = \begin{cases} true & \textit{if} \quad o \in Designators(w) \\ false & else \end{cases}$$

The operator $HasImplementation$ takes an entity w and a mathematical function f as input value, and returns true, if entity w specifies an operation with the implementation f

Definition 80 *Let* $w \in \Omega$,*and* $f \in \mathbb{FUN}_{\Sigma}$, *then*
$HasImplementation : \Omega \times \mathbb{FUN}_{\Sigma} \longrightarrow \mathbb{BOO}$ *is defined as*

$$HasImplementation(w, f) = \begin{cases} true & \textit{if} \quad f \in Implementations(w) \\ false & else \end{cases}$$

Navigational Selection Predicates

The following section introduces the Navigational Selection Predicates. Basically all operators defined in Section 4.2.2 will be used in the following definitions.

The operators $ContainsTypedAsso$, $ContainsTypedAssoToSource$ and $ContainsTypedAssoToTarget$ are very similar and vary only in their direction of navigation. Each of the three operators takes three entities e, w, and o as input value. The operator $ContainsTypedAsso$ returns true, if entity e if of kind EMMO, entity o is of kind ontology object, and entity (EMMO) e contains an association of type o with target or source entity equal w.

Definition 81 *Let $e, w, o \in \Omega$, then*
$ContainsTypedAsso : \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsTypedAsso(e,w,o) = \begin{cases} true & \textit{if} \quad \exists x \in asso(e) \\ & \qquad (o \in types(x) \ \wedge \ w \in \{s_x, t_x\}) \\ false & else \end{cases}$$

The operator $ContainsTypedAssoToSource$ only returns true, if entity o is an ontology object, and entity e is an EMMO, which contains an association of type o with source entity equal entity w.

Definition 82 *Let $e, w, o \in \Omega$, then*
$ContainsTypedAssoToSource : \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsTypedAssoToSource(e,w,o) = \begin{cases} true & \textit{if} \quad \exists x \in asso(e) \\ & \qquad (o \in types(x) \ \wedge \ w = s_x) \\ false & else \end{cases}$$

And finally, the operator $ContainsTypedAssoToTarget$ returns true, if entity o is an ontology object and entity e is an EMMO which contains an association of type o with target entity equal entity w.

Definition 83 *Let $e, w \in \Omega$, and $o \in \Theta$, then*
$ContainsTypedAssoToTarget : \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsTypedAssoToTarget(e,w,o) = \begin{cases} true & \textit{if} \quad \exists x \in asso(e) \\ & \qquad (o \in types(x) \ \wedge \ w = t_x) \\ false & else \end{cases}$$

The above definitions manifest the equivalence of the expressions $ContainsTypedAsso(e,w,o)$ and $Not(Empty(TypedAsso(e,w,o)$, the expressions $ContainsTypedAssoToSource(e,w,o)$ and $Not(Empty(TypedAssoToSource(e,w,o)$, and the expressions $ContainsTypedAssoToTarget(e,w,o)$ and $Not(Empty(TypedAssoToTarget(e,w,o)$.

By specifying four entities e, w, o, and a as input value, the following three operators constitute a more specific version of the just described operators. Thereby, the operator $IsTypedAsso$ returns true, if entity a is an association of type o which is contained in the nodes of entity e (requiring entity e to be an EMMO) and specifies entity w either as target or as source entity.

Definition 84 *Let $e, w, o, a \in Wm$, then*
$IsTypedAsso : \Omega \times \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *with*

$$IsTypedAsso(e,w,o,a) = \begin{cases} true & \textit{if} \quad a \in asso(e) \ \wedge \ o \in types(x) \ \wedge \ w \in \{s_a, t_a\} \\ false & else \end{cases}$$

The operator $IsTypedAssoToSource$ returns true, if entity a is an association of type o contained in the nodes of entity (EMMO) e and specifying entity w as source entity.

Definition 85 *Let* $e, w, o, a \in \Omega$*, then*
$IsTypedAssoToSource : \Omega \times \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *with*
$$IsTypedAssoToSource(e, w, o, a) = \begin{cases} true & \text{if} \quad a \in asso(e) \ \wedge \ o \in types(x) \ \wedge \ w = s_a \\ false & else \end{cases}$$

The operator $IsTypedAssoToTarget$ returns true, if entity a is an association of type o which is contained in the nodes of entity (EMMO) e and specifies entity w as target entity.

Definition 86 *Let* $e, w, o, a \in \Omega$*, then*
$IsTypedAssoToTarget : \Sigma \times \Omega \times \Omega \times \Omega \longrightarrow \mathbb{BOO}$ *with*
$$IsTypedAssoToTarget(e, w, o, a) = \begin{cases} true & \text{if} \quad a \in asso(e) \ \wedge \ o \in types(x) \ \wedge \ w = t_a \\ false & else \end{cases}$$

Again, the expressions $IsTypedAsso(e, w, o, a)$ and $Contains(a, TypedAsso(e, w, o))$, the expressions $IsTypedAssoToSource(e, w, o, a)$ and $Contains(a, TypedAssoToSource(e, w, o))$, and the expressions $IsTypedAssoToTarget(e, w, o, a)$ and $Contains(a, TypedAssoToTarget(e, w, o))$ are equivalent.

Both operators $IsRightOf$ and $IsLeftOf$ take three entities e, w_1 and w_2, and a regular path expression $r \in \mathbb{REG}$ as input value. The operator $IsRightOf$ returns true, if entity e is an EMMO containing two entities w_1, and w_2, such that the navigation along the regular path expression r in the right direction and with start point w_1 yields entity w_2.

Definition 87 *Let* $e, w_1, w_2 \in \Omega$ *and* $r \in \mathbb{REG}$ *then*
$IsRightOf : \Omega \times \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathbb{BOO}$ *is defined as*
$$IsRightOf(e, w_1, w_2, r) = \begin{cases} true & \text{if} \quad w_2 \in JumpRight(e, w_1, r) \\ false & else \end{cases}$$

The operator $IsLeftOf$ returns true, if entity e is an EMMO containing two entities w_1, and w_2, such that the navigation along the regular path expression r in the left direction and with start point w_1 yields entity w_2.

Definition 88 *Let* $e, w_1, w_2 \in \Omega$ *and* $r \in \mathbb{REG}$ *then*
$IsLeftOf : \Omega \times \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathbb{BOO}$ *is defined as*
$$IsLeftOf(e, w_1, w_2, r) = \begin{cases} true & \text{if} \quad w_2 \in JumpLeft(e, w_1, r) \\ false & else \end{cases}$$

Easily can be derived from the Definitions 87 and 88 that the two expressions $IsRightOf(e, w_1, w_2, r)$ and $Contains(w_2(JumpRight(e, w_1, r)))$, as well as the two expressions $IsLeftOf(e, w_1, w_2, r)$ and $Contains(w_2(JumpLeft(e, w_1, r)))$ are equivalent.

The operator $ContainsExpr$ is based on the operator $AnchorNodes$. It takes an entity e and a regular path expression r as input value, and returns true, if entity e is of kind EMMO containing a pair of nodes, such that each pair's second node can be reached starting from the pair's first node by traversing along the specified regular expression r in right direction.

Definition 89 *Let* $e \in \Omega$, *and* $r \in \mathbb{REG}$ *then*
$ContainsExpr : \Omega \times \mathbb{REG} \longrightarrow \mathbb{BOO}$ *is defined as*

$$ContainsExpr(e,r) = \begin{cases} true & \text{if} \quad \exists w_1, w_2 \in N_w \quad (w_1, w_2) \in AnchorNodes(e,r) \\ false & else \end{cases}$$

From Definition 89 follows the equivalence of the expressions $ContainsExpr(e,r)$ and $Not(Empty(AnchorNodes(e,r)))$

4.2.4 Constructors

In the following section, we introduce five so-called Constructors for EMMOs: $Union$, $Nest$, $Flatten$, $Difference$, and $Intersection$. These operators have all in common that they take at least one EMMO, and possibly other parameter as input value, and return exactly one EMMO as output value. The Constructors allow to combine and modify EMMOs, and thus to build new EMMOs. On the basis of Constructors, the closure property of the algebra can be accomplished. New EMMOs can either be constructed through the combination, i.e. union, intersection, etc. of already existing EMMOs, or be constructed by nesting extracted data into a new EMMO knowledge structure, the extracted data possibly arising from the result values of other operators of the algebra.

The Constructor $Union$ allows the unification of two different EMMOs. It takes two EMMOs and a string as input values, and returns an EMMO e_{new} with a newly generated unique oid, whereby the nodes of the returned EMMO represent the union of the nodes of the two specified input EMMOs. The EMMO e_{new}'s name is equal the specified input string, and its source and target entities are specified as ε (representing the empty entity). Furthermore, the remaining attribute sets of the new EMMO, i.e. the set of types, attribute values, connectors, predecessors, successors, features and operations, are specified as empty sets.

Definition 90 *Let* $e_1, e_2 \in \Sigma$, $s \in \mathbb{STR}$, *then*
$Union : \Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *with* $Union(e_1, e_2, s) = e_{new}$ *is defined as*

$o_{e_{new}} \in \mathbb{OID}$
$n_{e_{new}} = s$
$k_{e_{new}} =$ "emm"
$s_{e_{new}} = t_{e_{new}} = \varepsilon$
$N_{e_{new}} = nodes(e_1) \cup nodes(e_2)$ *and*
$\pi_i(e_{new}) = \emptyset \quad \forall i \in \{6, 7, 8, 10, 11, 12, 13\}$

To a given set of associations and a specified EMMO, the operator $Nest$ allows the generation of a new EMMO containing exactly those relationships described by the specified set of associations that are also contained in the specified EMMO. Thus, the operator $Nest$ takes an EMMO e, a set of associations A, and a string value s as input value, and returns EMMO e_{new} with a newly generated unique oid. The nodes of the new EMMO encompass all associations belonging to both, the specified set of associations A and nodes of EMMO e. Furthermore, the nodes of the new EMMO contain all source and target entities of the before selected associations. The name of the new EMMO is equal the specified input string, its source and target entities are specified as ε (representing the empty entity), and its remaining attribute sets, i.e. the set of types, attribute values, connectors, predecessors, successors, features and operations, are specified as empty sets.

Definition 91 *Let $e \in \Sigma$, $A \subseteq \Lambda$, and $s \in \mathbb{STR}$, then*
$Nest : \Sigma \times \mathcal{P}(\Lambda) \times \mathbb{STR} \longrightarrow \Sigma$ *with* $Nest(e, A, s) = e_{new}$ *is defined as*

$o_{e_{new}} \in \mathbb{OID}$
$n_{e_{new}} = s$
$k_{e_{new}} = "emm"$
$s_{e_{new}} = t_{e_{new}} = \varepsilon$

$N_{e_{new}} = (A \cap nodes(e)) \cup \{s_x \mid x \in A \cap nodes(e)\}$
$\cup \{t_x \mid x \in A \cap nodes(e)\}$ *and*
$\pi_i(e_{new}) = \emptyset \quad \forall i \in \{6, 7, 8, 10, 11, 12, 13\}$

To a specified EMMO e and string s, the operator $Flatten$ allows the construction of a new EMMO whose set of nodes contains all entities, which are recursively contained in EMMO e. The operator $Flatten$ takes an EMMO e and a string s as input value and returns the EMMO e_{new}, whose nodes encompass all entities that are recursively contained in the specified EMMO. Similar to the constructors defined before, the new EMMO is suited with a newly generated unique oid, named s, defines its source and target entities as empty entities, and assigns its remaining attribute sets, i.e. the set of types, attribute values, connectors, predecessors, successors, features and operations, as empty sets.

Definition 92 *Let $e \in \Sigma$ and $s \in \mathbb{STR}$, then*
$Flatten : \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *with* $Flatten(e, s) = e_{new}$ *is defined as*

$o_{e_{new}} \in \mathbb{OID}$
$n_{e_{new}} = s$
$k_{e_{new}} = "emm"$
$s_{e_{new}} = t_{e_{new}} = \varepsilon$
$N_{e_{new}} = AllEncEnt(e)$ *and*
$\pi_i(e_{new}) = \emptyset \quad \forall i \in \{6, 7, 8, 10, 11, 12, 13\}$

The operator $Difference$ allows the computation of the difference of two input EMMOs. The difference is determined by the difference of the set of nodes of the two input EMMOs, such that the returned EMMO contains all nodes which are contained in the first, but not in the second EMMO. Additionally, for each association that only belongs to the nodes of the first EMMO, its source and target entity is added to the nodes of the new EMMO. Again, the returned EMMO e_{new} is supplied with a new unique oid and labeled with the specified input string s, defines its source and target entities as empty entity, and assigns its remaining attribute sets, i.e. the set of types, attribute values, connectors, predecessors, successors, features and operations, as empty sets.

Definition 93 *Let $e_1, e_2 \in \Sigma$ and $s \in \mathbb{STR}$ then*
$Difference : \Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *with* $Difference(e_1, e_2, s) = e_{new}$ *is defined as*

$o_{e_{new}} \in \mathbb{OID}$
$n_{e_{new}} = s$
$k_{e_{new}} = "emm"$
$s_{e_{new}} = t_{e_{new}} = \varepsilon$

$N_{e_{new}} = \; nodes(e_1) \backslash nodes(e_2) \cup \{s_x \,|\, x \in asso(e_1) \backslash asso(e_2)\}$
$\quad\quad\quad\quad \cup \{t_x \,|\, x \in asso(e_1) \backslash asso(e_2)\}$ *and*
$\pi_i(e_{new}) = \emptyset \quad \forall i \in \{6, 7, 8, 10, 11, 12, 13\}$

The operator $Intersection$ allows the computation of the intersection of two specified input EMMOs. The $Intersection$ of two EMMOs is determined by the intersection of its contained nodes, such that the returned EMMO's nodes are the entities contained in both specified EMMOs. Note, that by computing the intersection of two EMMOs, for each association belonging to both EMMOs, we can deduce from entity's definition, that its source and target entity belong to the intersection as well. Similar to the four other Constructors, the returned EMMO e_{new} is supplied with a new unique oid and is labeled with the specified input string s, defines its source and target entities as empty entity, and assigns its remaining attribute sets, i.e. the set of types, attribute values, connectors, predecessors,successors, features and operations, as empty sets.

Definition 94 *Let* $e_1, e_2 \in \Sigma$ *and* $s \in \mathbb{STR}$ *then*
$Intersection : \Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *with* $Intersection(e_1, e_2, s) = e_{new}$ *is defined as*

$o_{e_{new}} \in \mathbb{OID}$
$n_{e_{new}} = s$
$k_{e_{new}} = "emm"$
$s_{e_{new}} = t_{e_{new}} = \varepsilon$
$N_{e_{new}} = nodes(e_1) \cap nodes(e_2)$
$\pi_i(e_{new}) = \emptyset \quad \forall i \in \{6, 7, 8, 10, 11, 12, 13\}$

4.2.5 Join Operator

In the following section, we introduce the Join Operator for entities specifying how to relate sets of entities in the query. The Join operator takes a set of entity sets, a set of functions and a predicate as input value, and selects only those entities contained in the first specified entity set, which fulfil a condition considering all specified input values. In this way, the Join operator is a generalization of the Select operator accounting for a join condition over not only one but a set of entity sets.

Definition 95 *[Join]Let* $i \in I = \{1, \ldots n\}, W_i \subseteq \Omega, R_i \in \mathbb{SET}, f_i \in \mathbb{FUN}_{[W_i,R_i]}$ *and* $p \in \mathbb{PRE}_{\prod_{i\in I} R_i}$ *then we define the operation*

$Join : \prod_{i\in I} \mathcal{P}(\Omega) \times \prod_{i\in I} \mathbb{FUN}_{[\Omega,\mathbb{SET}]} \times \mathbb{PRE}_{\mathbb{SEQ}_n} \longrightarrow \mathbb{SET}$ *with*

$Join(W_1, \ldots, W_n, f_1, \ldots, f_n, p) =$
$\{\pi_1(w_1, \ldots, w_n) \mid w_i \in W_i \wedge i \in I \wedge p(f_1(w_1), \ldots, f_n(w_n))\}$

After computing the cartesian product of the set of all specified entity sets, the Join operator evaluates for each tuple of the product whether the tuple which is achieved by applying the specified sequence of functions to the sequence of elements in the tuple fulfils the condition described by the predicate. If this is the case, the projection of the first element of the original tuple value is selected.

Bibliography

[1] D. Brickely and R.V. Guha. Resource Description Framework (RDF) Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, World Wide Web Consortium (W3C), April 2002.

[2] ISO/IEC JTC 1/SC 29/WG 11. Information Technology – Multimedia Content Description Interface – Part 5: Multimedia Description Schemes. ISO/IEC Final Draft International Standard 15938-5:2001, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), October 2001.

[3] P.J Leach. UUIDs and GUIDs. Network Working Group Internet-Draft, The Internet Engineering Task Force (IETF), February 1998.