DISSERTATION

A Query Algebra for Ontology-enhanced Management of Multimedia Meta Objects

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von

ao.Univ.-Prof. Dr. Werner Winiwarter

Institutsnummer 394
am Institut für Scientific Computing
der Universität Wien

Zweitgutachter

Univ.-Prof. Dr. Christian Breiteneder

Institutsnummer E188
am Institut für Softwaretechnik und Interaktive Systeme
der Technischen Universität Wien

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von

Dipl.-Math. Sonja Zillner

Matrikelnummer 0127220
Mittersteig 2/26
1040 Wien, Österreich

Wien, am 21.03.2005 [illegible]

Zusammenfassung

Existierende Beschreibungsformate für multimediale Inhalte kodieren in erster Linie die Präsentation des Inhalts, vernachlässigen dabei aber die Informationen, welche der Inhalt vermittelt. Diese präsentationsorientierte Beschreibung multimedialer Inhalte ermöglicht allerdings nur die inflexible, statische Darstellung multimedialer Inhalte. Für die Realisierung anspruchsvollerer Operationen, wie Zugriff auf und Wiederverwendung von Inhalten, automatische Generierung oder Personalisierung, müssen die multimedialen Inhalte mit zusätzlichem semantischem Wissen, z.B. den semantischen Beziehungen zwischen einzelnen multimedialen Objekten, angereichert werden.

Als Grundlage für die semantische Modellierung multimedialer Inhalte in verteilten, kollaborativen Anwendungen wurden im Rahmen dieser Dissertation Enhanced Multimedia Meta Objects (EMMOs) entwickelt. Ein EMMO stellt eine abgeschlossene Einheit multimedialen Inhalts dar, die drei Aspekte des Inhalts untrennbar vereinigt. Der Medienaspekt reflektiert, dass ein EMMO eine Aggregation der Basismedienobjekte darstellt, aus denen der multimediale Inhalt besteht, der semantische Aspekt ermöglicht die Beschreibung von semantischen Assoziationen zwischen den Medienobjekten des EMMOs und der funktionale Aspekt erlaubt es einem EMMO, beliebige, anwendungsspezifische Funktionen zu spezifizieren, die von externen Applikationen aufgerufen werden können. Zusätzlich dazu sind EMMOs versionierbar, d.h. sie können in einer verteilten Umgebung gemeinsam bearbeitet werden, und transferierbar, d.h. die drei Aspekte des multimedialen Inhalts und die Versionierungsinformationen können in eine Einheit gebündelt und in ein austauschbares Format serialisiert werden.

Um einen effizienten Zugriff auf EMMOs zu ermöglichen, wurde in der vorliegenden Arbeit die Abfragealgebra EMMA entwickelt, welche adäquat und vollständig in Bezug auf das EMMO-Modell ist. Indem einfache und orthogonale Operatoren zur Verfügung gestellt werden, die für die Formulierung komplexerer Abfragen kombiniert werden können, wird die Voraussetzung für effiziente Abfrageoptimierung in EMMA geschaffen. EMMA definiert fünf Klassen von Abfrageoperatoren. Die Extraktionsoperatoren ermöglichen den Zugriff auf alle Attribute der Entitäten im EMMO-Modell, die Navigationsoperatoren die Navigation entlang der semantischen Graphstruktur eines EMMOs, die Konstruktoren die Modifikation, Kombination und Erzeugung von neuen EMMOs, die Selektionsprädikate die Auswahl von Entitäten, die eine bestimmte Eigenschaft erfüllen, und der Verbundoperator die Zusammenführung mehrerer Entitäten oder EMMOs durch eine Verbundbedingung.

Sowohl das EMMO-Modell als auch die EMMA-Algebra stellen eine solide Basis für die Integration von ontologischem Wissen dar. Im Rahmen dieser Dissertation wird gezeigt, wie man ontologisches Wissen während der Entwicklung von EMMOs einsetzen, in EMMO-Wissensstrukturen integrieren und für die Verfeinerung von EMMA-Abfragen auswerten kann.

Nachdem die theoretische Grundlage erarbeitet wurde, wird in Folge die Realisierung der EMMO-Container-Infrastruktur erläutert, welche Plattformunabhängigkeit und Skalierbarkeit unterstützt, Export- und Importmöglichkeiten bietet und Werkzeuge für die Darstellung und Entwicklung von EMMOs zur Verfügung stellt. Es werden weiters das Design und die Implementierung der EMMA-Abfrageverarbeitungsarchitektur beschrieben und Evaluierungsergebnisse diskutiert.

Abschließend zeigt die Arbeit anhand von drei Anwendungsszenarien, wie das EMMO-Modell praktisch eingesetzt werden kann und welche Vorteile daraus resultieren. Das erste Szenario ist die Entwicklung einer Plattform für den Austausch kulturellen Wissens, das zweite Szenario beschäftigt sich mit dem Bereich eLearning und das dritte Szenario stellt ein multimediales Task-Management-System vor. Alle drei Anwendungsszenarien haben gemeinsam, dass sie eine Infrastruktur benötigen, welche die verteilte und kollaborative Entwicklung von multimedialen Inhalten ermöglicht. Das erste Szenario ist bereits erfolgreich umgesetzt worden, die Realisierung der beiden anderen Szenarien ist Gegenstand zukünftiger Forschung.

Abstract

Today's multimedia content formats primarily encode the presentation of content but not the information the content conveys. However, this presentation-oriented modeling only permits the inflexible, hard-wired presentation of multimedia content. For the realization of advanced operations like the retrieval and reuse of content, automatic composition, or adaptation to a user's needs, the multimedia content has to be enriched by additional semantic information, e.g. the semantic interrelationships between single multimedia content items.

To provide a basis for the semantic modeling of multimedia content in content sharing and collaborative applications, we have developed Enhanced Multimedia Meta Objects (EMMOs). An EMMO constitutes a self-contained piece of multimedia content that indivisibly unites three of the content's aspects. The media aspect reflects that an EMMO aggregates the basic media objects of which the multimedia content consists, the semantic aspect allows the specification of semantic associations between an EMMO's media objects, and, finally, the functional aspect provides means for the definition of arbitrary, domain-specific operations on the content that can be invoked by applications. Furthermore, EMMOs are versionable – they can be modified concurrently in a distributed environment – and tradeable, i.e. all three aspects of the multimedia content and the versioning information can be bundled into one unit and serialized into an exchangeable format.

To enable the efficient retrieval of EMMOs, we have developed the query algebra EMMA, which is adequate and complete with regard to the EMMO model. By providing simple and orthogonal operators, which can be combined to formulate more complex queries, EMMA enables efficient query optimization. EMMA defines five general classes of query operators. The extraction operators provide means to access all the attributes of the entities within the EMMO model, the navigational operators enable the navigation along an EMMO's semantic graph structure, the constructors allow for the modification, combination, and creation of new EMMOs, the selection predicates facilitate the selection of only those entities satisfying a specific characteristic, and the join operator relates several entities or EMMOs with a join condition.

Both, the EMMO model and the EMMA algebra, provide a sound basis for the integration of ontological knowledge. We demonstrate how ontological knowledge can be used within the authoring process of EMMOs, can be integrated within the EMMO knowledge structures, and can be exploited for refining EMMA queries.

After establishing the theoretical foundation, we explain the realization of the EMMO container infrastructure supporting platform independency and scalability, providing export and import facilities, and tools for displaying and authoring EMMOs. Moreover, we describe the design and implementation of the EMMA query processing architecture and discuss evaluation results.

Finally, we illustrate by means of three application scenarios how our approach may be practically applied and what benefits it offers. The first scenario is the development of a platform for the exchange of cultural knowledge, the second scenario addresses the domain of eLearning, and the third scenario introduces a multimedia task management system. All three application scenarios have in common that they require an infrastructure providing means for the distributed and collaborative authoring of multimedia content. The first scenario has already been successfully deployed, the realization of the other two scenarios is the subject of future research.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Dr. Werner Winiwarter whose help, stimulating suggestions and encouragement helped me during all the time of writing this thesis. I am also grateful to Prof. Dr. Christian Breiteneder for being my second advisor.

I warmly thank all my colleagues who supported me in my research work. Dr. Utz Westermann, thank you for your help, support, interest, and valuable hints with regard to my work, Karin Schellner, thank you for the inspiring collaboration within the CULTOS project, and Bernhard Haslhofer, thank you for your work on the implementation of the EMMA algebra.

And to my parents, thank you very much for your continuous and loving support over all these years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Although more and more multimedia content becomes available, an efficient management enabled by the *sharing and collaborative authoring of multimedia content* remains still an open issue.

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

In order to facilitate a semantic modeling of multimedia content in content sharing and collaborative applications, we have developed *Enhanced Multimedia Meta Objects (EMMOs)*. EMMOs establish tradeable knowledge-enriched units of multimedia content that indivisibly combine three of the content's aspects into a single object:

- *The media aspect:* an EMMO encapsulates the basic media objects of which the multimedia content consists.
- *The semantic aspect:* an EMMO further encapsulates semantic associations between its media objects.
- *The functional aspect:* an EMMO may define arbitrary, domain-specific operations on the content that can be invoked by applications.

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

For the realization of advanced operations on EMMOs, efficient retrieval and processing of the information captured by EMMOs is required. Therefore, we have developed the *query algebra EMMA*, which provides a formal basis for querying the complete information captured by EMMOs. By featuring simple and orthogonal operators, which can be combined to formulate more complex queries, EMMA enables efficient query optimization. EMMA defines five general classes of query operators. The *extraction operators* provide means to access all the attributes of the entities within the EMMO model, the *navigational operators* enable the navigation along an EMMO's semantic graph structure, the *constructors* allow for the modification, combination, and creation of new EMMOs, the *selection predicates* facilitate the selection of only those entities satisfying a specific characteristic, and the *join operator* relates several entities or EMMOs with a join condition.

Both, the EMMO model and the EMMA algebra, provide a sound basis for the *integration of ontological knowledge.* Within each application scenario, there exists a shared and common understanding of the domain that can be used for the efficient management of EMMOs, such as authoring of, comfortable access to, or searching for EMMOs. The integration of ontology knowledge into the EMMO model and the EMMA algebra has three appealing benefits:

- Ontological knowledge can be used for checking integrity constraints during the design and *authoring process of EMMOs.*
- Ontological knowledge can be incorporated within the *EMMO model* by extending the knowledge structure of EMMOs.
- Knowledge inherent in a domain ontology can be seamlessly integrated into *EMMA queries* allowing to refine imprecise user queries by drawing inferences over the ontological knowledge.

In this thesis we describe three *application scenarios* that illustrate how our approach may be practically applied and what benefits it offers. The first scenario is the development of a platform for the exchange of cultural knowledge, the second scenario addresses the domain of eLearning, and the third scenario introduces a multimedia task management system. All three application scenarios have in common that they require an infrastructure providing means for the distributed and collaborative authoring of multimedia content. The first scenario has already been successfully deployed, the realization of the other two scenarios is the subject of future research.

To summarize, the main contributions of this thesis are:

- We propose and formally define *Enhanced Multimedia Meta Objects (EMMOs)* as tradeable, knowledge-enriched units of multimedia content. The formal basis of the EMMO model are *entities*, which occur in four different specializations, i.e. *logical media parts* modeling media objects, *ontology objects* representing concepts of an ontology, *associations* describing binary relationships between entities, and *EMMOs* aggregating semantically related entities. The formalization of the EMMO model has also been published in [SWZK03].
- Based on the formal definition of the EMMO model, we propose and formally define the *query algebra EMMA*, which is adequate and complete with regard to the EMMO model. EMMA offers a rich set of orthogonal query operators, sufficiently expressive to provide access to all aspects of EMMOs and to enable efficient query rewriting and optimization. The query algebra EMMA has also been published in [ZWW04b] and [ZWW04a].

- Both, the EMMO model and the EMMA algebra, allow for the seamless *integration of ontological knowledge.* We illustrate how ontological knowledge can be used within the authoring process of EMMOs, can be integrated within the EMMO knowledge structures, and can be utilized for refining EMMA queries. The integration of ontological knowledge has also been published in [ZW04b], [ZW04a], and [ZW05].

- We outline three *application scenarios* – a platform for the exchange of cultural knowledge, a distributed and collaborative eLearning platform, and a multimedia task management system – to demonstrate the benefits of using the EMMO infrastructure as platform for collaborative and distributed authoring and sharing of multimedia content. The first application scenario is also described in [SWZK03] and [WZSK05].

The remainder of this thesis is structured as follows. In Chapter 2, we formulate the requirements for sharing and distributed authoring of multimedia content. Based on these requirements, we discuss related standards and approaches. In particular we analyze standards for semantic modeling and object-oriented approaches to what extent they can fulfil the requirements.

In Chapter 3, we first introduce the three aspects of an EMMO, before we formalize the EMMO model based on entities occurring in four different kinds, i.e. logical media parts, ontology objects, associations, and EMMOs.

In Chapter 4, we specify the most important requirements for a query algebra for EMMOs, i.e. the access to all information captured by EMMOs, joins, subgraph matching, and orthogonal operators with precise semantics as a basis for query optimization. Related approaches and standards are analyzed with regard to the requirements.

In Chapter 5, we present the formal definitions of the five classes of EMMA operators, i.e. extraction operators, navigational operators, constructors, selection predicates, and the join operator, along with illustrative examples, and show how these operators satisfactorily address all our requirements.

In Chapter 6, we first point out the benefits of integrating ontology knowledge within the EMMO model, identify the requirements for an ontology description language for EMMOs, i.e. maximum expressiveness while still enabling reasoning support, and support for the exchange and sharing of ontologies, before we analyze to what extent related ontology description languages fulfil these requirements.

In Chapter 7, we demonstrate how ontological knowledge can be used for checking integrity constraints within the design and authoring process of EMMOs, renders it possible to extend the knowledge structures described by EMMOs, and allows for the refinement of EMMA query expressions. To conclude the chapter, we discuss different ways of representing ontology structures and introduce an EMMO representation to enable the seamless integration into the EMMO model.

After establishing the theoretical foundation, Chapter 8 deals with the implementation and evaluation of the EMMO model. We first describe the realization of the platform independent and scalable EMMO container infrastructure along with its export and import facilities, and its tools for displaying and authoring EMMOs, before we introduce the design and implementation of the EMMA query processing architecture and report on some first evaluation results.

Chapter 9 demonstrates the practical benefits of the EMMO model by outlining three application scenarios, i.e. a platform for the exchange of cultural knowledge, a distributed and collaborative eLearning platform, and a multimedia task management system.

Chapter 10 concludes this thesis. It provides a summary of the key contributions of our research and gives an outlook on future work.

Chapter 2

Towards Multimedia Meta Modeling

Although more and more multimedia content becomes available, an efficient management enabled by the *sharing and collaborative authoring of multimedia content* remains still an open issue.

Current multimedia content formats, such as HTML [RHJ99], SMIL [A$^+$01], or SVG [FJJ03], focus on the encoding of the presentation of content but neglect the information the content conveys. By limiting the modeling to the presentation-oriented aspects of multimedia content, only hard-wired presentations of multimedia content specified in exactly one way and for one single purpose can be realized. As valuable information about the semantics of content is lacking, advanced operations, such as retrieval and reuse, automatic composition, recommendation, and adaptation of content according to user interests, information needs, and technical infrastructure cannot be realized. Except static rendering functionality, multimedia content formats provide no further functionality on the encoded multimedia content.

Current semantic approaches to multimedia content modeling, such as RDF [Bec04] or Topic Maps [ISO00a], establish a starting point for the integration of meta knowledge within the management of multimedia content. However, those approaches handle the content's media objects, the semantic description of the content, and the functionality, such as rendering, to be applied to the content's semantic and media description as separate entities. The media, semantic, and functional information is usually stored at separate servers in separate files or databases. Typically, the latter are even under the control of different authorities. Because of the separation of the media content from its semantic description and functionality, the exchange, sharing, and collaborative authoring of multimedia content is not possible.

In the following sections, we will first discuss the requirements for the sharing and collaborative authoring of multimedia meta content, and then analyze to what extent related standards and approaches satisfy the introduced requirements.

2.1 Requirements

For enabling the sharing and collaborative authoring of multimedia content, the content needs to be exchanged and traded between different applications. Thus, one important prerequisite for the realization of a collaborative multimedia platform is to establish a *container unit* encompassing all aspects of the multimedia content.

The container unit should address the multimedia content's *media aspect*, i.e. it should aggregate the described media objects. As multimedia content is very storage intensive, the trading and exchange of multimedia content can cause high CPU usage or result in problems regarding storage capacity. Therefore, the containment of media data should be enabled in two different ways, i.e. either by directly embedding the media data within the container unit or by referencing the media object without storing the data. This can be realized by describing the physical media data on a logical level.

For enabling the sophisticated reuse of data, the container unit should address the multimedia content's *semantic aspect*, i.e. the content has to be enriched with semantic information. Thus, the container unit should establish a *meta object* that contains references to knowledge and multimedia content:

- It should be possible to relate the media objects to each other, comparable to a semantic network. Therefore, the meta model should be *graph-based* with links established by binary directed relationships. In the context of collaborative authoring of multimedia content, the expressiveness of binary directed relationships seems perfectly sufficient; the modeling of n-ary relationships would only increase the complexity of the modeling approach without significantly enhancing its expressiveness for the application domain.

- To enable the reuse and adaptation of multimedia content, it should be possible to combine multimedia content units and relate them to each other. For instance, by capturing multimedia enhanced learning material in modular learning units, and by combining and nesting of modular units into more complex units, sophisticated, context-aware, and personalized lectures can be created dynamically. Thus, the *encapsulation* of container units, i.e. the containment of container units within other container units, should be supported.

- By providing a platform for the collaborative authoring of multimedia content, there's a real possibility of encountering statements, i.e. single relationships or even graph structures that are in conflict with each other. Therefore, a mechanism for recording the source of information or for commenting the information is required. The approach should render it possible to make statements about other statements, i.e. to *reify* statements.

The final aim is to work with the content represented by the container unit in an efficient and flexible way. This can be realized by offering operations, such as a rendering or payment function, which can be invoked by external applications. Thus, the container unit should address the *functional aspect* of multimedia content allowing the container unit to know how to process its content, e.g. how to render it or how much to charge for it.

Within a collaborative scenario, the multimedia content evolves and changes over time. Several users are working on the same objects stored in multiple databases. For the purpose of reference, the chronological steps in the development of a container unit and its components are an important information to know. This information can be captured by the creation and administration of *different versions* of the container units and their components.

2.2 Related Approaches

The fundamental requirement for enabling the collaborative and distributed authoring of multimedia content is the creation of a container unit unifying three different aspects of multimedia content, namely the media aspect, the semantic aspect, and the functional aspect, as well as the versioning information. In the following, we analyze related approaches to what extent they can provide a basis for the modeling of multimedia content by establishing a container unit capturing the content's three aspects and versioning information.

Interrelating basic media objects like single images and videos to form *multi*media content is the task of *multimedia document models*. Recently, several standards for multimedia document models have emerged [BKW00], such as HTML [RHJ99], XHTML+SMIL [NPS02], HyTime [ISO97], MHEG-5 [ISO96], MPEG-4 BIFS and XMT [PE02], SMIL [A+01], and SVG [FJJ03]. Multimedia document models can be regarded as composite media formats that model the presentation of multimedia content by arranging basic media objects according to temporal, spatial, and interaction relationships. Therefore, they mainly address the media aspect of multimedia content. However, multimedia document models neither interrelate multimedia content according to semantic aspects nor do they allow to provide functionality on the content. They rely on external applications like presentation engines for reasonable content processing.

Middleware approaches like Enterprise Java Beans (EJBs) [MH98], CORBA [OMG02], DCOM [Mic98], or Web Services [C+02] are also not really compatible with the idea of a container unit unifying all three aspects of multimedia content and versioning information. Although they all feature notions of objects offering functionality that can be remotely invoked by applications, they constitute general purpose infrastructures with no special support for the representation of multimedia content. Even specialized multimedia middleware like PREMO [JH98] uses the notion of objects primarily to represent components of a distributed multimedia infrastructure, such as streaming servers, instead of employing the notion of objects to represent multimedia content.

2.2.1 Standards for Semantic Modeling

In parallel to research concerning the Semantic Web, a variety of standards have emerged that can be used to model multimedia content by describing the information it conveys on a semantic level. The most prominent examples are MPEG-7 (especially MPEG-7's graph tools for the description of content semantics [ISO01a]), RDF [Bec04, BG04], Topic Maps [ISO00a], and Conceptual Graphs [ISO01b]. As those semantic standards offer means to address media objects within a description, they undoubtedly refer to the media aspect of multimedia content. In addition, those semantic standards clearly cover some parts of the semantic aspect of multimedia content. The discussion of the requirements for multimedia meta modeling in the previous section showed that the semantic aspect covers many different characteristics. In the following, we will introduce the above-mentioned semantic standards in more detail and discuss to what extent they establish a graph-based meta model, enable the reification of statements, and allow the encapsulation of container units.

MPEG-7

MPEG-7 [ISO01a] is an ISO/IEC standard for describing features of multimedia content, which was developed by the Moving Picture Expert Group (MPEG). It provides a flexible and extensible framework for describing audio-visual content in such a way that users can browse, search, and retrieve content more efficiently than by only using text-based search engines. It standardizes a set of *Descriptors*, a set of *Description Schemes*, and the *Description Definition Language*. A Descriptor (D) is a representation of a feature that defines the syntax and semantics of the feature representation. A Description Scheme (DS) specifies the structure and semantics of relationships between components. Those components may be both Descriptors and Description Schemes. The Description Definition Language (DDL) is a language that specifies Description Schemes and possibly Descriptors, and allows for the extension and modification of existing Description Schemes. MPEG-7 uses an extension of the XML Schema Language as the MPEG-7 DDL.

The scope of MPEG-7 is clearly stated (see Fig. 2.1). It addresses the feature description of multimedia content. However, the automatic feature extraction is outside the scope of the specification, and also programs, which can make use of the descriptions, such as search engines or filter agents, are not specified in the MPEG-7 standard.

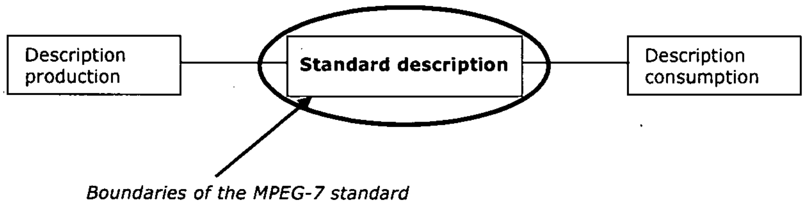


Figure 2.1: Scope of MPEG-7

The MPEG-7 standard consists of seven parts: System, Description Definition Language, Audio, Visual, Multimedia Description Schemes, Reference Software, and Conformance. The Multimedia Description Schemes (MDS) of the MPEG-7 standard provides a Description Scheme, called the `Graph DS`, for representing graphs. A graph consists of nodes and relations. The relations are defined in the Description Scheme `Relation DS` as directed relations between two or more nodes. Thus, MPEG-7 provides a tool for describing n-ary and directed graph structures. As the nodes of the `Graph DS` reference description scheme instantiation, they possibly can reference other graph structures as well, enabling the encapsulation of graph structures within one document. However, as a graph is defined as extension of the Description Scheme `DSType`, it is referenced by an attribute of type ID, which is only valuable within the same document. Therefore, only graphs contained within the same document can be referenced. This limits the description of encapsulated graph structures to the boundaries of one document. Therefore, MPEG-7 does not sufficiently support the required encapsulation functionality.

A relation described within the Description Scheme `Graph DS` refers to a source and target node, which again can reference other relations. Thus, MPEG-7 enables the reification of relations. However, as the `Relation DS` is again an extension of type `DSType`, also the reification of relations is restricted to the boundaries of the underlying document.

RDF

Resource Description Framework (RDF) [Bec04] is a language for the modeling of meta data about Web resources. It provides a way for expressing simple statements about resources by using named properties and values. RDF Schema (RDFS) [BG04] adds a type system to RDF, i.e. it does not specify a vocabulary of application-specific classes, but establishes a mechanism for describing classes and properties.

The RDF model describes *resources*, *properties*, and *statements*. Resources are all things being described, e.g. a Web page, or a part of a Web page, an object stored in a database, or an object which is not directly accessible via the Web, e.g. a natural person or a printed book. Resources are identified by a resource identifier, i.e. a Uniform Resource Identifier (URI). Properties are resources that are used for describing other resources. Property names are associated with an RDF Schema, which specifies the permitted values of a property and how a property relates to other properties. A statement is the base element of an RDF model. Basically, it is a triple with three elements *subject*, *predicate*, and *object*. The subject constitutes a resource, the predicate a named property, and the object, i.e. the value of the predicate, a resource or a literal. Thus, statements establish semantic relationships between two resources, or between a resource and a literal.

The basic RDF model is described as set of statements, i.e. it consists of a set of triples. A set of triples establishes a directed graph with the nodes and links representing resources that are labeled by URIs. Thus, RDF models describe directed, binary graph structures.

The boundaries of an RDF model are specified in an RDF document. An RDF document is a file which is labeled by an RDF header and contains a collection of RDF triples. By not defining a mechanism for the classification of RDF documents as resources, RDF provides no way for defining the boundaries of RDF models on a virtual level. Therefore, RDF models cannot be specified as subject resource within other statements, i.e. the encapsulation of an RDF model within another RDF model cannot be realized by only referencing without copying it. Thus, RDF does not provide sufficient means for realizing the encapsulation functionality.

RDF provides a built-in vocabulary that can be used for creating a description of a statement, i.e. a resource of type `rdf:Statement` can be used for building a model of the original statement. This model represents the reified statement and has to be distinguished from the original statement, i.e. the model of a statement is just a representation of the statement, but not the statement itself.

Besides the fact that the built-in approach for making statements about statements is very tedious and cumbersome – the reification of a statements requires the creation of four additional triples describing the resources necessary to make the intended statement – problems of data consistency arise. As there is no direct correlation between the original statement and its model, changes within the original statement are not automatically reflected within the reification of the model. By not establishing a first-class object representing a statement, RDF – although establishing a way of making statements about statements – does not provide the reification functionality required for the collaborative multimedia meta modeling.

Topic Maps

The specification XML Topic Maps (XTM) 1.0 [PM01] was written by the TopicMaps.Org, which is an independent consortium of parties interested in making the Topic Maps Paradigm applicable in the World Wide Web. The Topic Maps Paradigm is fully described in the ISO/IEC 13250:2000 "Topic Maps" standard [ISO00a], which is based on SGML [ISO86] and HyTime [ISO97]. The specification provides an abstract model and XML grammar for interchanging Web-based topic maps. By enabling the representation of the structure of information resources, it facilitates the organization and navigation of large collections of information objects.

The key concepts in Topic Maps are *topics*, *occurrences*, and *associations*. A topic is a resource within the computer that stands for, or reifies, some real-world subject, i.e. it is a resource that acts as a proxy. The relationship between a topic and its subject is defined to be one of `reification`[1]. The act of creating a topic is called `reification`. When something is reified it becomes the subject of the topic, created in the `reification` process.

Topics can have occurrences. An occurrence is any information that is specified to be relevant to a given subject. Occurrences must be resources that are either addressable via Uniform Resource Identifiers (URI), or can be placed in-line as character data. Moreover, topics can participate in relationships – called associations – in which they play roles as members. The number of members of an association is unlimited, and associations specify no directionality, i.e. their directionality is determined by their type and by the roles their members play.

Topic Map documents contain one or more *topic maps*, and topic maps are collections of topics, occurrences, and associations. By associations describing relationships between topics without limiting the number of their members or specifying any directionality, Topic Map documents establish n-ary graph structures.

A topic map element, i.e. the XML element specifying the root of a Topic Map document or the root of a sub tree of a Topic Map document, constitutes a resource, but no topic. Since in XML Topic Maps (XTM) 1.0 anything can be subject of `reification`, topic map elements can be reified: A new topic establishing a surrogate of the topic map element is created and can now be used as topic within other topic maps. By topic maps encapsulating other

[1]Please note, that the semantics of the term reification in XTM differs from the semantics of the term reification used in the context of the requirements described in Sect. 2.1. In order to distinguish this in the text, we typeset the term reification in "teletype" when used with the semantics of XTM.

topic maps just by references and without copying their data, XML Topic Maps (XTM) 1.0. provides the required encapsulation functionality.

Although associations express relationships between topics, they are no topics and, thus, cannot participate themselves in associations, i.e. it is not possible to express relationships between associations. However, again, by `reifying` the association, one can create a topic acting as a proxy of the association. This newly created topic can participate in another association, i.e. it can be used as member of another association. Thus, by using the `reification` process, XML Topic Maps (XTM) 1.0. enables the reification of statements as specified in Sect. 2.1.

Conceptual Graphs

The Conceptual Graphs specification [ISO01b] is an ISO draft version which specifies the syntax and semantics of conceptual graphs. It aims to express meaning in a form that is logically precise, human readable, and machine processable. A conceptual graph is an abstract representation for first-order logic with nodes called concepts and conceptual relations. A concept refers to an entity, a set of entities, or a range of entities. A conceptual relation has zero or more arcs, each of which links the conceptual relation to some concept. Conceptual Graphs realize the full expressive power of first-order logic, i.e. concepts can be quantified and statements can be negated.

As the name suggests, a conceptual graph can be represented as graph structure, i.e. the annex of the Conceptual Graphs specification describes a graphical display format (DF). A conceptual graph is a bipartite graph, which consists of two kinds of nodes. These are the concepts and the conceptual relations. The arcs of the graph link some conceptual relations to some concepts. By determining the roles of the adjacent concepts, the direction and semantics of the arcs are defined. As one conceptual relation can be connected to several concepts, Conceptual Graphs enable the description of n-ary relationships, i.e. they allow the definition of n-ary graph structures.

Although conceptual graphs can be nested within other conceptual graphs, this can be only realized by copying the data, and not by remote references. Therefore, Conceptual Graphs do not satisfy the required encapsulation functionality.

In Conceptual Graphs, statements are represented by concepts being connected to one or more conceptual relations. However, as statements themselves constitute no concepts, they cannot be subject of another statement. Therefore, Conceptual Graphs provide no mechanism for the reification of statements.

As already mentioned, Conceptual Graphs cover the full expressive power of first-order logic. For the expression of the encapsulation and reification functionality, modeling constructs of second-order logic, i.e. the quantification of relations, are required.

2.2.2 Object-Oriented Approaches

There exist several approaches that represent multimedia content by means of object-oriented technology. All those approaches have in common that they are specialized for a specific application scenario and only address one, sometimes two, of the multimedia content's aspects.

Enterprise Media Beans

Enterprise Media Beans (EMBs) [Bau02] extend the Enterprise Java Beans (EJBs) architecture [MH98] with predefined entity beans for the representation of basic media objects within enterprise applications. These come with rudimental access functionality but can be extended with arbitrary functionality using the inheritance mechanisms available to all EJBs, and allow the specification of predecessor and successor relationships. Though providing versioning support and addressing the media and functional aspects of content, EMBs are mainly concerned with single media content and not with *multi*media content. Furthermore, EMBs do not offer any dedicated support for the semantic aspect of content.

Adlets

Adlets [CZ01] are objects that represent individual (not necessarily multimedia) documents. Adlets support a fixed set of predefined functionality, which enables them to advertise themselves to other Adlets. They are thus content representations that address the media as well as the functional aspect. However, the functionality supported by Adlets is limited to advertisement and there is no explicit modeling of the semantic aspect. Although Adlets are intended to continuously change over time and the Adlet infrastructure implements a dynamical update algorithm, a dedicated versioning support is not provided.

Tele-Action Objects

Tele-Action Objects (TAOs) [C$^+$95] are object representations of multimedia content that encapsulate the basic media objects of which the content consists and interlink these objects with associations. However, TAOs therefore mainly address the media aspect of multimedia content, they do not adequately cover the semantic aspect of multimedia content: only a fixed set of 5 association types is supported, mainly concerned with temporal and spatial relationships for presentation purposes. TAOs can further be augmented with functionality. Such functionality is automatically invoked as the result of system events and cannot be explicitly invoked by applications. TAOs provide no versioning support.

Distributed Active Relationships

Distributed Active Relationships [DLP98] define an object model based on the Warwick Framework [LLD96]. In this model, Digital Objects (DOs), which are interlinked with each other by semantic relationships, act as containers of metadata describing multimedia content. DOs do not address the media aspect of multimedia content but focus on the semantic aspect. The links between containers can be supplemented with arbitrary functionality. However, the functionality is not explicitly invoked by applications but implicitly invoked whenever an application traverses a link between two DOs. Although the semantic relationships provide a basis for describing predecessor and successor relationships between DOs, a dedicated versioning support is not implemented.

2.3 Summary

For enabling the sharing and collaborative authoring of multimedia content, a versionable container unit unifying the media, semantic, and functional aspect of multimedia content, has to be established. For providing the semantic aspect of multimedia content, the container unit needs to describe a graph-based meta model enabling the encapsulation of container units and the reification of statements.

The four semantic standards we analyzed clearly cover parts of the semantic aspect of multimedia content. Table 2.1 summarizes to what extent the standards MPEG-7, RDF, Topic Maps, and Conceptual Graphs support the semantic aspect of multimedia content.

Table 2.1: Fulfilment of requirements by standards for semantic modeling

Requirements + := support, − := no support	MPEG-7	RDF	Topic Maps	Conceptual Graphs
Directed binary graphs	+	+	+	+
Encapsulation	−	−	+	−
Reification	−	−	+	−

All four standards provide means for describing binary, directed graph structures, but only Topic Maps satisfy the required encapsulation and reification functionality. Thus, Topic Maps cover all necessary parts of the semantic aspect. However, all four approaches provide no versioning support and offer no functionality on multimedia content. They rely on external software like database or knowledge base technology, search engines, user agents, etc. for the processing of content descriptions. Furthermore, media descriptions and the media objects described are separate entities – potentially scattered around different places on the Internet, created and maintained by different authorities, who might not be synchronized or not even aware of each other. The existence of a container unit unifying all the three aspects and versioning information of multimedia content is missing.

Table 2.2 summarizes to what extent the discussed object-oriented approaches satisfy the three aspects of multimedia content and provide versioning support. The four object-oriented approaches provide no or only limited support for the semantic aspect of multimedia content. Although establishing versionable objects carrying functionality, Enterprise Media Beans – by only addressing single media objects – provide only limited support for the media aspect of multimedia content. Adlets and TAOs address the multimedia content's media aspect and implement some application-dependent functionality, but maintain no versioning support. Distributed Active Relationships establish objects that are interlinked by semantic relationships carrying functionality, but do neither address the media aspect nor establish a versioning mechanism.

Table 2.2: Fulfilment of required features by object-oriented approaches

Requirements + := support, (+) := limited support, – := no support	Enterprise Media Beans	Adlets	Tele-Action Objects	Distributed Active Relationships
Media aspect	(+)	+	+	–
Semantic aspect	–	–	(+)	(+)
Functional aspect	+	(+)	(+)	(+)
Versioning support	+	–	–	–

Chapter 3

EMMOs

EMMOs (Enhanced Multimedia Meta Objects) provide the basis for the distributed, collaborative authoring of multimedia content. In the following sections, we will first describe the aspects of an EMMO, and subsequently introduce the EMMO model along with illustrative examples.

3.1 The Aspects of an EMMO

An EMMO is a self-contained unit of multimedia content that encompasses three aspects, which we illustrate using EMMO "Dracula Movies" in Fig. 3.1.

1. *The media aspect:* An EMMO aggregates the media objects of which the multimedia content consists. In the example, we see that the depicted EMMO contains two MPEG videos "Caligari.mpeg" and "Nosferatu.mpeg", and two AVI videos "Salem183.avi" and "Salem112.avi". 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 a 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, it is stated that the media objects contained within the EMMO are digital manifestations of Wiene's movie "The Cabinet of Dr. Caligari", Murnau's movie "Nosferatu", and Hooper's movie "Salem's Lot". The model used for semantic associations is expressive, e.g. it is 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 can be invoked by applications. In the example, the depicted EMMO is associated with two operations: one for rendering the EMMO, which returns a presentation of the encapsulated multimedia content in different formats, such as SMIL or SVG. Another operation is provided to handle the payment of the media objects contained in the EMMO before rendering, e.g. by performing a credit card transaction.

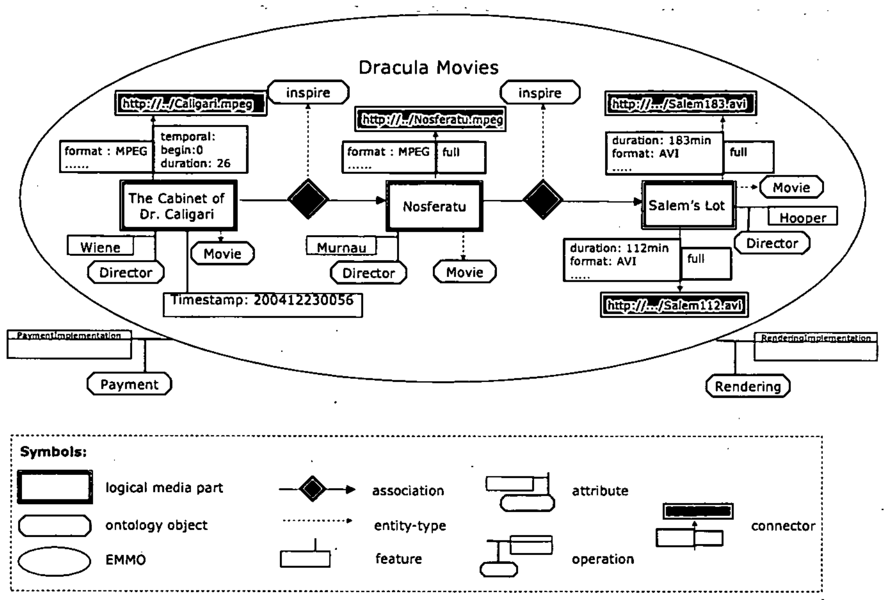


Figure 3.1: EMMO "Dracula Movies" (e_{movies})

In addition, EMMOs have further desirable characteristics: They provide *versioning* support, i.e. all the constituents of an EMMO can be versioned, and EMMOs can be *serialized* into a bundle that completely encompasses all three aspects and the versioning information. Therefore, an EMMO is *transferable* in its entirety between different EMMO providers, including its contained media objects, semantic associations between these objects, functionality, and versions. Thus, EMMOs pave the way for the *distributed, collaborative construction* of knowledge enriched multimedia content.

3.2 The EMMO Model

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

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

In the following subsections, we formally define entities and their four specializations, and use EMMO "Dracula Movies" in Fig. 3.1 as a running example.

3.2.1 Entities

Each entity w is characterized by thirteen different properties:

- Each entity w has a *global* and *unique object identifier (OID)* o_w represented as universal unique identifier (UUID) [Lea98], which enables the unique identification of entities in distributed scenarios.
- As UUIDs are not really useful for humans, each entity w has also a human readable *name* n_w expressed as string value.
- For classifying whether an entity w is a logical media part, an ontology object, an association, or an EMMO, its *kind* k_w is specified accordingly.
- Each entity w is described by a set of *types* T_w, i.e. a set of ontology objects, enabling the classification by concepts taken from a domain ontology, e.g. the entity "Salem's Lot" is classified as "Movie" (see Fig. 3.1).
- Each entity possesses an arbitrary number of application-dependent *attributes* A_w. Attributes are represented as attribute-value pairs with the attribute name being a concept of a domain ontology. For example, by attaching the value "Murnau" for the attribute "Director" to the entity representing the movie "Nosferatu" (see Fig. 3.1), one can express that the movie was directed by Murnau. The attribute value is per default untyped, however, typing constraints can be introduced via the domain ontology (see Sect. 7.1).
- For providing versioning support, a set of *preceding versions* P_w and *succeeding versions* S_w can be assigned to each entity w. Each version of w is again an entity of the same kind k_w. By also treating an entity's versions as entities, different versions of an entity can be interrelated just like any other entities, thus allowing to establish relationships between entity versions. Figure 3.2 shows several versions of the EMMO "Dracula Movies" and their interrelationships.
- As it might be necessary in an implementation of the model to augment an entity w with further low-level data, such as time stamps or status information, in a flexible, ad-hoc manner, a set of *features* F_w, represented as feature-value pairs, can be attached to the entity. In contrast to attributes, feature names are not ontology objects but simple strings.

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

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

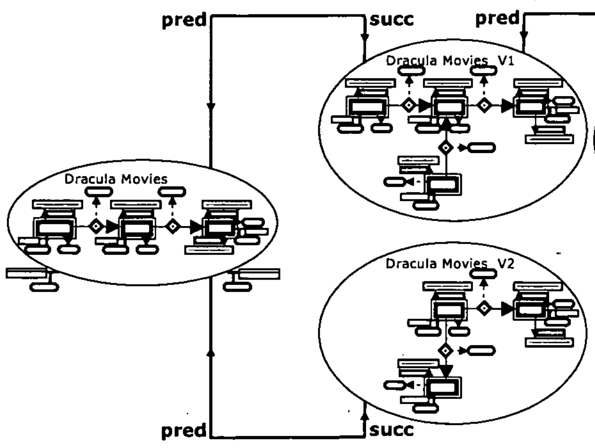


Figure 3.2: EMMO "Dracula Movies"'s versions

- An *EMMO* offers *operations* O_w, which can be invoked by external applications. The implementation of an operation is described by a mathematical *function.*

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

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

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

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

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

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

Constraint (3.1) enforces that each entity has a unique identifer and Constraint (3.2) ensures that each version of w is again an entity of the same kind k_w.

3.2.2 Logical Media Parts

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

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

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

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

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

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

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

In Example 1 we illustrate how the three logical media parts depicted in Fig. 3.1 representing the media objects "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot" can be formally described within the EMMO model. The symbols $l_{caligari}$, $l_{nosferatu}$, and l_{salem} represent the three logical media parts. For example, the thirteen-tuple $l_{caligari}$ indicates that there exists an entity which is uniquely identified by the OID "12471", is named "The Cabinet of Dr. Caligari", is of kind logical media part ("lmp"), specifies no source and target entity, is classified as "Movie", has the value "Wiene" for the attribute "Director", describes its physical media data by the connector (ms_1, mp_1), is augmented by its timestamp information, and specifies its sets of nodes, predecessors, successors, and operations as empty. The connector (ms_1, mp_1) references the temporal selection of the first 26 minutes from the MPEG movie "`Caligari.mpeg`".

(ms_2, mp_2) is the connector of the logical media part $l_{nosferatu}$ associating the complete MPEG movie "Nosferatu.mpeg", and, finally, (ms_3, mp_3) and (ms_4, mp_4) represent two versions of different length of the movie "Salem's Lot".

Example 1

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

3.2.3 Ontology Objects

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

- for designating the types of entities,
- for designating the attributes of attribute values,
- for designating the operations attached to EMMOs (see Def. 9),
- as nodes within the EMMO knowledge structure (see Sect. 3.2.5).

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

As can be seen from Def. 6, the types T_o of an ontology object o can be a non-empty set, i.e. ontology objects can again be classified by other ontology objects. This provides the basis for expressing ontological structures within the EMMO model. The development of a dedicated ontology engineering environment is the focus of future work. The final aim is the seamless integration of ontological knowledge into the EMMO model (see Chapter 7).

In Example 2 all four different ways of using ontology objects are illustrated: Within EMMO "Dracula Studies" (see Fig. 3.3), ontology object $o_{inspire}$ represents the type of the association connecting the two logical media parts

"Vampyre" and "Dracula", ontology object o_{movie} the type of the logical media part "Nosferatu"; ontology object $o_{director}$ is used as name of the attribute attached to the logical media part "Nosferatu", and ontology object o_{render} represents the designator of the operation provided by the EMMO. Furthermore, the ontology object o_{miller}, which represents the concept "Elizabeth Miller", is specified as node contained within EMMO "Dracula Studies" and classified as "Researcher" by additionally typing this ontology object with the ontology object $o_{researcher}$.

Example 2

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

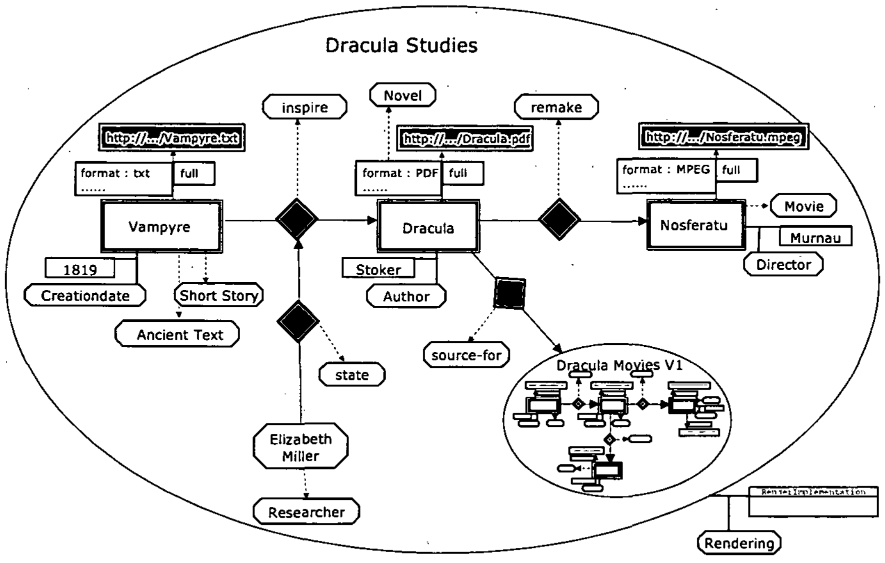


Figure 3.3: EMMO "Dracula Studies" ($e_{studies}$)

3.2.4 Associations

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

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

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

Example 3 shows the formal definition of the two associations $a_{ca \to no}$ and $a_{no \to sa}$ contained within EMMO "Dracula Movies"(Fig. 3.1) and of the four associations $a_{va \to dr}$, $a_{dr \to no}$, $a_{mi \to (va \to dr)}$, and $a_{dr \to mo}$ contained within EMMO "Dracula Studies"(Fig. 3.3). Association $a_{va \to dr}$ models the fact that the text "Vampyre" inspired the novel "Dracula", and by expressing that this statement was made by "Elizabeth Miller", association $a_{mi \to (va \to dr)}$ exemplifies the reification of associations.

Example 3

$$\begin{aligned}
a_{ca \to no} &= (\text{“}a0225\text{”}, \text{“}ca \to no\text{”}, \text{“asso”}, l_{caligari}, l_{nosferatu}, \{o_{inspire}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{no \to sa} &= (\text{“}a5461\text{”}, \text{“}no \to sa\text{”}, \text{“asso”}, l_{nosferatu}, l_{salem}, \{o_{inspire}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{va \to dr} &= (\text{“}a6390\text{”}, \text{“}va \to dr\text{”}, \text{“asso”}, l_{vampyre}, l_{dracula}, \{o_{inspire}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{dr \to no} &= (\text{“}a5461\text{”}, \text{“}dr \to no\text{”}, \text{“asso”}, l_{dracula}, l_{nosferatu}, \{o_{remake}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{mi \to (va \to dr)} &= (\text{“}a4771\text{”}, \text{“}mi \to (va \to dr)\text{”}, \text{“asso”}, o_{miller}, a_{va \to dr}, \{o_{state}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset),\\
a_{dr \to moV1} &= (\text{“}a1289\text{”}, \text{“}dr \to moV1\text{”}, \text{“asso”}, l_{dracula}, e_{moviesV1}, \{o_{source\text{-}for}\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}$$

3.2.5 EMMOs

An EMMO constitutes the core component of our model. It is a container that combines several entities into a single unit. By aggregating media data (i.e. logical media parts) and enriching this media data by semantic data (i.e. associations and ontology objects), an EMMO addresses the *media* and *semantic aspect* of multimedia content modeling. For instance, EMMO "Dracula Movies" groups the semantic descriptions of the logical media parts "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot" into one single unit. Since EMMOs are modeled as entities, EMMOs can be contained within other EMMOs, just as any other entity. Therefore, a structure of hierarchically nested EMMOs can be established: EMMO "Dracula Studies" in Fig. 3.3, for example, contains EMMO "Dracula Movies V1", a direct successor version of EMMO "Dracula Movies". Furthermore, an EMMO can also take part in associations, facilitating the representation of knowledge about the EMMO. For instance, within EMMO "Dracula Studies" it is stated that the novel "Dracula" was the source for EMMO "Dracula Movies V1". Finally, by specifying operations that process its content, EMMOs address the *functional aspect* of multimedia content.

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

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

According to this definition, an EMMO e constitutes a container of other entities because its set of nodes N_e is not restricted to an empty set, as it is the case with other kinds of entities. The contained entities form a connected graph structure when they are interlinked by associations. Constraint 3.3 ensures that associations can specify only those entities as source or target entity which already belong to the EMMO's nodes, and thus, guarantees that any established relationship is fully contained within the EMMO.

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

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

In Example 4, the six EMMOs "Dracula Movies", "Dracula Studies", "Dracula Movies V1", "Dracula Movies V2", "Dracula Movies V3", and "Dracula Research" are formally described. EMMO "Dracula Movies"(see Fig. 3.1) consists of five nodes, i.e. the three logical media parts "The Cabinet of Dr. Caligari", "Nosferatu", and "Salem's Lot", and two associations; it defines EMMO "Dracula Movies V1" and EMMO "Dracula Movies V2" as its direct successor versions (see Fig. 3.2), and specifies the functions f_{render} implementing a "rendering" operation and $f_{payment}$ implementing a "payment transaction" operation. EMMO "Dracula Studies"(see Fig. 3.3) aggregates nine entities, i.e. the three logical media parts "Vampyre", "Dracula", and "Nosferatu", the ontology object "Elizabeth Miller", the EMMO "Dracula Movies V1", as well as four associations, and offers a rendering functionality. EMMO "Dracula Movies V1" (see Fig. 3.4), a direct successor version of EMMO "Dracula Movies", contains two additional nodes, i.e. the logical media part representing the novel "Dracula" and a retelling association connecting the logical media parts "Dracula" and "Nosferatu" (named $a_{(dr \rightarrow no)2}$ to distinguish it from the association named $a_{(dr \rightarrow no)}$ in EMMO "Dracula Studies"); it defines again a direct successor version, i.e. EMMO "Dracula Movies V3". EMMO "Dracula Movies V2" (see Fig. 3.5) consists of five entities: the two logical media parts "Nosferatu" and "Salem's Lot" with their connecting association originate from EMMO "Dracula Movies", whereas the logical media part "Van Helsing" and the connecting association are newly added. EMMO "Dracula Movies V3" (see Fig. 3.6) contains all nodes of EMMO "Dracula Movies V1" and two additional nodes, i.e. the logical media part "A Return to Salem's Lot" and an association of type "similar audience" connecting this logical media part with the logical media part "Salem's Lot". Finally, EMMO "Dracula Research" (see Fig. 3.7) consists of eight nodes, i.e. the EMMOs "Dracula Movies","Dracula Movies V3", and "Dracula Studies", the ontology object "Elizabeth Miller", and four associations.

Example 4

$$\begin{aligned}
e_{movies} =& (\text{“e7921”}, \text{“Dracula Movies”}, \text{“emm”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \{l_{caligari}, l_{nosferatu}, l_{salem}, a_{ca\rightarrow no}, a_{no\rightarrow sa}\},\\
& \emptyset, \{e_{moviesV1}, e_{moviesV2}\}, \emptyset, \{(o_{render}, f_{render}), (o_{payment}, f_{payment})\}),\\
e_{studies} =& (\text{“e3811”}, \text{“Dracula Studies”}, \text{“emm”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset,\\
& \{l_{vampyre}, l_{dracula}, l_{nosferatu}, o_{miller}, e_{moviesV1}, a_{va\rightarrow dr}, a_{dr\rightarrow no}, a_{mi\rightarrow(va\rightarrow dr)}, a_{dr\rightarrow moV1}\},\\
& \emptyset, \emptyset, \emptyset, \{(o_{render}, f_{render})\}),\\
e_{moviesV1} =& (\text{“e7390”}, \text{“Dracula Movies V1”}, \text{“emm”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset,\\
& \{l_{caligari}, l_{nosferatu}, l_{salem}, l_{dracula}, a_{ca\rightarrow no}, a_{no\rightarrow sa}, a_{(dr\rightarrow no)2}\}, \{e_{movies}\}, \{e_{moviesV3}\}, \emptyset, \emptyset),\\
e_{moviesV2} =& (\text{“e3491”}, \text{“Dracula Movies V2”}, \text{“emm”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset,\\
& \{l_{nosferatu}, l_{salem}, l_{helsing}, a_{no\rightarrow sa}, a_{no\rightarrow he}\}, \{e_{movies}\}, \emptyset, \emptyset, \emptyset),\\
e_{moviesV3} =& (\text{“e3225”}, \text{“Dracula Movies V3”}, \text{“emm”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset,\\
& \{l_{caligari}, l_{nosferatu}, l_{salem}, l_{dracula}, l_{return}, a_{ca\rightarrow no}, a_{no\rightarrow sa}, a_{(dr\rightarrow no)2}, a_{sa\rightarrow re}\}, \{e_{moviesV1}\}, \emptyset, \emptyset, \emptyset),\\
e_{research} =& (\text{“e1411”}, \text{“Dracula Research”}, \text{“emm”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \{e_{movies}, e_{moviesV3}, e_{studies},\\
& o_{miller}, a_{mo\rightarrow moV3}, a_{moV3\rightarrow st}, a_{mi\rightarrow(mo\rightarrow moV3)}, a_{mi\rightarrow(moV3\rightarrow st)}\}, \emptyset, \emptyset, \emptyset, \emptyset).
\end{aligned}$$

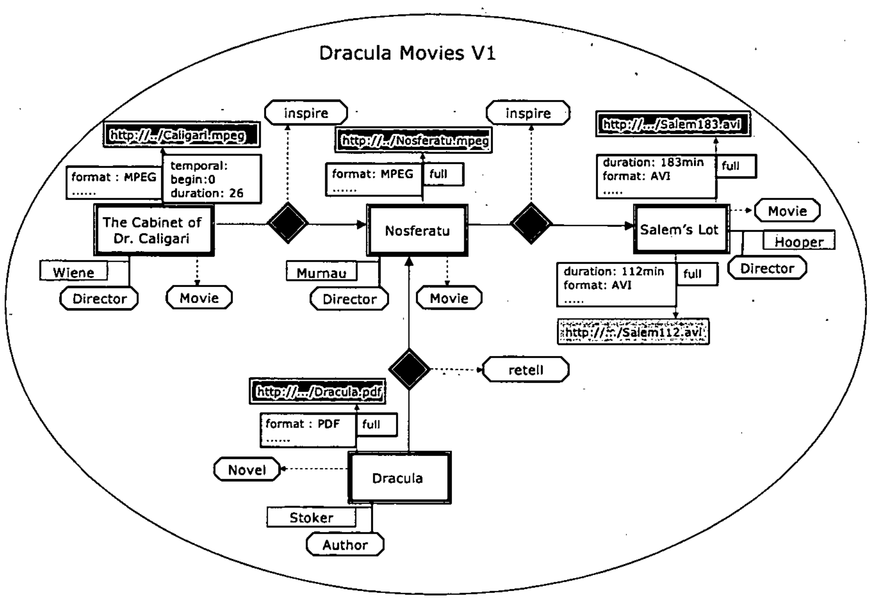


Figure 3.4: EMMO “Dracula Movies V1” ($e_{moviesV1}$)

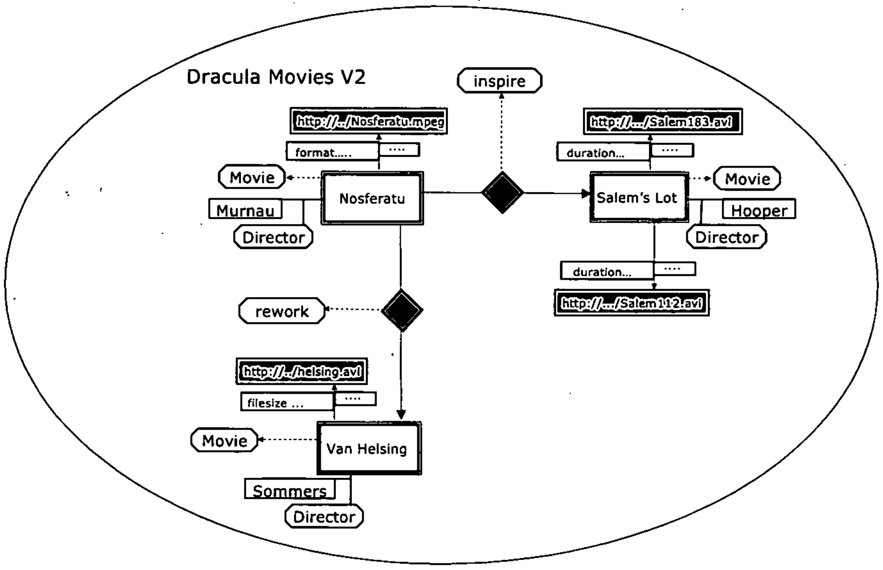


Figure 3.5: EMMO "Dracula Movies V2" ($e_{moviesV2}$)

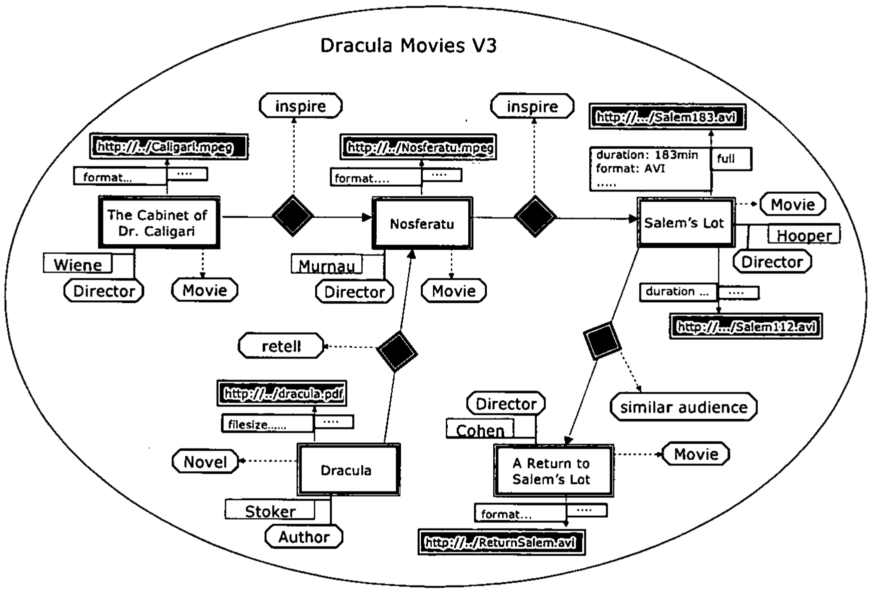


Figure 3.6: EMMO "Dracula Movies V3" ($e_{moviesV3}$)

3.3 Summary

In this chapter, we have introduced the EMMO model, a novel approach for semantic multimedia content modeling for collaborative and content sharing applications.

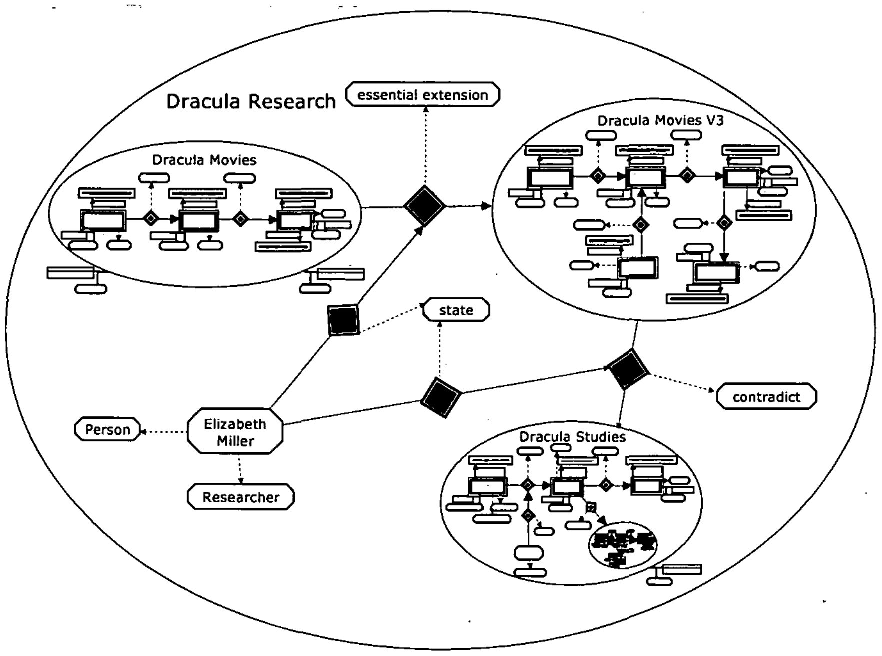


Figure 3.7: EMMO "Dracula Research" ($e_{research}$)

An EMMO constitutes a self-contained piece of multimedia content that indivisibly unites three of the content's aspects. The media aspect reflects that an EMMO aggregates the basic media objects of which the multimedia content consists, the semantic aspect allows the specification of semantic associations between an EMMO's media objects, and, finally, the functional aspect provides means for the definition of arbitrary, domain-specific operations on the content that can be invoked by applications. Furthermore, EMMOs are versionable – they can be modified concurrently in a distributed environment – and tradeable, i.e. all three aspects of the multimedia content and the versioning information can be bundled into one unit and serialized into an exchangeable format.

The formal basis of EMMOs are entities, which are characterized by thirteen properties. Furthermore, entities occur in four different kinds, i.e. logical media parts representing the media data, ontology objects representing concepts of an ontology, associations describing binary relationships, and EMMOs aggregating semantically related entities.

For the retrieval of EMMOs and the knowledge they incorporate, suitable querying facilities are required. The formal definitions of the EMMO model provide the basis for the development of the EMMO query algebra EMMA, which we will introduce in Chapter 5.

Chapter 4

Towards a Query Algebra for EMMOs

EMMOs provide a framework for the distributed and collaborative authoring of semantically enhanced multimedia objects. For the realization of advanced operations on EMMO structures, query facilities enabling the efficient access and processing of the information captured by EMMOs are required. Therefore, we have developed the query algebra EMMA, which provides a formal basis for querying EMMOs, i.e. an algebra providing a set of formal query operators suitable for the EMMO model. In the following, we will first introduce the essential requirements for such a query algebra, before we analyze related approaches.

4.1 Requirements

The EMMO model has no inherent semantics, i.e. the particular semantics of an application scenario implementing the EMMO model is derived from the integrated domain ontology (see Chapters 6 and 7). Therefore, the requirements for accessing the information captured by EMMOs are resulting from structural and syntactical issues, and have to be seen as being independent from the semantics of any particular application scenario. Thus, the design of the EMMO query algebra EMMA was in the first place driven by the requirements for accessing the complete information stored within one or more EMMOs, providing a basis for efficient query rewriting and optimization, and enabling the integration of the knowledge specific to an application domain, i.e. the knowledge from the underlying domain ontology.

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

- an EMMO's *media aspect*, i.e. operators that give access to logical media parts and their connectors;
- an EMMO's *semantic aspect*, i.e. operators that facilitate the retrieval of all kinds of entities contained in an EMMO, the querying of the types of entities and their attribute values, as well as the traversal of associations

between entities; the operators must be expressive enough to cope with the more advanced constructs of the EMMO model, such as the reification of associations and the nesting of EMMOs;

- an EMMO's *functional aspect*, i.e. operators that allow the access to and permit the execution of the operations of an EMMO;
- an EMMO's *versioning information*, i.e. operators for the querying of an entity's direct and indirect versions.

To combine information contained within different EMMOs, the algebra has to support *joins* between entities. Moreover, a suitable algebra should support some basic construction and manipulation operators, such as union, intersection, and difference. However, since we have a graphical authoring tool available (see Sect. 8.1.4), such construction and manipulation operators can be kept simple.

EMMOs encapsulate graph structures of interrelated entities. Therefore, the query algebra should support some basic *subgraph matching* enabling to identify EMMOs encapsulating similar graph structures.

In addition, an EMMO query algebra has to meet basic query algebra requirements. Its operators should be formally defined with *precise semantics* to provide the basis for query rewriting and optimization. Furthermore, the operators should be *orthogonal* and arbitrarily nestable for enabling the formulation of expressive queries, i.e. complex queries can be formulated through the combination and nesting of modular operators.

Finally, because the EMMO model uses concepts of an ontology (i.e. ontology objects) to describe the meaning of the entities contained in an EMMO and the associations between them, a suitable EMMO query algebra should be expressive enough to *integrate ontological knowledge* into a query. Thus, for example, it should be possible to consider supertype/subtype relationships, transitive or inverse associations, etc.

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

4.2 Related Approaches

In this section, we will discuss related approaches to the EMMO query algebra EMMA. Compared to other standards for multimedia content representation (see Sect. 2.2), the EMMO approach is unique in such a way that none of the query languages for those standards can fulfill all the requirements regarding the expressiveness of a language for querying EMMOs. However, valuable aspects of their design have been integrated into the design of the query algebra EMMA.

As the EMMO model describes graph structures, we did not analyze query languages for the relational data model, like SQL. Moreover, as we need to query the instance level, we also did not consider query languages for ontology languages, like DQL [FHH03a] or OWL-QL [FHH03b], which merely address the schema level.

In the following, we will first take a brief look at object-oriented query approaches and query approaches for semi-structured data and multimedia content, before we analyze query approaches for semantic standards in more detail.

4.2.1 Object-Oriented Query Approaches

The object-oriented data model is schema-oriented and represents the data as objects organized in classes. Classes are hierarchically structured and specify relationships between objects. Although object-oriented database systems establish a graph-based data model, the object-oriented data model and the EMMO model are very different from each other, i.e. the former operates on the schema level and gathers objects of a particular type within one class, whereas the EMMO model operates on the instance level, defines no object classes, and leaves the specification of the type semantics to the integrated domain ontology. Moreover, EMMOs provide means for reification, enable the explicit modeling of media data, and provide versioning support. However, as EMMOs establish complex graph structures, the implementation of EMMOs should be based on the object-oriented data model (see Sect. 8.1). Thus, object-oriented query algebras, such as OQL [Cat94], AQUA Algebra [L+93], or XSQL [KKS92], can be used for retrieving the data objects on the implementation level. However, for accessing the information captured by EMMOs, i.e. an EMMO's three aspects and versioning information, one needs an adequate EMMO query approach. Therefore, we did not analyze query languages for object-oriented databases in further detail. However, as any operator accessing information captured by EMMOs, i.e. an EMMO's three aspects and versioning information, is transformed at implementation level to operators of an object-oriented query language, we plan to integrate query optimization strategies for object algebras within our query approach (see Sect. 8.2).

4.2.2 Query Approaches for Semi-Structured Data

Semi-structure data in general, and XML in particular, provide a basis for representing graph-based data structures at the instance level. The Object Exchange Model OEM [PGW95] is a data format that can be used for exchanging arbitrary database structures between applications and is the widely accepted data model for semi-structured data, i.e. data with no assigned schema. Similar to the EMMO model, OEM is schema-less and describes graph structures. Nevertheless, there are many differences between the two data models. OEM does neither address the three aspects of multimedia content, nor provides versioning support. Therefore, query languages for semi-structured data, such as Lorel [A+97], UnQL [BFS00], SAL [BT99], XQuery [B+05b], or XPath [B+05a], are inadequate for querying EMMOs. However, query languages for semi-structured data provide a profound basis for graph navigation by establishing regular path expressions. Thus, we could use those query languages as inspiration for the design of the regular path expressions and navigational operators in EMMA (see Sect. 5.3).

4.2.3 Query Approaches for Multimedia Content

On the search for a suitable query algebra for EMMOs, we take a look at related query algebras and languages in the context of multimedia content, whether they enable to access the complete information captured by EMMOs.

In the literature, several query algebras for multimedia content have been proposed, such as GCalculus/S [L+99], Algebraic Video [DWG94], or the Multimedia Presentation Algebra (MPA) [ASS99]. These algebras have in common that they mainly address the media aspect of multimedia content. They focus on the querying of the temporal and spatial presentation relationships between the basic media of multimedia content and the construction of new presentations out of these media. However, they ignore semantic relationships between media as well as the functional aspect of multimedia content. Therefore, these query approaches are only of limited interest to us, which is the reason why we omit a more detailed discussion in this thesis.

4.2.4 Semantic Query Approaches

In the context of the Semantic Web, several standards have emerged that can be used to model the semantic relationships between the basic media of multimedia content addressing the content's semantic aspect, such as RDF [Bec04, BG04], Topic Maps [ISO00a], Conceptual Graphs [ISO01b], and MPEG-7 (especially MPEG-7's `Graph` tools for the description of content semantics [ISO01a]) (see Sect. 2.2). For these standards, a variety of proposals for query languages and algebras have been made.

Querying RDF

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

Although there exist many query languages for RDF, there is no official standard yet [Mil03]. In the following, we discuss several RDF query approaches that we have analyzed.

As far as we know, there is only one approach, *RAL* [F+02], which provides precise operator semantics for querying RDF and thus a basis for query optimization. By specifying the input and output value as collection of nodes, the operators of the algebra, which are based on a binary labeled graph model, can be arbitrarily combined, and thus optimized. RAL defines a "join operator", but no operator suitable for subgraph matching. As RDF Schema constitutes an RDF description instance, the RAL operators can be used for the navigation along types and subclass edges, therefore enabling the integration of ontological knowledge.

RQL [K+02] is a declarative query language for RDF based on a directed labeled graph model. The input and output values of the queries are specified as RDF graphs, such that new queries can be built through functional composition and iteration. Nevertheless, as the explicit evaluation mechanism of the query language is not specified, the query optimization remains an open issue. RQL provides no subgraph matching and the extension of a query across multiple RDF graphs can only be realized by nesting queries. By defining *schema queries*, e.g. functions to traverse the class/property hierarchies defined in RDF Schema, RQL provides a basis for combining schema with data queries, thus enabling the integration of ontological knowledge.

The RDF query language *SPARQL* [PS04], which is specified as W3C working draft, provides means for accessing RDF graphs, i.e. sets of triples. Although defining query formulas with precise semantics, SPARQL provides no basis for query rewriting. By matching graph patterns against the target graph of the query, SPARQL allows subgraph matching. Moreover, as query forms can be used for querying aggregated graphs, the functionality of a join operator is available. Although – as RDF Schema instances are again RDF documents – SPARQL can be used for querying RDF Schema data, it provides no means for reflecting inferences derived from integrating the ontology knowledge represented in the RDF Schema.

The graph navigation language *SquishQL* [MSR02] establishes a declarative query language for RDF operating on a triple based data model. By only specifying the syntax of its SQL-like operators, query optimization has to be solved at implementation level. SquishQL was the basis for a number of implementations, e.g. RDQL [Sea04], Inkling [Mil02], or RDFStore [RG03]. It is based on subgraph matching, but provides no means for extending queries across multiple RDF graphs, and does not address the integration of ontology knowledge. However, one of its implementations – RDQL – realizes some basic support for the integration of ontological knowledge.

Finally, there are query approaches viewing RDF data as knowledge base of triples, e.g. the RDF query and inference languages *Triple* [SD01] or *Metalog* [MS98]. Both languages are based on Horn Logic and allow to query RDF data on a high level of abstraction that supports inference. Therefore, they merely operate on the schema level, and are not appropriate for querying the instance level.

Querying Topic Maps

The situation for Topic Maps is quite similar as for RDF. The Topic Map data model focuses on the semantic aspect and rather neglects the media and functional aspects of multimedia content. Moreover, although topic maps can be hierarchically nested, they have no explicit versioning support. Consequently, query languages for Topic Maps are generally incomplete and inadequate with regard to the EMMO model.

Within the context of the ongoing standardization of the Topic Maps Query Language (TMQL) [ISO00b], several query approaches have been introduced.

Tolog [Gar03] is a logic based query language combining Prolog and SQL to query topic maps. By only defining the query syntax, issues regarding query optimization are not addressed. The basic building blocks of a tolog query expression are predicates, which are matched to the edges of a topic map's graph structure. More advanced queries can be expressed by combining basic building blocks, thus subgraph matching can be simulated to some extent. Tolog provides no means for extending the query expression across multiple topic maps. As tolog enables the querying of the class-instance relationship, some basic ontology support can be provided.

The approaches *TMPath* [Bog04] and *XTMPath* [BG02] focus on the navigation of Topic Maps. TMPath can be regarded as derivate of XPath [B$^+$05a] – a query approach for XML which has to be embedded into a host language. XTMPath provides a programming technique to navigate through Topic Maps. By specifying the syntax of path expressions, i.e. patterns that can be used to select a subset of a topic map, both approaches provide some limited subgraph matching. As both languages are not constructed as fully fledged query languages, query optimization, a join functionality, and ontology integration are not addressed. .

Another approach for querying Topic Maps, *TOMA* [Pin04], defines the syntax for operators to access the knowledge associated with constituents of topic maps, i.e. topics, associations, and scopes. It provides no dedicated support for subgraph matching. As the semantics of the query operators is not defined, query optimization is rendered impossible. TOMA establishes a SQL-style syntax enabling the extension of queries across multiple topic maps. Finally, by defining operators for the access of type edges, a certain degree of integration of ontology knowledge can be provided.

All the proposals have in common that they remain on the syntactic level and do not provide formal definitions of their operators. No formal algebra with precise semantics as a sound foundation for the querying of Topic Maps exists so far.

Querying Conceptual Graphs

Conceptual Graphs is a knowledge representation language with the expressiveness of first-order logic, which allows to specify query graphs to formulate any database query that can be expressed by SQL [Sow00]. However, again, the Conceptual Graphs specification is merely concerned with the schema level rather than the instance level. As conceptual graphs can be quite complex regarding their expressiveness, there exists quite a lot of research focusing on the development of faster algorithms suitable for subgraph matching [OP98]. However, to the best of our knowledge, no explicit query algebra for Conceptual Graphs has been developed so far.

Querying MPEG-7

Regarding the querying of semantic descriptions of multimedia content on the basis of MPEG-7's `Graph` tools, we find quite a few approaches adapting XQuery [B$^+$05b] for the querying of MPEG-7 media descriptions [MSS02]. However, these approaches do not provide specific operators that would allow a reasonable processing of the `Graph` tools.

4.3 Summary

For the processing of EMMOs and the knowledge they incorporate, suitable querying facilities are required. Therefore, we have developed an EMMO query algebra providing a formal basis for the querying of EMMOs. The most important requirement for such a query algebra is to provide operators for accessing an EMMO's three aspects and versioning information. Moreover, it should enable joins between entities and allow for subgraph matching. To provide a basis for query optimization, its operators need to be orthogonal with precise semantics, and, finally, the query algebra should be expressive enough to integrate ontological knowledge.

To summarize the discussion of related approaches, we have not been able to find a formally sound foundation that would allow an adequate and complete querying of EMMOs.

Both, the object-oriented and semi-structured data model, are very different from the EMMO model. Therefore, object-oriented and semi-structured query approaches are not appropriate for querying EMMOs. However, as the implementation of the EMMO model is based on the object-oriented DBMS ObjectStore, we plan to integrate query optimization strategies for object algebras within our EMMO query approach. Query approaches for semi-structured data provide profound means for navigating graph structures. Therefore, we could use them as inspiration for the design of the navigational operators in EMMA.

Although there are some formal algebras available for querying the media aspect of multimedia content like GCalculus/S, Algebraic Video, or MPA, those approaches do neither address the semantic and functional aspect of multimedia content, nor reflect versioning features.

Regarding the semantic query approaches, especially query languages for RDF and Topic Maps were of particular interest, because they fulfil at least some of our requirements, whereas query approaches for Conceptual Graphs and MPEG-7 `Graph` tools could not be found.

Table 4.1 summarizes to what extent the RDF query approaches provide a basis for query optimization, enable joins and subgraph matching, and facilitate the integration of ontological knowledge. RAL is the only RDF query approach that defines an algebra suitable for query optimization. As can be seen, all four RDF query approaches address the integration of ontological knowledge to some extent. Except SquishQL, the extension of queries across multiple RDF graphs is addressed by all approaches, whereas only SquishQL and SPARQL provide means for subgraph matching.

Table 4.1: Fulfilment of required features by RDF query languages

Requirements + := support, (+) := limited support, – := no support	Query optimization	Joins	Subgraph matching	Integration of ontological knowledge
RAL	+	+	–	+
RQL	–	(+)	–	+
SPARQL	–	+	+	(+)
SquishQL	–	–	+	(+)

Table 4.2 shows the corresponding results for Topic Maps query approaches. None of the four analyzed approaches provides a basis for query optimization, and only TOMA allows the extension of queries across multiple topic maps. Except TOMA all approaches support some basic subgraph matching, and Tolog and TOMA address some basic features enabling the integration of ontological knowledge.

Table 4.2: Fulfilment of required features by Topic Maps query languages

Requirements + := support, (+) := limited support, – := no support	Query optimization	Joins	Subgraph matching	Integration of ontological knowledge
Tolog	–	–	(+)	(+)
TMPath	–	–	(+)	–
XTMPath	–	–	(+)	–
TOMA	–	+	–	(+)

As none of the discussed approaches is suitable for querying the complete information captured by EMMOs, we were forced to develop a dedicated algebra to obtain a sound foundation for querying EMMOs. At least for the design of this algebra, however, we were able to gain valuable insights from the examined approaches and to incorporate aspects of their design.

Chapter 5

EMMA – The EMMO Algebra

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

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

Before we will present the formal basis of the five operator classes in the following sections, we first provide some basic definitions required for the understanding of the definitions to follow.

5.1 Basic Definitions

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

Definition 10 *[Set and Sequence] Let $I\!N$ denote the set of all natural numbers, I an arbitrary index set, $\mathbb{BOO} = \{true, false\}$ the Boolean set, and $\mathbb{SET}$ the set of all sets. Let A, B, and $A_i, i \in I$, be arbitrary sets, then $\mathcal{P}(A) = \{x \,|\, x \subseteq A\}$ denotes the powerset of A, $A \times B := \{(x,y) \,|\, x \in A \wedge y \in B\}$ the Cartesian product over A and B, and $\prod_{i \in I} A_i := \{x : I \longrightarrow \bigcup_{i \in I} A_i \,|\, \forall i \in I : x(i) \in A_i\} \subset (\bigcup_{i \in I} A_i)^I$ the Cartesian product over the sets $A_i, i \in I$. The elements of a Cartesian product are called sequences or tuples. The length of a sequence (tuple) is determined by the number of its contained elements, and a sequence (tuple) of length n is called n-*

sequence (n-tuple). $\mathbb{SEQ}_n$ *denotes the set of all n-sequences, and* $\mathbb{SEQ}$ *the set of all sequences. A sequence of length* 1 *is equal to its single element, i.e.* $\forall x\,(x) = x$. *Let* $j \in I$ *then* $\pi_j : \prod_{i \in I} A_i \longrightarrow A_j$ *with* $\pi_j(a_1, a_2, \dots, a_n) = a_j$ *denotes the* j^{th} *projection of* $\prod_{i \in I} A_i$.

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

Definition 11 *[Function and Predicate] Let* $A, B \in \mathbb{SET}$, *and* $f \in \mathbb{FUN}$ *with* $f : A \longrightarrow B$ *be a function, then* $\mathcal{D}(f) = A$ *denotes the domain and* $\mathcal{R}(f) = B$ *the range of function* f, $\mathbb{FUN}_A$ *the set of all functions with* $\mathcal{D}(f) = A$, *and* $\mathbb{FUN}_{[A,B]}$ *the set of all functions with* $\mathcal{D}(f) = A$ *and* $\mathcal{R}(f) = B$. *Furthermore,* $p \in \mathbb{FUN}_{[A,\mathbb{BOO}]}$ *denotes a predicate,* $\mathbb{PRE}_A = \mathbb{FUN}_{[A,\mathbb{BOO}]}$ *the set of all predicates with domain* A, *and* $\mathbb{PRE} = \{\mathbb{PRE}_A \mid A \in \mathbb{SET}\}$ *the set of all predicates. Let* $f \in \mathbb{FUN}_{\prod_{i \in I} A_i}$, $j \in I, g \in \mathbb{FUN}$ *with* $\mathcal{R}(g) = A_j$, $x \in \mathcal{D}(g)$ *and* $(a_1, \dots, a_{j-1}, a_{j+1}, \dots, a_n) \in \prod_{i \in I \setminus \{j\}} A_i$, *then the function* $f_{[a_1, \dots, a_{j-1}, g(\$), a_{j+1}, \dots, a_n]} : A_j \longrightarrow \mathbb{SET}$ *with* $f_{[a_1, \dots, a_{j-1}, g(\$), a_{j+1}, \dots, a_n]}(x) = f(a_1, \dots, a_{j-1}, g(x), a_{j+1}, \dots, a_n)$ *is called f-projection onto* A_j.

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

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

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

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

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

5.2 Extraction Operators

The extraction operators render it possible to access the information stored within an EMMO. In the following, we introduce the extraction operators for the general properties, the three different aspects, and for the versioning information. We use the EMMOs introduced in Sect. 3.2 as running example to illustrate the usage of EMMA operators. To ease the task of keeping track of the example, we repeat the display of the example EMMOs from Sect. 3.2 where appropriate.

5.2.1 General Properties

Each entity is characterized by several general properties important for its handling and management, i.e. the identifier, the name, the kind, and additional features (see Sect. 3.2). For accessing an entity's global and unique identifier (OID), the operator *oid* can be used, e.g. the query expression

$$oid(l_{nosferatu}) \quad = \quad \text{“}l9462\text{”}$$

returns the universal unique identifier (UUID) of the logical media part "Nosferatu".

Definition 14 *[oid] Let $w \in \Omega$, then the operator oid : $\Omega \longrightarrow \mathbb{UUID}$ is defined as $oid(w) = o_w$.*

The human readable name of an entity can be retrieved by applying the operator *name*, e.g. the query operation

$$name(l_{caligari}) \quad = \quad \text{“The Cabinet of Dr. Caligari”}$$

returns the string value "The Cabinet of Dr. Caligari".

Definition 15 *[name] Let $w \in \Omega$, then the operator name : $\Omega \longrightarrow \mathbb{STR}$ is defined as $name(w) = n_w$.*

An entity can be of kind logical media part, ontology object, association, or EMMO. By applying the operator *kind*, one can find out the kind of an entity, e.g.

$$kind(l_{caligari}) \quad = \quad \text{“lmp”}$$

yields the string "lmp" indicating that the entity "The Cabinet of Dr. Caligari" is a logical media part.

Definition 16 *[kind] Let $w \in \Omega$, then the operator $kind : \Omega \longrightarrow \{\text{“}lmp\text{”}, \text{“}emm\text{”}, \text{“}asso\text{”}, \text{“}ont\text{”}\}$ is defined as $kind(w) = k_w$.*

For the access to the low-level feature-value pairs attached to an entity, the operator *features* can be applied. For example, asking for the set of features of the logical media part "The Cabinet of Dr. Caligari", i.e.

$$features(l_{caligari}) \quad = \quad \{(\text{“timestamp”}, \text{“200412230056”})\},$$

returns its time stamp information.

Definition 17 *[features] Let $w \in \Omega$, then the operator $features : \Omega \longrightarrow \mathcal{P}(\mathbb{STR} \times \mathbb{VAL})$ is defined as $features(w) = F_w$.*

5.2.2 Media Aspect

Logical media parts model media objects at a logical level and maintain connections to their physical representations, i.e. to their media profiles and media selectors. For retrieving all logical media parts contained within an EMMO, the operator *lmp* can be used. For instance, the execution of operator *lmp* with input value e_{movies} yields the three logical media parts "The Cabinet of Dr.

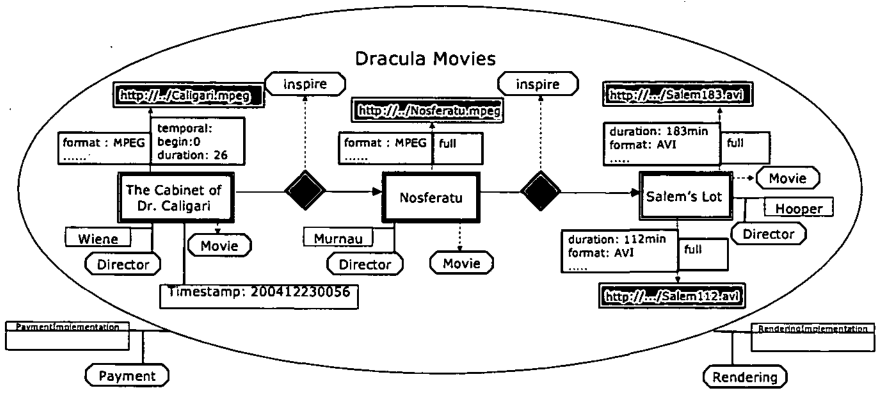


Figure 5.1: EMMO "Dracula Movies"(e_{movies})

Caligari", "Nosferatu", and "Salem's Lot" contained within EMMO "Dracula Movies"in Fig. 5.1:

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

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

For accessing the information described by a logical media part's connectors, EMMA defines several modular operators, as well.as some more complex operators constructed by nesting those modular operators. The operator *MediaProfiles* can be used for locating media profiles. Applying the operator *MediaProfiles* to a logical media part returns the union of all its associated media profiles, e.g. the query expression

$$\begin{aligned} MediaProfiles(l_{salem}) \quad = \quad & \{(\texttt{www.../Salem183.avi}, \{(\textit{"duration"}, \textit{183}), (\textit{"format"}, \textit{"AVI"})\}), \\ & (\texttt{www.../Salem112.avi}, \{(\textit{"duration"}, \textit{112}), (\textit{"format"}, \textit{"AVI"})\})\} \end{aligned}$$

returns a set of two media profiles, each of them consisting of a URI locating the media data and a metadata set describing the low-level characteristics of the media data. The operator *MediaProfiles* is defined as a combination of the operators *connectors* and *MediaProfile.* For a specified entity, the operator *connectors* returns its set of connectors, and the operator *MediaProfile* returns the media profile for a given connector. By using the operators *Apply* and *Elements* in its definition, the operator *MediaProfiles* can be used to access the union of associated media profiles of a logical media part.

Definition 19 *[connectors, MediaProfile, and MediaProfiles] Let $w \in \Omega$, $ms \in \mathcal{MS}$, and $mp \in \mathcal{MP}$, then the operator connectors : $\Omega \longrightarrow \mathcal{P}(\mathcal{MS} \times \mathcal{MP})$ is defined as $connectors(w) = C_w$, the operator $MediaProfile : \mathcal{MS} \times \mathcal{MP} \longrightarrow \mathcal{MP}$ as $MediaProfile((ms, mp)) = mp$, and $MediaProfiles : \Omega \longrightarrow \mathcal{P}(\mathcal{MP})$ as $MediaProfiles(w) = Elements(Apply(MediaProfile, connectors(w)))$.*

The algebra further provides operators to extract the storage location of the media data as well as the metadata from a given media profile, e.g.

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

extracts the storage location, in this example the URI pointing to the media data. Similarly, the operator *Metadata* extracts the physical metadata from the media profile, e.g.

$$\mathit{Metadata}((\texttt{www.../Salem183.avi}, \{(\text{"}\mathit{duration}\text{"}, \mathit{183}), \ldots\})) = \{(\text{"}\mathit{duration}\text{"}, \mathit{183}), \ldots\}.$$

Definition 20 *[MediaInstance and Metadata] Let $mp \in \mathcal{MP}$, then the operator $MediaInstance : \mathcal{MP} \longrightarrow \mathbb{URI} \cup \mathbb{RMD}$ is defined as $MediaInstance(mp) = i_{mp}$, and the operator $Metadata : \mathcal{MP} \longrightarrow \mathcal{P}(\mathbb{STR} \times \mathbb{VAL})$ as $Metadata(mp) = M_{mp}$.*

The operator *MediaSelector* can be used for accessing the information stored within a connector that describes the selected part of the media data. For example, applying the operator *MediaSelector* to the connector attached to the logical media part "The Cabinet of Dr. Caligari", i.e.

$$\begin{aligned} \mathit{connectors}(l_{\mathit{caligari}}) &= \{(ms_1, mp_1)\} \\ &= \{((\text{"temporal"}, \{(\text{"begin"}, 0), (\text{"duration"}, 26)\}), \\ &\qquad (\text{"}\texttt{www.../Caligari.mpeg}\text{"}, \{(\text{"format"}, \text{"MPEG"})\}))\}, \\ \mathit{MediaSelector}((ms_1, mp_1)) &= (\text{"temporal"}, \{(\text{"begin"}, 0), (\text{"duration"}, 26)\}), \end{aligned}$$

yields the media selector specifying a temporal selection from the first until the 26th minute.

Definition 21 *[MediaSelector] Let $w \in \Omega$, $mp \in \mathcal{MP}$, and $ms \in \mathcal{MS}$, then the operator $MediaSelector : \mathcal{MS} \times \mathcal{MP} \longrightarrow \mathcal{MS}$ is defined as $MediaSelector((ms, mp)) = ms$.*

For a given media selector, the operator *Kind* enables the access to its kind information, and the operator *Parameter* the access to the set of parameters. For example, asking for the kind and parameter information of the media selector of the connector of the logical media part "The Cabinet of Dr. Caligari", i.e.

$$\begin{aligned} \mathit{Kind}(ms_1) &= \text{"temporal"}, \\ \mathit{Parameter}(ms_1) &= \{(\text{"begin"}, 0), (\text{"duration"}, 26)\}, \end{aligned}$$

shows that the media selector is a temporal selector starting at the very beginning with a duration of 26 minutes.

Definition 22 *[Kind and Parameter] Let $w \in \Omega$, and $ms \in \mathcal{MS}$, then the operator $Kind : \mathcal{MS} \longrightarrow \{\text{"spatial"}, \text{"textual"}, \text{"temporal"}, \text{"full"}\}$ is defined as $Kind(ms) = k_{ms}$, and the operator $Parameter : \mathcal{MS} \longrightarrow \mathcal{P}(\mathbb{STR} \times \mathbb{VAL})$ is defined as $Parameter(ms) = P_{ms}$.*

5.2.3 Semantic Aspect

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

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

Definition 23 *[types] Let $w \in \Omega$, then the operator types : $\Omega \longrightarrow \mathcal{P}(\Theta)$ is defined as $types(w) = T_w$.*

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

$$attributes(l_{nosferatu}) = \{(o_{director}, \text{“}Murnau\text{”})\},$$

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

Definition 24 *[attributes] Let $w \in \Omega$, then the operator $attributes : \Omega \longrightarrow \mathcal{P}(\Theta \times \mathbb{VAL})$ is defined as $attributes(w) = A_w$.*

EMMOs encapsulate a graph-like knowledge structure of entities. The algebra provides the operator *asso* for accessing all associations contained in an EMMO, e.g. the query expression

$$asso(e_{studies}) = \{a_{va\rightarrow dr}, a_{dr\rightarrow no}, a_{dr\rightarrow moV1}, a_{mi\rightarrow(va\rightarrow dr)}\}$$

returns the associations within EMMO "Dracula Studies"(see Fig. 5.2).

Definition 25 *[asso] Let $w \in \Omega$, then the operator asso : $\Omega \longrightarrow \mathcal{P}(\Lambda)$ is defined as $asso(w) = \{x \mid x \in N_w \cap \Lambda\}$.*

Associations establish a binary directed semantic relationship by describing a source and a target entity. For retrieving the source or target entity of an association, the operator *source*, or respectively, the operator *target* can be used, e.g.

$$\begin{aligned} source(a_{va\rightarrow dr}) &= l_{vampyre}, \\ target(a_{va\rightarrow dr}) &= l_{dracula}. \end{aligned}$$

Definition 26 *[source and target] Let $w \in \Omega$, then the operator $source : \Omega \longrightarrow \Omega \cup \{\epsilon\}$ is defined as $source(w) = s_w$, and the operator $target : \Omega \longrightarrow \Omega \cup \{\epsilon\}$ is defined as $target(w) = t_w$.*

EMMOs establish a container unit that combines several entities into a single package. The encapsulated entities are arranged as semantic graph structure. The remaining operators to be introduced in this subsection focus on accessing the information described by an EMMO's encapsulated graph structure. Therefore, these operators only return a valid output if applied to EMMOs, otherwise

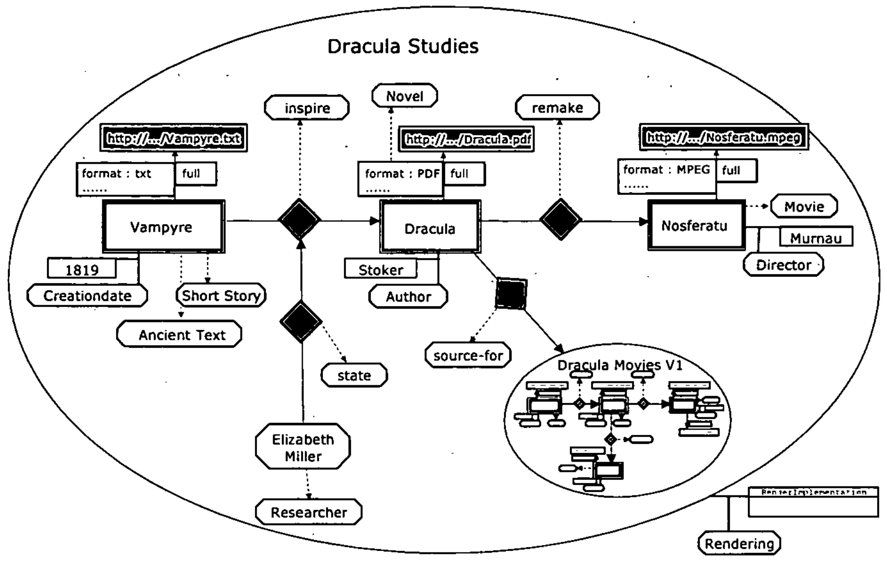


Figure 5.2: EMMO "Dracula Studies" ($e_{studies}$)

an empty result is returned. The operator *nodes* provides the basis for accessing all entities contained within an EMMO, e.g. the query expression

$$\begin{aligned} nodes(e_{studies}) &= \{l_{vampyre}, l_{dracula}, l_{nosferatu}, e_{movies\,V1}, o_{miller}, \\ &\quad a_{va \rightarrow dr}, a_{dr \rightarrow no}, a_{dr \rightarrow moV1}, a_{mi \rightarrow (va \rightarrow dr)}\} \end{aligned}$$

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

Definition 27 *[nodes] Let $w \in \Omega$, then the operator nodes : $\Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $nodes(w) = N_w$.*

Within an EMMO's set of nodes, any kind of entities can be contained. For accessing all logical media parts comprised within the nodes of an EMMO, the operator *lmp* (see Def. 18), and for accessing all associations within an EMMO the operator *asso* (see Def. 25) can be used. Finally, for retrieving all EMMOs or all ontology objects included within an EMMO's nodes, the operators *emm* and *ont* can be employed. For instance, asking for all EMMOs and for all ontology objects described within the nodes of EMMO "Dracula Studies", i.e.

$$\begin{aligned} emm(e_{studies}) &= \{e_{movies\,V1}\}, \\ ont(e_{studies}) &= \{o_{miller}\}, \end{aligned}$$

returns EMMO "Dracula Movies V1", and the ontology object "Elizabeth Miller".

Definition 28 *[emm and ont] Let $w \in \Omega$, then the operator emm : $\Omega \longrightarrow \mathcal{P}(\Sigma)$ is defined as $emm(w) = \{x \mid x \in N_w \cap \Sigma\}$, and the operator ont : $\Omega \longrightarrow \mathcal{P}(\Theta)$ as $ont(w) = \{x \mid x \in N_w \cap \Theta\}$.*

In Sect. 5.3, we will introduce the *navigational operators* enabling the navigation along the edges of an EMMO's graph structure. An EMMO's graph structure is established by the associations between source and target entities. Thus, the entities describing a source or target entity of an association constitute possible starting points of navigation. For accessing all entities used as source entity of an association, the operator *Sources* can be applied. Similarly, the operator *Targets* retrieves all target entities of the associations comprised within an EMMO's nodes. For example, requesting all entities within the nodes of EMMO "Dracula Movies" (Fig. 5.1) being specified as source or target entity of an association, i.e.

$$\begin{aligned} Sources(e_{movies}) &= \{l_{caligari}, l_{nosferatu}\}, \\ Targets(e_{movies}) &= \{l_{nosferatu}, l_{salem}\}, \end{aligned}$$

yields the source entities "The Cabinet of Dr. Caligari" and "Nosferatu", as well as the target entities "Nosferatu" and "Salem's Lot".

Definition 29 *[Sources and Targets] Let $w \in \Omega$, then the operator Sources : $\Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $Sources(w) = \{s \mid \exists x \in asso(w) \quad s = s_x\}$, and the operator Targets : $\Omega \longrightarrow \mathcal{P}(\Omega)$ as $Targets(w) = \{t \mid \exists x \in asso(w) \quad t = t_x\}$.*

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

$$\begin{aligned} AllEncEnt(e_{studies}) &= nodes(e_{studies}) \cup nodes(e_{movies\,V1}) = \\ &= \{l_{vampire}, l_{dracula}, l_{nosferatu}, e_{movies\,V1}, o_{miller}, \\ &\quad a_{va \to dr}, a_{dr \to no}, a_{dr \to moV1}, a_{mi \to (va \to dr)}, \\ &\quad l_{caligari}, l_{salem}, a_{ca \to no}, a_{no \to sa}, a_{(dr \to no)2}\} \end{aligned}$$

unifies the nodes of EMMO "Dracula Studies" (Fig. 5.2) with the nodes of EMMO "Dracula Movies V1" (Fig. 5.3), because this EMMO is the only one contained within EMMO "Dracula Studies" and contains no further EMMOs itself.

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

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

The operator *EncEnt* takes an EMMO e and a natural number n as input, and returns the nodes of EMMO e at n^{th} level. By defining a unification over the operator *EncEnt*, the operator *AllEncEnt* returns, for a specified EMMO, the set of all its recursively contained entities.

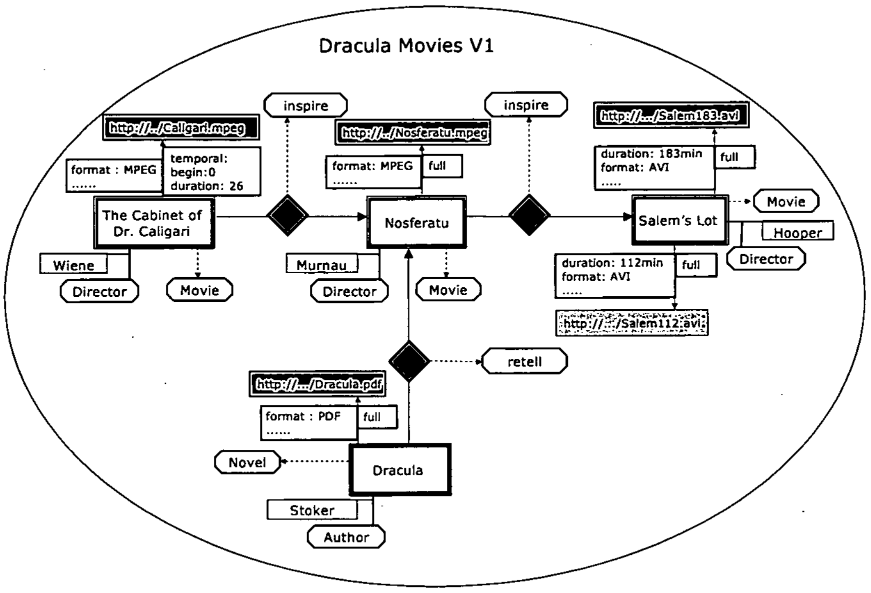


Figure 5.3: EMMO "Dracula Movies V1" ($e_{moviesV1}$)

Definition 30 *[AllEncEnt] Let* $e \in \Omega$, *then the operator* $EncEnt : \Omega \times IN \longrightarrow \mathcal{P}(\Omega)$ *is defined inductively over* IN *as follows:* $EncEnt(e,1) = N_e$, *and by assuming* $EncEnt(e,n)$ *is defined, one defines* $EncEnt(e,n{+}1) = \{x \in \Omega \mid \exists y \in (EncEnt(e,n) \cap \Sigma) \wedge x \in N_y\}$. *The operator* $AllEncEnt : \Omega \longrightarrow \mathcal{P}(\Omega)$ *is defined as* $AllEncEnt(e) = \bigcup_{i \geq 1} EncEnt(e,i)$.

By defining a specialized version of the operators *EncEnt* and *AllEncEnt*, the operators *EncEmm* and *AllEncEmm* enable the direct access to the set of encapsulated EMMOs. For example, asking for all EMMOs recursively contained within EMMO "Dracula Research"(Fig. 5.4), i.e.

$$\begin{aligned} AllEncEmm(e_{research}) &= emm(e_{research}) \cup emm(e_{studies}) = \\ &= \{e_{movies}, e_{moviesV3}, e_{studies}, e_{moviesV1}\}, \end{aligned}$$

yields the three EMMOs contained within the nodes of EMMO "Dracula Research", i.e. the EMMOs "Dracula Movies", "Dracula Movies V3", and "Dracula Studies", and additionally EMMO "Dracula Movies V1", which belongs to the nodes of EMMO "Dracula Studies", and thus is recursively contained within EMMO "Dracula Research".

Definition 31 *[AllEncEmm] Let* $e \in \Omega$, *then the operator* $EncEmm : \Omega \times IN \longrightarrow \mathcal{P}(\Sigma)$ *is defined inductively over* IN *as follows:* $EncEmm(e,1) = emm(e)$, *and by assuming* $EncEnt(e,n)$ *is defined, one defines* $EncEmm(e,n{+}1) = \{x \in \Sigma \mid \exists y \in EncEmm(e,n) \wedge x \in emm(y)\}$. *The operator* $AllEncEmm : \Omega \longrightarrow \mathcal{P}(\Sigma)$ *is defined as* $AllEncEmm(e) = \bigcup_{i \geq 1} EncEmm(e,i)$.

Moreover, the operator *AllEncEmm* can be used for accessing all EMMOs stored within the database. We assume the existence of a database keeping all

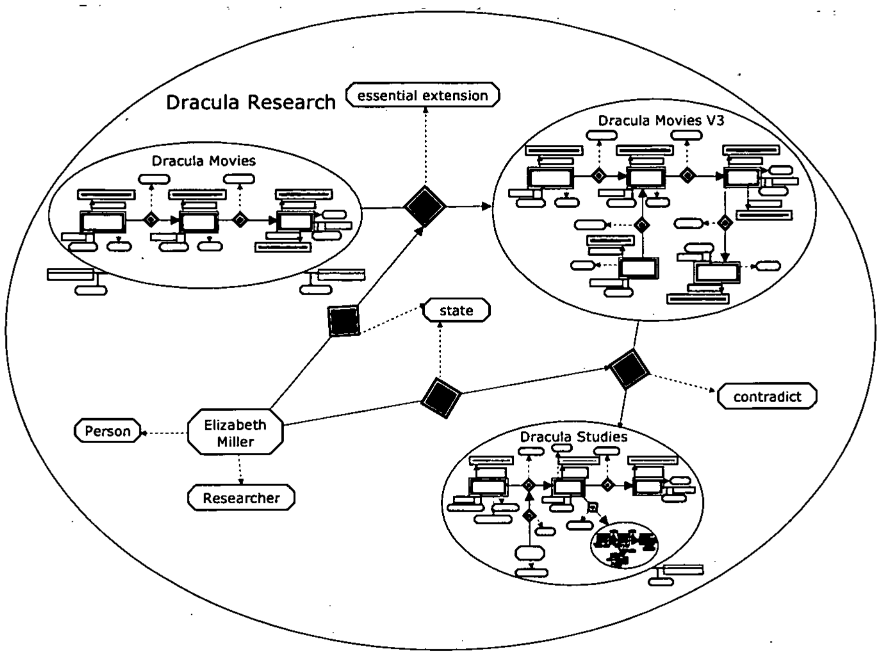


Figure 5.4: EMMO "Dracula Research" ($e_{research}$)

EMMOs. As the query formulation is realized on an abstract level, it is not important whether there exists one central or several distributed but connected databases. Independent of its realization, we simply refer in the following to one database. This database contains one EMMO called "Root" shown in Fig. 5.5, which basically serves as container EMMO for all other EMMOs stored in the database. In other words, every EMMO stored in the database, except EMMO "Root", is either contained or recursively contained in EMMO "Root". Any EMMO which is recursively contained within another EMMO, is not directly contained within the nodes of EMMO "Root", for instance, the EMMOs "Dracula Movies", "Dracula Movie V3", and "Dracula Studies" contained within the nodes of EMMO "Dracula Research" are contained in EMMO "Root" at second level, but not at first level. As EMMO "Root" serves as container structure, which can be compared to a database proxy, it is treated differently from the other EMMOs. For instance, queries asking for all EMMOs with specific characteristics using *selection predicates* (see Sect. 5.5), do not consider the selection of EMMO "Root". This is due to the fact, that applying the operator AllEncEmm to EMMO "Root", i.e.

$$AllEncEmm(e_{root}) \quad = \quad \{e_{movies}, e_{movies\,V1}, e_{movies\,V2}, e_{movies\,V3}, e_{studies}, e_{research}\},$$

yields any EMMO being stored in the database, except EMMO "Root".

5.2.4 Functional Aspect

EMMOs offer functions for handling their content. The operator *operations* can be used for finding all operations attached to an EMMO, e.g. asking for all

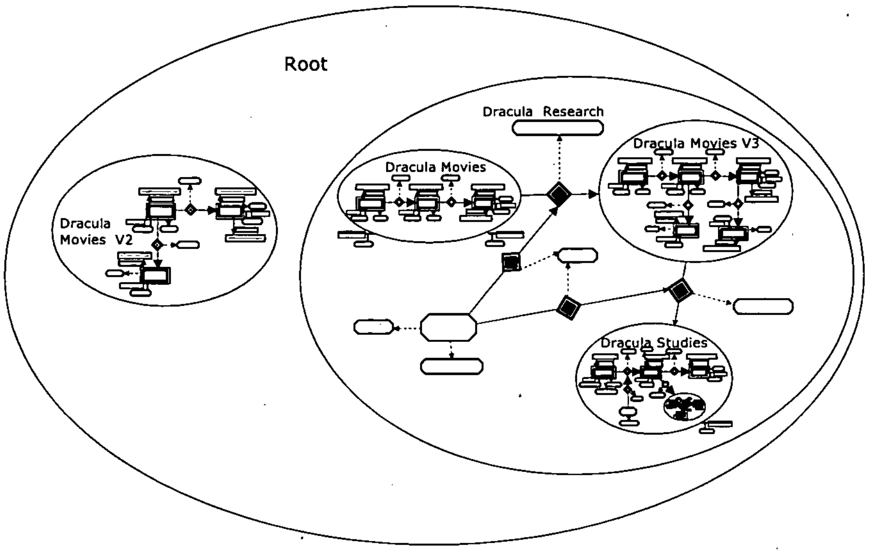


Figure 5.5: EMMO "Root"(e_{root})

operations of EMMO "Dracula Movies"(Fig. 5.1), i.e.

$$operations(e_{movies}) = \{(o_{render}, f_{render}), (o_{payment}, f_{payment})\},$$

returns the two operations describing the EMMO's rendering and payment functionality. For accessing only the ontology objects labeling an EMMO's functions, the operator *Designators* can be used, e.g. the result set of the query

$$Designators(e_{movies}) = \{o_{render}, o_{payment}\}$$

describes the two ontology objects "rendering" and "payment" indicating that EMMO "Dracula Movies" offers a rendering and a payment functionality.

Definition 32 *[operations and Designators] Let $w \in \Omega$, then the operator operations : $\Omega \longrightarrow \mathcal{P}(OP)$ is defined as operations$(w) = O_w$, and the operator Designators : $\Omega \longrightarrow \mathcal{P}(\Theta)$ is defined as Designators$(w) = \{o \in \Theta \mid \exists x \in O_w$ $o = \pi_1(x)\}$.*

The operator *Implementations* enables the access to all mathematical functions specifying the implementations of an EMMO's operations. For example, the request for all mathematical functions attached to EMMO "Dracula Movies", i.e.

$$Implementations(e_{movies}) = \{f_{render}, f_{payment}\},$$

returns the functions f_{render} and $f_{payment}$ describing the implementation of the rendering, and respectively, payment functionality of EMMO "Dracula Movies".

By enabling the access to the mathematical function describing an EMMO's operation labeled by a specific designator, the operator *ImpToName* is more specific than the operator *Implementations*. For example, one might ask for all mathematical functions attached to EMMO "Dracula Movies" which specify a rendering operation, i.e.

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

and receive f_{render} implementing a rendering function.

Definition 33 *[Implementations and ImpToName] Let $w \in \Omega$, then the operator Implementations* : $\Omega \longrightarrow \mathcal{P}(\mathbb{FUN})$ *is defined as Implementations*$(w) = \{f \in \mathbb{FUN} \mid \exists x \in O_w \quad f = \pi_2(x)\}$ *and the operator ImpToName* : $\Omega \times \Theta \longrightarrow \mathbb{FUN}$ *as ImpToName*$(w, o) = \{f \in \mathbb{FUN} \mid \exists x \in O_w \quad o = \pi_1(x) \wedge f = \pi_2(x)\}$.

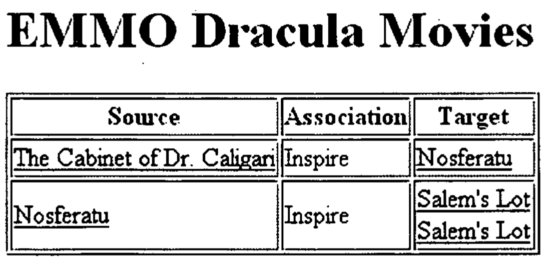
For the execution of an EMMO's functionality, the query algebra EMMA specifies the operator *Execute*. Applying the operator *Execute* to EMMO "Dracula Movies", the ontology object "rendering", and the parameter HTML, i.e.

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

returns an HTML-document representing the content of EMMO "Dracula Movies", e.g. an HTML-document containing a table with the rows being the EMMO's associations as illustrated in Fig. 5.6 and Fig. 5.7.

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

Figure 5.6: DraculaMovies.html



EMMO Dracula Movies

Source	Association	Target
The Cabinet of Dr. Caligari	Inspire	Nosferatu
Nosferatu	Inspire	Salem's Lot Salem's Lot

Figure 5.7: The presentation of DraculaMovies.html

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

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

yields a SMIL-document about the EMMO's content, e.g. a SMIL-document sequentially representing the EMMO's associations as illustrated in Fig. 5.8 and Fig. 5.9.

```
<smil>
<head><layout>
<root-layout height="200" width="620"/>
<region id="1" left="0" ..../> .......
</layout></head>
<body> <seq>
<par end="60s" >
<video src="./Caligari.mpeg" type="video/mpeg" region="1"/>
<text src="./inspire.txt" type="text/plain" region="m"/>
<video src="./Nosferatu.mpeg" type="video/mpeg" region="r"/>
</par>
<par end="60s" >
<video src="./Nosferatu.mpeg" type="video/mpeg" region="1"/>
<text src="./inspire.txt" type="text/plain" region="m"/>
<video src="./Salem183.avi" type="video/mpeg" region="r"/>
</par>
</seq></body>
</smil>
```

Figure 5.8: DraculaMovies.smil

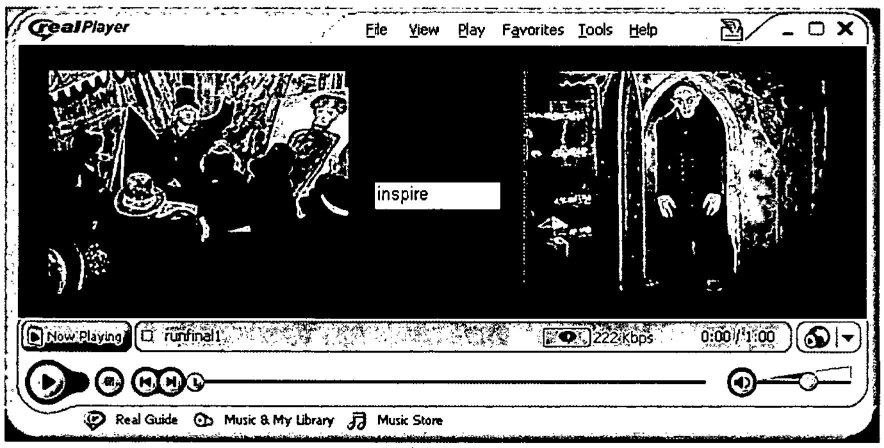


Figure 5.9: The presentation of DraculaMovies.smil

Definition 34 *[Execute] Let* $e \in \Omega$, $op \in \mathcal{OP}$, *and* $s \in \mathbb{SEQ}$, *then the operator* $Execute : \Omega \times \mathcal{OP} \times \mathbb{SEQ} \longrightarrow \mathbb{VAL}$ *is defined as*

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

5.2.5 Versioning

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

$$successors(e_{movies}) = \{e_{movies\,V1}, e_{movies\,V2}\}$$

returns EMMO "Dracula Movies V1" and EMMO "Dracula Movies V2", the two direct successor versions of EMMO "Dracula Movies" (see Fig. 5.10).

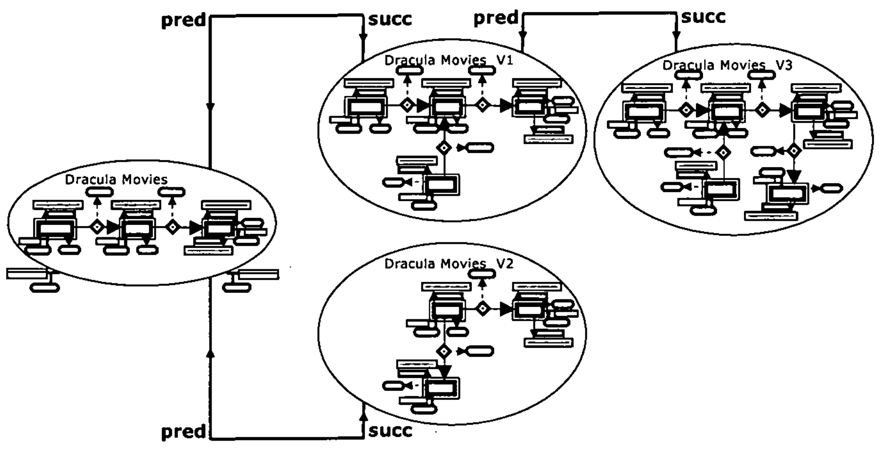


Figure 5.10: EMMO "Dracula Movies"'s versions

Definition 35 *[successors] Let $w \in \Omega$, then the operator $successors : \Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $successors(w) = S_w$.*

For accessing all succeeding versions, the operator *AllSuccessors* is applied, e.g. the query expression

$$AllSuccessors(e_{movies}) = \{e_{movies\,V1}, e_{movies\,V2}, e_{movies\,V3}\}$$

returns again the two direct successors of EMMO "Dracula Movies", and in addition EMMO "Dracula Movies V3", because it is specified as direct successor of EMMO "Dracula Movies V1".

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

Definition 36 *[AllSuccessors] Let $w \in \Omega$, then the operator $Successors : \Omega \times IN \longrightarrow \mathcal{P}(\Omega)$ is defined by induction over IN as follows: $Successors(w, 1) = successors(w)$, and by assuming $Successors(w, n)$ is defined, one defines $Successors(w, n+1) = \{x \in \Omega \mid \exists y \in Successors(w, n) \wedge x \in successors(y)\}$, and the operator $AllSuccessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ as $AllSuccessors(w) = \bigcup_{i \geq 1} Successors(w, i)$.*

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

Definition 37 *[predecessors and AllPredecessors] Let $w \in \Omega$, then the operator $predecessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ is defined as $predecessors(w) = P_w$; the operator $Predecessors : \Omega \times IN \longrightarrow \mathcal{P}(\Omega)$ is defined by induction over IN as follows: $Predecessors(w, 1) = predecessors(w)$, and by assuming $Predecessors(w, n)$ is defined, one defines $Predecessors(w, n + 1) =$ $\{x \in \Omega \mid \exists y \in Predecessors(w, n) \wedge x \in predecessors(y)\}$, and the operator $AllPredecessors : \Omega \longrightarrow \mathcal{P}(\Omega)$ as $AllPredecessors(w) = \bigcup_{i \geq 1} Predecessors(w, i)$.*

5.3 Navigational Operators

An EMMO establishes a graph-like knowledge structure of entities with associations being labeled by ontology objects describing the edges in the graph structure. The navigational operators provide means for traversing the semantic graph structure of an EMMO. Navigation through an EMMO's graph structure is controlled by a navigation path defined as a set of sequences of ontology objects. A mapping for each ontology object in the sequence to the corresponding association of an EMMO defines the traversal path of the graph structure. We have defined *regular path expressions* over ontology objects for describing the syntax of a navigation path. The basic building blocks of regular path expressions are ontology objects which can be modified and combined using conventional regular expression operators.

Definition 38 *[Regular path expression] Given a symbol set $S = \{\varepsilon, \_, +, *, ?, |, -, (,)\}$, an alphabet $\Psi = \Theta \cup S$, and Ψ^*, the set of words over Ψ (finite strings over elements of Ψ). Then, we define $\mathbb{REG} \subseteq \Psi^*$ as the smallest set with the following properties:*

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

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

Navigational operators take a regular path expression as input and specify how this syntactic expression is applied to navigate the graph structure. For example, for a given EMMO, start entity, and regular path expression, the navigational operator *JumpRight* returns the set of all entities that can be reached by traversing the navigation path in the right direction, i.e. by following associations from source to target entities. Applying the operator *JumpRight* to EMMO "Dracula Movies V1"(Fig. 5.3), the starting entity "The Cabinet of Dr. Caligari", and the regular path expression consisting of only one single ontology object $o_{inspire}$ yields the logical media part representing the movie "Nosferatu":

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

In addition to one single ontology object, there exist two other primitive regular path expressions:

- "ε" refers to the empty entity and is interpreted by the operation *JumpRight* as absence of movement, e.g.:

$$JumpRight(e_{moviesV1}, l_{caligari}, \varepsilon) = \{l_{caligari}\}.$$

- "_" refers to any arbitrary ontology object, e.g. (see Fig. 5.11):

$$JumpRight(e_{moviesV2}, l_{nosferatu}, \_) = \{l_{salem}, l_{helsing}\}.$$

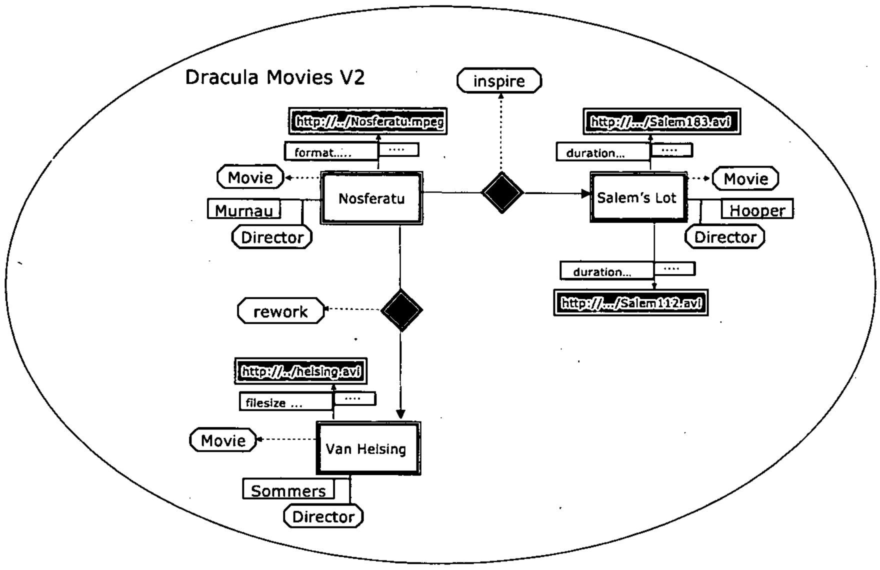


Figure 5.11: EMMO "Dracula Movies V2" ($e_{moviesV2}$)

Regular path expressions may include two operators for the combination of other regular path expressions:

- Regular path expressions can be *concatenated* to specify a longer navigation path, e.g.:

$$JumpRight(e_{moviesV1}, l_{dracula}, o_{retell} o_{inspire}) = \{l_{salem}\}.$$

- "|" allows to combine two regular path expressions as alternative branches, e.g.:

$$JumpRight(e_{moviesV2}, l_{nosferatu}, o_{inspire} \,|\, o_{rework}) = \{l_{salem}, l_{helsing}\}.$$

Finally, there exist four unary operators to modify regular path expressions:

- "?" added to a regular path expression describes its optionality, e.g.:

$$\begin{aligned} & JumpRight(e_{moviesV1}, l_{dracula}, o_{retell} o_{inspire}?) = \\ = \; & JumpRight(e_{moviesV1}, l_{dracula}, o_{retell} \,|\, (o_{retell} o_{inspire})) = \\ = \; & \{l_{nosferatu}, l_{salem}\}. \end{aligned}$$

- "+" defines an iteration of path expressions, which is interpreted as navigation along the same regular path expression any number of times, but at least once, e.g.:

$$JumpRight(e_{moviesV1}, l_{caligari}, o_{inspire}+) = \{l_{nosferatu}, l_{salem}\}.$$

- "*" defines an iteration of path expressions, which is interpreted as navigation along the same regular path expression any number of times, e.g.:

$$\begin{aligned} & JumpRight(e_{moviesV1}, l_{caligari}, o_{inspire}*) = \\ = \; & JumpRight(e_{moviesV1}, l_{caligari}, \varepsilon \,|\, o_{inspire}+) = \\ = \; & \{l_{caligari}, l_{nosferatu}, l_{salem}\}. \end{aligned}$$

- "−" allows to express the inversion of ontology objects, i.e. to follow associations from target to source entities, e.g.:

$$JumpRight(e_{moviesV2}, l_{helsing}, o_{rework}-) = \{l_{nosferatu}\}.$$

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

$$JumpRight(e_{movies\,V2}, l_{helsing}, o_{is-reworked} \,|\, o_{rework}-) = \{l_{nosferatu}\}.$$

The integration of ontology knowledge within the authoring, management, and retrieval of EMMOs will be discussed in full detail in Chapter 7.

The operator *JumpRight*, as formally defined below, takes two entities and one regular path expression as input values. The first input entity specifies the navigation space, the second entity the starting point of navigation, and the regular path expression the set of navigation paths.

Definition 39 *[JumpRight] For $e, w \in \Omega$, and a regular path expression $r \in \mathbb{REG}$, the operator JumpRight : $\Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega)$ is defined as follows:*

$$
\begin{array}{lll}
(1) & \forall r \in \Theta : & \mathit{JumpRight}(e,w,r) = \{x \in N_e \mid \exists y\, y \in \mathit{asso}(e) \wedge \\
 & & \qquad \wedge\, r \in \mathit{types}(y) \wedge w = s_y \wedge x = t_y\} \\
(2) & r = \varepsilon : & \mathit{JumpRight}(e,w,\varepsilon) = \{w \mid w \in N_e\} \\
(3) & r = \_ : & \mathit{JumpRight}(e,w,\_) = \{x \in N_e \mid \exists y \in \mathit{asso}(e) \wedge \\
 & & \qquad \wedge\, w = s_y \wedge x = t_y\} \\
(4) & \forall r_1, r_2 \in \mathbb{REG} & : \mathit{JumpRight}(e,w,r_1 \mid r_2) = \bigcup_{x \in \{r_1,r_2\}} \mathit{JumpRight}(e,w,x) \\
(5) & \forall r_1, r_2 \in \mathbb{REG} & : \mathit{JumpRight}(e,w,r_1 r_2) = \\
 & & = \bigcup_{x \in \mathit{JumpRight}(e,w,r_1)} \mathit{JumpRight}(e,x,r_2) \\
(6) & \forall r \in \mathbb{REG} : & \mathit{JumpRight}(e,w,r?) = \bigcup_{x \in \{r,\varepsilon\}} \mathit{JumpRight}(e,w,x) \\
(7) & \forall r \in \mathbb{REG} : & \mathit{JumpRight}(e,w,r+) = \bigcup_{n \geq 1} JR_n(e,w,r) \quad \mathit{with} \\
 & & JR_n(e,w,r)\, \mathit{defined\ by\ induction\ over\, IN} : \\
 & & \quad JR_1(e,w,r) = \mathit{JumpRight}(e,w,r) \\
 & & \quad JR_n(e,w,r) = \bigcup_{x \in JR_{n-1}(e,w,r)} \mathit{JumpRight}(e,x,r) \\
(8) & \forall r \in \mathbb{REG} : & \mathit{JumpRight}(e,w,r*) = \bigcup_{x \in \{r+,\varepsilon\}} \mathit{JumpRight}(e,w,x) \\
(9) & \forall o \in \Theta : & \mathit{JumpRight}(e,w,o-) = \{x \in N_e \mid \exists y\, y \in \mathit{asso}(e) \wedge \\
 & & \qquad \wedge\, o \in \mathit{types}(y) \wedge x = s_y \wedge w = t_y\}.
\end{array}
$$

The navigational operator *JumpLeft* is the symmetric counterpart to the operator *JumpRight*, i.e. for a given EMMO, starting entity, and regular path expression, the navigational operator *JumpLeft* returns the set of all entities that can be reached by traversing the navigation path in the left direction, i.e. by following associations from target to source entities. Applying the operator *JumpLeft* to EMMO "Dracula Movies V2", the starting entity "Salem's Lot", and the regular path expression consisting of only one single ontology object "$o_{inspire}$" yields the logical media part representing the movie "Nosferatu":

$$\mathit{JumpLeft}(e_{movies\,V2}, l_{salem}, o_{inspire}) = \{l_{nosferatu}\}.$$

Thus, the traversal along the opposite direction of a single association can also be expressed with the navigational operator *JumpLeft*, e.g.

$$\mathit{JumpLeft}(e_{movies\,V2}, l_{helsing}, o_{rework}) \quad = \quad \mathit{JumpRight}(e_{movies\,V2}, l_{helsing}, o_{rework}-).$$

To be more precise, if the *JumpLeft* operator is applied to a regular path expression described by only one single ontology object, the query expression can equivalently be expressed by a query expression using the *JumpRight* operator and the inversion of the regular path expression, i.e. for all EMMOs $e \in \Sigma$, all ontology objects $o \in \Theta$, and all entities w contained within the nodes of e ($w \in N_e$), the following equation holds

$$\mathit{JumpLeft}(e,w,o) = \mathit{JumpRight}(e,w,o-).$$

As the semantic interpretation of the inversion of a regular path expression specified by the operator *JumpRight* is limited to single ontology objects, the operator *JumpLeft* is not only syntactic sugar for the algebra, but adds to its expressivity.

Definition 40 *[JumpLeft] For $e, w \in \Omega$, and a regular path expression $r \in \mathbb{REG}$, the operator $JumpLeft : \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega)$ is defined as follows:*

$$
\begin{array}{lll}
(1) & \forall r \in \Theta: & JumpLeft(e,w,r) = \{x \in N_e \,|\, \exists y\, y \in asso(e) \wedge \\
 & & \qquad \wedge\, r \in types(y) \wedge x = s_y \wedge w = t_y\} \\
(2) & r = \varepsilon: & JumpLeft(e,w,\varepsilon) = \{w \,|\, w \in N_e\} \\
(3) & r = \_: & JumpLeft(e,w,\_) = \{x \in N_e \,|\, \exists y \in asso(e) \wedge \\
 & & \qquad \wedge\, x = s_y \wedge w = t_y\} \\
(4) & \forall r_1, r_2 \in \mathbb{REG}: & JumpLeft(e,w,r_1|r_2) = \bigcup_{x \in \{r_1,r_2\}} JumpLeft(e,w,x) \\
(5) & \forall r_1, r_2 \in \mathbb{REG}: & JumpLeft(e,w,r_1r_2) = \\
 & & = \bigcup_{x \in JumpLeft(e,w,r_1)} JumpLeft(e,x,r_2) \\
(6) & \forall r \in \mathbb{REG}: & JumpLeft(e,w,r?) = \bigcup_{x \in \{r,\varepsilon\}} JumpLeft(e,w,x) \\
(7) & \forall r \in \mathbb{REG}: & JumpLeft(e,w,r+) = \bigcup_{n \geq 1} JL_n(e,w,r) \quad with \\
 & & JL_n(e,w,r)\, defined\ by\ induction\ over\, IN: \\
 & & \quad JL_1(e,w,r) = JumpLeft(e,w,r) \\
 & & \quad JL_n(e,w,r) = \bigcup_{x \in JL_{n-1}(e,w,r)} JumpLeft(e,x,r) \\
(8) & \forall r \in \mathbb{REG}: & JumpLeft(e,w,r*) = \bigcup_{x \in \{r+,\varepsilon\}} JumpLeft(e,w,x) \\
(9) & \forall o \in \Theta: & JumpLeft(e,w,o-) = \{x \in N_e \,|\, \exists y\, y \in asso(e) \wedge \\
 & & \qquad \wedge\, o \in types(y) \wedge x = t_y \wedge w = s_y\}.
\end{array}
$$

The idea and design of the operator *AnchorNodes* is very similar to that of the operators *JumpRight* and *JumpLeft*: Being applied to an EMMO and a regular path expression, the operator *AnchorNodes* retrieves all pairs of entities contained in the EMMO that can be connected by the regular path expression, i.e. starting from the first entity and traversing the navigation path specified by the regular path expression in the right direction by following associations from source to target entities, yields the second entity. For example, asking for all pairs of entities contained within EMMO "Dracula Movies V1" which can be connected by the regular path expression $o_{inspire}+$, i.e.

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

yields the three entity pairs "The Cabinet of Dr. Caligari" and "Nosferatu", "Nosferatu" and "Salem's Lot", and "The Cabinet of Dr. Caligari" and "Salem's Lot".

Definition 41 *[AnchorNodes] For $e \in \Omega$, and a regular path expression $r \in \mathbb{REG}$, the operator $AnchorNodes : \Omega \times \mathbb{REG} \longrightarrow \mathcal{P}(\Omega \times \Omega)$ is defined as follows:*

$$\begin{array}{lll}
(1) & \forall r \in \Theta : & AnchorNodes(e,r) = \{(x,y) \in N_e \times N_e \,|\, \exists a \in asso(e) \wedge \\
 & & \qquad \wedge\, r \in types(a) \wedge x = s_a \wedge y = t_a\} \\
(2) & r = \varepsilon : & AnchorNodes(e,\varepsilon) = \{(x,x) \,|\, x \in N_e\} \\
(3) & r = \_ : & AnchorNodes(e,\_) = \{(x,y) \in N_e \times N_e \,| \\
 & & \qquad \exists a \in asso(e) \wedge x = s_a \wedge y = t_a\} \\
(4) & \forall r_1, r_2 \in \mathbb{REG} : & AnchorNodes(e, r_1 | r_2) = \bigcup_{x \in \{r_1, r_2\}} AnchorNodes(e,x) \\
(5) & \forall r_1, r_2 \in \mathbb{REG} : & AnchorNodes(e, r_1 r_2) = \{(x,y) \in N_e \times N_e \,|\, \exists z \in N_e \\
 & & \qquad ((x,z) \in AnchorNodes(e,r_1) \wedge \\
 & & \qquad \wedge (z,y) \in AnchorNodes(e,r_2))\} \\
(6) & \forall r \in \mathbb{REG} : & AnchorNodes(e, r?) = \bigcup_{x \in \{r, \varepsilon\}} AnchorNodes(e,x) \\
(7) & \forall r \in \mathbb{REG} : & AnchorNodes(e, r+) = \bigcup_{n \geq 1} AN_n(e,w,r) \quad with \\
 & & AN_n(e,r)\, defined\ by\ induction\ over\, \mathbb{N} : \\
 & & \quad AN_1(e,r) = AnchorNodes(e,r) \\
 & & \quad AN_n(e,r) = \{(x,y) \,|\, \exists z \in N_e \quad ((x,z) \in AN_{n-1}(e,r) \\
 & & \qquad \wedge (z,y) \in AnchorNodes(e,r))\} \\
(8) & \forall r \in \mathbb{REG} : & AnchorNodes(e, r*) = \bigcup_{x \in \{r+, \varepsilon\}} AnchorNodes(e,x) \\
(9) & \forall o \in \Theta : & AnchorNodes(e, o-) = \{(\pi_2(x), \pi_1(x)) \,| \\
 & & \qquad x \in AnchorNodes(e,o))\}
\end{array}$$

5.4 Constructors

EMMA specifies five constructors for EMMOs, i.e. the operators *Difference*, *Intersection*, *Union*, *Nest*, and *Flatten*. All the constructors take at least one EMMO and possibly other parameters as input value, and return exactly one EMMO as output value.

The *Difference* operator takes two EMMOs and a string value. It creates a new EMMO which is denoted by the specified string value. The new EMMO's nodes encompass all entities belonging to the first, but not the second EMMO, and additionally, the source and target entities of each association contained within the first EMMO. Applying the *Difference* operator to the successor EMMO "Dracula Movies V1"(Fig. 5.3) and the original EMMO "Dracula Movies"(Fig. 5.1), generates a new EMMO "Newcomers" (see Fig. 5.12) consisting of the logical media parts describing the novel "Dracula" and the movie "Nosferatu", as well as their connecting "retell" association, i.e.

$$\begin{array}{ll}
 & Difference(e_{movies V1}, e_{movies}, \text{“}Newcomers\text{”}) = e_{newcomers} \\
\text{with} & nodes(e_{newcomers}) = \{l_{dracula}, a_{(dr \rightarrow no)2}, l_{nosferatu}\}.
\end{array}$$

Definition 42 *[Difference] Let* $e_1, e_2 \in \Sigma$ *and* $s \in \mathbb{STR}$*, then the operator* Difference $: \Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *is defined as* $\text{Difference}(e_1, e_2, s) = (o_{e_s}, s, \text{“}emm\text{”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, N_{e_s}, \emptyset, \emptyset, \emptyset, \emptyset)$ *with* $o_{e_s} \in \mathbb{UUID}$ *and* $N_{e_s} = \text{nodes}(e_1) \backslash \text{nodes}(e_2) \cup \{x \,|\, \exists y \in \text{asso}(e_1) \backslash \text{asso}(e_2) \quad x = t_y \vee x = s_y\}$.

Given two EMMOs, the operator *Intersection* enables the generation of a new EMMO containing all nodes which belong to both EMMOs. Thus, it becomes possible to generate an EMMO representing all common nodes of, for instance, two different successor versions of an arbitrary EMMO. For example, applying the operator *Intersection* to the two direct successor versions of

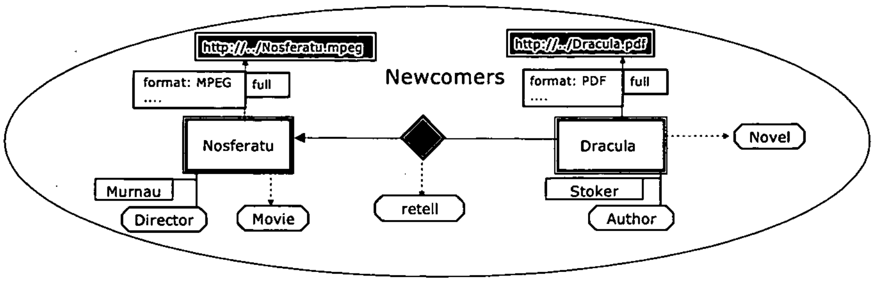


Figure 5.12: EMMO "Newcomers" ($e_{newcomers}$)

EMMO "Dracula Movies" – EMMO "Dracula Movies V1" (Fig. 5.3) and EMMO "Dracula Movies V2" (Fig. 5.11), i.e.

$$Intersection(e_{movies V1}, e_{movies V2}, \text{"}Common\ Nodes\text{"}) = e_{commonnodes}$$
$$\text{with} \quad nodes(e_{commonnodes}) = \{l_{nosferatu}, a_{no \rightarrow sa}, l_{salem}\},$$

constructs a new EMMO containing the two logical media parts "Nosferatu" and "Salem's Lot", as well as their connecting association of type "inspire" (see Fig. 5.13).

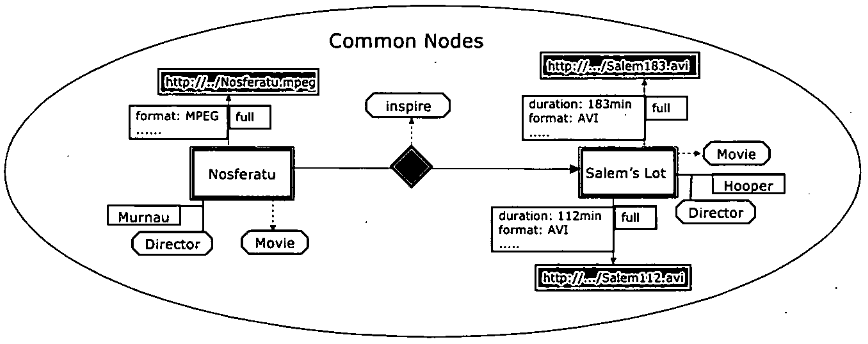


Figure 5.13: EMMO "Common Nodes" ($e_{commonnodes}$)

Definition 43 *[Intersection] Let* $e_1, e_2 \in \Sigma$ *and* $s \in \mathbb{STR}$*, then the operator* Intersection : $\Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *is defined as* Intersection(e_1, e_2, s) = $(o_{e_s}, s, \text{"}emm\text{"}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, N_{e_s}, \emptyset, \emptyset, \emptyset, \emptyset)$ *with* $o_{e_s} \in \mathbb{UUID}$ *and* $N_{e_s} = \text{nodes}(e_1) \cap \text{nodes}(e_2)$.

The operator *Union* can be used for the generation of a new EMMO containing all nodes of two specified EMMOs within its set of nodes. For instance, applying the operator *Union* to the two direct successor versions of EMMO

"Dracula Movies", i.e.

$$\begin{aligned} \mathit{Union}(e_{\mathit{movies\,V1}}, e_{\mathit{movies\,V2}}, \text{"}\mathit{All\ Nodes}\text{"}) &= e_{\mathit{allnodes}} \\ \text{with} \quad \mathit{nodes}(e_{\mathit{allnodes}}) &= \mathit{nodes}(e_{\mathit{movies\,V1}}) \cup \mathit{nodes}(e_{\mathit{movies\,V2}}) = \\ &= \{l_{caligari}, l_{nosferatu}, l_{salem}, l_{dracula}, l_{helsing}, \\ &\quad a_{ca \to no}, a_{no \to sa}, a_{(dr \to no)2}, a_{no \to he}\}, \end{aligned}$$

generates a new EMMO consisting of the five logical media parts describing the novel "Dracula" and the movies "The Cabinet of Dr. Caligari", "Nosferatu", "Salem's Lot", and "Van Helsing", as well as of four connecting associations (see Fig. 5.14).

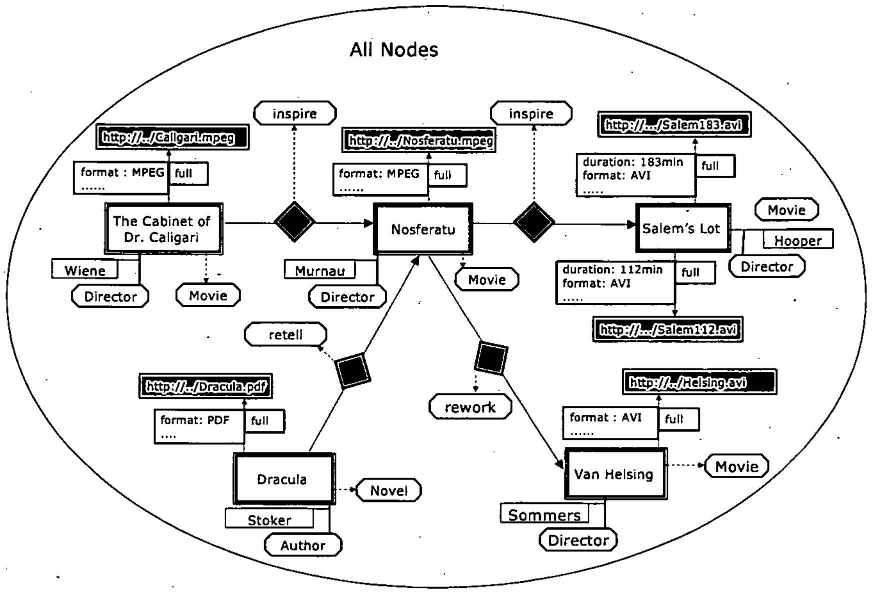


Figure 5.14: EMMO "All Nodes"($e_{allnodes}$)

Definition 44 *[Union] Let* $e_1, e_2 \in \Sigma$ *and* $s \in \mathbb{STR}$*, then the operator* Union : $\Sigma \times \Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *is defined as* Union$(e_1, e_2, s) = (o_{e_s}, s, \text{"}emm\text{"}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, N_{e_s}, \emptyset, \emptyset, \emptyset, \emptyset)$ *with* $o_{e_s} \in \mathbb{UUID}$ *and* $N_{e_s} = \text{nodes}(e_1) \cup \text{nodes}(e_2)$.

The *Nest* operator maps the information stored within an association, i.e. the triple consisting of source entity, association, and target entity, into an EMMO knowledge structure. It takes an EMMO, a string value, and a set of associations as input values and creates a new EMMO which is denoted by the specified string value and includes a subgraph consisting of only the specified associations together with their source and target entities. Applying the *Nest* operator to EMMO "Dracula Studies" (Fig. 5.2) and to the associations which were stated by "Elizabeth Miller"(i.e. the return value of the operation

$JumpRight(e_{studies}, o_{miller}, o_{state}))$:

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

constructs a new EMMO consisting of three entities, i.e. the ancient text “Vampyre”, the novel “Dracula”, and the connecting association of type “inspire” as illustrated in Fig. 5.15.

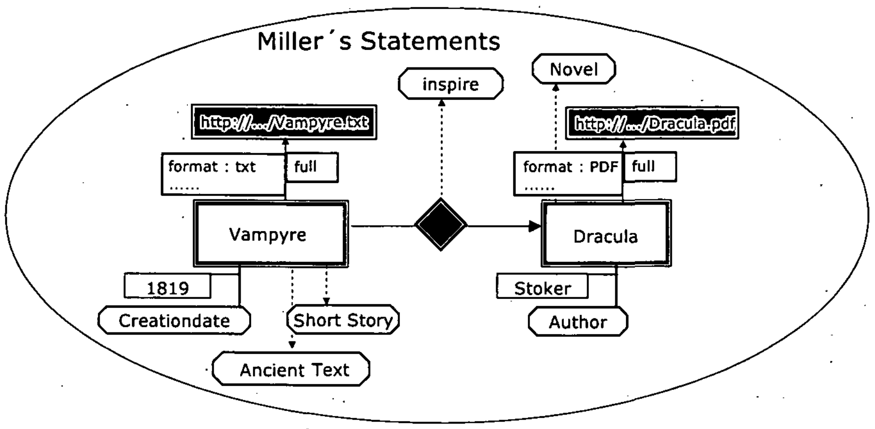


Figure 5.15: EMMO ”Miller's Statements” (e_{miller})

Definition 45 *[Nest] Let* $e \in \Sigma$*,* $A \subseteq \Lambda$*, and* $s \in \mathbb{STR}$*, then the operator* $\mathrm{Nest} : \Sigma \times \mathcal{P}(\Lambda) \times \mathbb{STR} \longrightarrow \Sigma$ *is defined as* $\mathrm{Nest}(e, A, s) = (o_{e_s}, \text{“}s\text{”}, \text{“}emm\text{”}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, N_{e_s}, \emptyset, \emptyset, \emptyset, \emptyset)$ *with* $o_{e_s} \in \mathbb{UUID}$ *and* $N_{e_s} = (A \cap \mathrm{asso}(e)) \cup \{x \mid \exists y \in A \cap \mathrm{asso}(e) \quad x = t_y \vee x = s_y\}$.

Finally, the operator *Flatten* generates a flattened EMMO, i.e. all recursively contained higher level entities are added as first level entities to the nodes of the input EMMO. For example, applying the operator *Flatten* to EMMO “Dracula Studies” (Fig. 5.2) yields a new EMMO encapsulating all entities contained within EMMO “Dracula Studies” unified with all entities contained within EMMO “Dracula Movies V1”, the only EMMO being recursively contained in EMMO “Dracula Studies”, i.e.

$$\begin{aligned}
Flatten(e_{studies}, \text{“}Flatten\ Studies\text{”}) &= e_{flattenstudies} \\
\text{with} \quad nodes(e_{flattenstudies}) &= \mathrm{AllEncEnt}(e_{studies}) = \\
&= nodes(e_{studies}) \cup nodes(e_{movies\,V1}) = \\
&= \{l_{vampyre}, l_{dracula}, l_{nosferatu}, e_{movies\,V1}, o_{miller}, \\
&\quad a_{va\rightarrow dr}, a_{dr\rightarrow no}, a_{dr\rightarrow moV1}, a_{mi\rightarrow(va\rightarrow dr)}, \\
&\quad l_{caligari}, l_{salem}, a_{ca\rightarrow no}, a_{no\rightarrow sa}, a_{(dr\rightarrow no)2}\}
\end{aligned}$$

generates the EMMO “Flatten Studies”, which describes within its set of nodes (see Fig. 5.16) all the entities contained within EMMO “Dracula Studies”, and, in addition, the two logical media parts “The Cabinet of Dr. Caligari” and

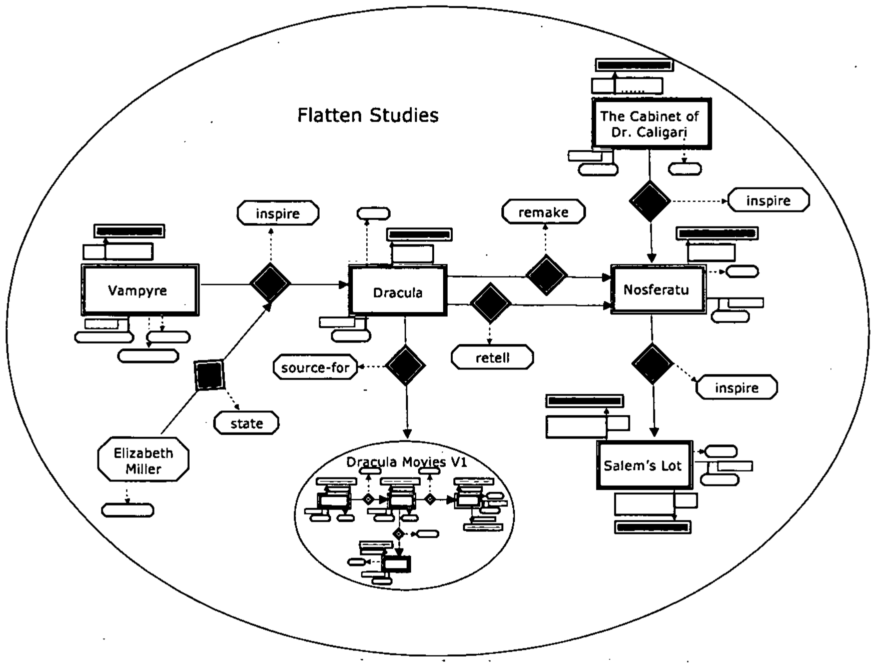


Figure 5.16: EMMO "Flatten Studies"($e_{flattenstudies}$)

"Salem's Lot", as well as three connecting associations taken from EMMO "Dracula Movies V1".

Definition 46 *[Flatten] Let* $e \in \Sigma$ *and* $s \in \mathbb{STR}$, *then the operator* Flatten : $\Sigma \times \mathbb{STR} \longrightarrow \Sigma$ *is defined as*
Flatten$(e, s) = (o_{e_s}, s, \text{"emm"}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, N_{e_s}, \emptyset, \emptyset, \emptyset, \emptyset)$ *with* $o_{e_s} \in \mathbb{UUID}$ *and* $N_{e_s} =$ AllEncEnt(e).

5.5 Selection Predicates

The selection predicates allow the selection of only those entities that satisfy a specific condition. They basically use other operators in their definition to create Boolean predicates. For instance, applying the predicate *IsType* (see Def. 62) to the logical media part "Salem's Lot" and the set containing one ontology object "Movie" returns true:

$$IsType(l_{salem}, \{o_{movie}\}) = true.$$

In the following subsections, we will introduce some basic predicates that enable the combination of selection predicates and other predicates, then we present the *Extraction Selection Predicates* operating on the result values of extraction operators, and, finally, we define the *Navigational Selection Predicates* that use navigational operators in their definitions.

5.5.1 Basic Predicates

The predicates introduced in this subsection, i.e. the *Select* operator, a set of *logical predicates*, a set of *comparison predicates*, and a set of *set predicates*, provide means for combining the selection predicates with other EMMA operators and for enhancing their expressivity.

The selection predicates can be combined with other operators by using the generic *Select* operator, which takes a predicate and an arbitrary set as input values, and returns all elements of the set that satisfy the condition of the specified predicate. For instance, if we apply the *Select* operator to the selection predicate *IsType* (see Def. 62) with the set consisting of the ontology objects "Text"and "Novel" as fixed parameter value and to the set of all logical media parts contained within EMMO "Dracula Studies" (Fig. 5.2), the result set consists of the logical media part representing Stoker's novel "Dracula":

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

Definition 47 *[Select] Let* $A \in \mathbb{SET}$ *and* $p \in \mathbb{PRE}$*, then let the operator* $\mathit{Select} : \mathbb{PRE} \times \mathbb{SET} \longrightarrow \mathbb{SET}$ *be* $\mathit{Select}(p, A) = \{x \mid x \in A \cap \mathcal{D}(p) \wedge p(x)\}$.

For enhancing their expressivity, selection predicates can be combined with the *logical predicates Not, And, Or, Exists,* and *Forall.* The Boolean predicates *Not*, *And*, and *Or* have the usual semantics. For example, one can ask for all logical media parts within EMMO "Dracula Studies" which are not of type "Novel":

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

Definition 48 *[Not, And, and Or] Let* $x, y \in \mathbb{BOO}$*, then the predicate*

$$\begin{array}{ll}
\mathit{Not} : \mathbb{BOO} \longrightarrow \mathbb{BOO} \text{ is defined as} & \mathit{Not}(x) = \begin{cases} true & if \quad x = false \\ false & else, \end{cases} \\
\mathit{And} : \mathbb{BOO} \times \mathbb{BOO} \longrightarrow \mathbb{BOO} \text{ as} & \mathit{And}(x,y) = \begin{cases} true & if \quad x \wedge y \\ false & else, \end{cases} \\
\text{and } \mathit{Or} : \mathbb{BOO} \times \mathbb{BOO} \longrightarrow \mathbb{BOO} \text{ as} & \mathit{Or}(x,y) = \begin{cases} true & if \quad x \vee y \\ false & else. \end{cases}
\end{array}$$

Using the logical predicates *Exists* and *Forall*, one can evaluate whether at least one or all elements of a specified set satisfy a specific condition. For example, the selection of all EMMOs within the database (Fig. 5.5) which contain at least one logical media part of type "Movie", i.e.

$$\begin{aligned}
& \mathit{Select}(\mathit{Exists}_{[\mathit{lmp}(\$),\mathit{IsType}_{[\$,\{o_{movie}\}]}]}, \mathit{AllEncEmm}(e_{root}))) = \\
= \quad & \{e_{movies}, e_{movies V1}, e_{movies V2}, e_{movies V3}, e_{studies}\},
\end{aligned}$$

yields the EMMOs "Dracula Movies", "Dracula Movies V1","Dracula Movies V2","Dracula Movies V3", and "Dracula Studies"; whereas asking for all EMMOs containing only logical media parts which are of type "Movie", i.e.

$$\mathit{Select}(\mathit{Forall}_{[\mathit{lmp}(\$),\mathit{IsType}_{[\$,\{o_{movie}\}]}]}, \mathit{AllEncEmm}(e_{root}))) = \{e_{movies}\}$$

returns EMMO "Dracula Movies".

Definition 49 *[Exists and Forall] Let $A \in \mathbb{SET}$, and $p \in \mathbb{PRE}$, then the predicate Exists* : $\mathbb{SET} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *is defined as*

$$Exists(A,p) = \begin{cases} true & if \quad A \neq \emptyset \wedge \exists x \in A \quad p(x) \\ false & else, \end{cases}$$

and the predicate Forall : $\mathbb{SET} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ *as*

$$Forall(A,p) = \begin{cases} true & if \quad A \neq \emptyset \wedge \forall x \in A \quad p(x) \\ false & else. \end{cases}$$

The *comparison predicates Less, LessEq, Greater, GreaterEq,* and *Equal* provide means for comparing data values. Through the combination with selection predicates and the *Select* operator, they facilitate the precise specification of the condition that forms the basis for the selection. Thus, for example, selection predicates can be used for the selection of all logical media parts within EMMO "Dracula Movies"(see Fig. 5.1) that are associated with a media profile describing media data with a maximum duration of 60 minutes, i.e. the query operation

$$Select(HasMediaProfileValue_{[\$,\text{"duration"},LessEq_{[\$,60]}]}, lmp(e_{movies})) = \{l_{caligari}\},$$

yields the logical media part "The Cabinet of Dr. Caligari" because it has one media profile which contains the attribute "duration" with value 26, i.e. a value which is smaller than 60, in its metadata.

Definition 50 *[Less, LessEq, Greater, GreaterEq, and Equal] Let $x,y \in IR$, then the predicate Less* : $IR \times IR \longrightarrow \mathbb{BOO}$ *is defined as*

$$Less(x,y) = \begin{cases} true & if \quad x < y \\ false & else, \end{cases}$$

the predicate LessEq : $IR \times IR \longrightarrow \mathbb{BOO}$ *as*

$$LessEq(x,y) = \begin{cases} true & if \quad x \leq y \\ false & else, \end{cases}$$

the predicate Greater : $IR \times IR \longrightarrow \mathbb{BOO}$ *as*

$$Greater(x,y) = \begin{cases} true & if \quad x > y \\ false & else, \end{cases}$$

the predicate GreaterEq : $IR \times IR \longrightarrow \mathbb{BOO}$ *as*

$$GreaterEq(x,y) = \begin{cases} true & if \quad x \geq y \\ false & else, \end{cases}$$

and the predicate Equal : $\mathbb{VAL} \times \mathbb{VAL} \longrightarrow \mathbb{BOO}$ *as*

$$Equal(a,b) = \begin{cases} true & if \quad a = b \\ false & else, \end{cases}$$

The *set operators Subset, SubsetEq, Superset, SupersetEq, Contains,* and *Empty* provide means for the handling of sets and their data values. The first four predicates compare two sets. Thus, for example, the operator *SubsetEq* can be used for retrieving all successor versions of EMMO "Dracula Movies" that encompass all logical media parts contained within the nodes of EMMO "Dracula Movies", i.e.

$$Select(SubsetEq_{[lmp(e_{movies}),lmp(\$)]}, AllSuccessors(e_{movie})) = \{e_{movies V1}, e_{movies V3}\}.$$

Definition 51 *[Subset, SubsetEq, Superset, and SupersetEq] Let $A, B \in \mathbb{SET}$, then the predicate $Subset : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ is defined as*

$$Subset(A,B) = \begin{cases} true & if \quad A \subset B \\ false & else, \end{cases}$$

the predicate $SubsetEq : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ as

$$SubsetEq(A,B) = \begin{cases} true & if \quad A \subseteq B \\ false & else, \end{cases}$$

the predicate $Superset : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ as

$$Superset(A,B) = \begin{cases} true & if \quad A \supset B \\ false & else, \end{cases}$$

and the predicate $SupersetEq : \mathbb{SET} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ as

$$SupersetEq(A,B) = \begin{cases} true & if \quad A \supseteq B \\ false & else. \end{cases}$$

The predicate *Contains* checks whether a specified object is contained in a specified set, and the predicate *Empty* whether the specified set is an empty set.

Definition 52 *[Contains and Empty] Let $x \in \mathbb{VAL}$, and $A \in \mathbb{SET}$ then the predicate $Contains : \mathbb{VAL} \times \mathbb{SET} \longrightarrow \mathbb{BOO}$ is defined as*

$$Contains(x,A) = \begin{cases} true & if \quad x \in A \\ false & else, \end{cases}$$

and the predicate $Empty : \mathbb{SET} \longrightarrow \mathbb{BOO}$ as

$$Empty(A) = \begin{cases} true & if \quad A = \emptyset \\ false & else \end{cases}$$

5.5.2 Extraction Selection Predicates

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

General Properties

The predicate *HasOid* enables the access to an entity with a specific oid within an EMMO. For example, asking for the entity with OID "19462" within the nodes of EMMO "Dracula Movies" (see Fig. 5.1), i.e.

$$Select(HasOid_{[\$, \text{“19462”}]}, nodes(e_{movies})) = \{l_{nosferatu}\},$$

yields the logical media part "Nosferatu".

Definition 53 *[HasOid] Let $w \in \Omega$, and $s \in \mathbb{UUID}$, then the predicate $HasOid : \Omega \times \mathbb{UUID} \longrightarrow \mathbb{BOO}$ is defined as $HasOid(w,s) = Equal(oid(w), s)$.*

The predicate *HasName* identifies entities with a specific name, for example, asking for the entity named "The Cabinet of Dr. Caligari" within EMMO "Dracula Movies" (see Fig. 5.1), i.e.

$$Select(HasName_{[\$, \text{“The Cabinet of Dr. Caligari”}]}, nodes(e_{movies})) = \{l_{caligari}\}$$

returns the corresponding logical media part.

Definition 54 *[HasName] Let $w \in \Omega$, and $s \in \mathbb{STR}$, then the predicate $HasName : \Omega \times \mathbb{STR} \longrightarrow \mathbb{BOO}$ is defined as $HasName(w,s) = Equal(name(w), s)$.*

The predicate *IsOfKind* can be used for selecting all entities of a specific kind. For instance, requesting all logical media parts within EMMO "Dracula Movie", i.e.

$$Select(IsOfKind_{[\$, \text{“lmp”}]}, nodes(e_{movies})) = \{l_{caligari}, l_{nosferatu}, l_{salem}\},$$

yields the three logical media parts contained within EMMO "Dracula Movies".

Definition 55 *[IsOfKind] Let $w \in \Omega$, and $s \in \{\text{“emm”}, \text{“asso”}, \text{“ont”}, \text{“lmp”}\}$, then the predicate $IsOfKind : \Omega \times \{\text{“emm”}, \text{“asso”}, \text{“ont”}, \text{“lmp”}\} \longrightarrow \mathbb{BOO}$ is defined as $IsOfKind(w,s) = Equal(kind(w), s)$.*

The predicate *IsOfKind* enables to define the operator *AllEncEmm* (see Def. 31) by using only EMMA operators, i.e. for all EMMOs e holds the following equation:

$$AllEncEmm(e) = Select(IsOfKind_{[\$, \text{“emm”}]}, AllEncEnt(e)).$$

The predicate *HasFeature* takes three input parameter – an entity w, a string value s, and a predicate p – and returns true, if the entity w contains within its features set a name-value pair such that the name is equal to s and the value fulfils the condition described by the specified predicate p. For example, using the predicate *HasFeature* one can access entities that are described by a specific timestamp information, e.g. the query expression

$$\begin{aligned} &Select(HasFeature_{[\$, \text{“Timestamp”}, Equal_{[\$, \text{“200412230056”}]}]}, nodes(e_{movies})) = \\ = \ &\{l_{caligari}\} \end{aligned}$$

returns the logical media part "The Cabinet of Dr. Caligari".

Definition 56 *[HasFeature] Let $w \in \Omega, s \in \mathbb{STR}$, and $p \in \mathbb{PRE}$, then the predicate $HasFeature : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ is defined as*

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

Media Aspect

The predicate *HasMediaProfileValue* enables the selection of logical media parts associated with media data being described by some specific metadata. The predicate takes three input parameters, i.e. an entity w, a string value s, and a predicate p, and returns true, if the entity w contains a media profile with a set of metadata including a name-value pair, with the name being equal to s and the value satisfying the condition described by the specified predicate p. For example, the predicate *HasMediaProfileValue* can be used for identifying all logical media parts within EMMO "Dracula Movies"(see Fig. 5.1) that are associated with a media profile describing media data in AVI format, i.e. the query expression

$$\mathit{Select}(\mathit{HasMediaProfileValue}_{[\$,\text{"format"},\mathit{Equal}_{[\$,\text{"AVI"}]}]}, \mathit{Imp}(e_{movies})) = \{l_{salem}\},$$

yields the logical media part "Salem's Lot".

Definition 57 *[HasMediaProfileValue] Let $w \in \Omega$, $s \in \mathbb{STR}$, and $p \in \mathbb{PRE}$, then $HasMediaProfileValue : \Omega \times \mathbb{STR} \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ is defined as*

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

The predicate *HasMediaInstance* enables the search for logical media parts that describe a specific media object. The predicate takes two input parameters, i.e. the entity w and the media instance d – which is either specified as raw media data or as uniform resource identifier. It returns true, if w is associated with at least one media instance equal to d, i.e. w has a connector specifying the media instance d in its media profile. For instance, a user has already identified the location of the media data he is interested in, i.e. the media instance being located at "`www.../Salem183.avi`", and now intends to search for all logical media parts describing this media instance. By executing the query expression

$$\mathit{Select}(\mathit{HasMediaInstance}_{[\$,\text{"www.../Salem183.avi"}]}, \mathit{AllEncEnt}(e_{root})) = \{l_{salem}\}$$

he receives the logical media part "Salem's Lot" as answer.

Definition 58 *[HasMediaInstance] Let $w \in \Omega$, and $d \in \mathbb{URI} \cup \mathbb{RMD}$, then the predicate $HasMediaInstance : \Omega \times \mathbb{URI} \cup \mathbb{RMD} \longrightarrow \mathbb{BOO}$ is defined as*

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

The predicate *HasMediaSelectorKind* enables the search for logical media parts which describe specific parts of media data. The predicate takes two input parameters, i.e. the entity w and a string value k, and returns true, if w is a logical media part with a media selector of kind k. For example, asking for all logical media parts focusing on the description of segments of movies, i.e.

$$\mathit{Select}(\mathit{HasMediaSelectorKind}_{[\$,\text{"temporal"}]}, \mathit{Select}(\mathit{IsType}_{[\$,\{o_{movie}\}]}, \mathit{nodes}(e_{movies}))) = \\ = \{l_{caligari}\},$$

yields the logical media part "The Cabinet of Dr. Caligari" specifying a temporal selection of the first 26 minutes from the movie.

Definition 59 *[HasMediaSelectorKind] Let $w \in \Omega$, and $k \in \{\text{"spatial"}, \text{"textual"}, \text{"temporal"}, \text{"full"}\}$, then the predicate $HasMediaSelectorKind : \Omega \times \{\text{"spatial"}, \text{"textual"}, \text{"temporal"}, \text{"full"}\} \longrightarrow \mathbb{BOO}$ is defined as*

$$HasMediaSelectorKind(w,k) = \begin{cases} true & if\ \exists c \in C_w \\ & \quad k = Kind(MediaSelector(c)) \\ false & else \end{cases}$$

Semantic Aspect

The predicate *ContainsNode* can be used for identifying EMMOs that contain at least one entity from a specified set of entities in its nodes. The predicate specifies an entity w and a set of entities W as input values, and returns true, if at least one entity of W belongs to the nodes of w. For example, one can search for all EMMOs in the database (Fig. 5.5) which contain at least one entity from the nodes of EMMO "Miller's Statements" (Fig. 5.15), i.e.

$$\begin{aligned} &\mathit{Select}(\mathit{ContainsNode}_{[\$,nodes(e_{miller})]}, \mathit{AllEncEmm}(e_{root})) = \\ = \ &\{e_{studies}, e_{movies\,V1}, e_{movies\,V3}\} \end{aligned}$$

returns the EMMOs "Dracula Studies","Dracula Movies V1", and "Dracula Movies V3".

Definition 60 *[ContainsNode] Let $w \in \Omega$, $W \in \mathcal{P}(\Omega)$, then the predicate $ContainsNode : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

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

The predicate *ContainsAllNodes* enables to select EMMOs by subgraph matching, i.e. the predicate renders it possible to search for EMMOs containing all entities from a specified set of entities. The predicate specifies an entity w and a set of entities W as input values, and returns true, if all entities of W are contained within the nodes of w. For example, all EMMOs within the database (Fig. 5.5) that contain all nodes from EMMO "Dracula Movies"(Fig. 5.1) can be retrieved by the query expression

$$\begin{aligned} &\mathit{Select}(\mathit{ContainsAllNodes}_{[\$,nodes(e_{movies})]}, \mathit{AllEncEmm}(e_{root})) = \\ = \ &\{e_{movies}, e_{movies\,V1}, e_{movies\,V3}\}. \end{aligned}$$

Definition 61 *[ContainsAllNodes] Let $w \in \Omega$, $W \in \mathcal{P}(\Omega)$, then the predicate $ContainsAllNodes : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

$$ContainsAllNodes(w, W) = \begin{cases} true & if \ \ W \subseteq N_w \\ false & else \end{cases}$$

The predicate *IsType*, already used as example in the beginning of this section, can be applied to select entities of a specific type, i.e. entities classified by specific ontology objects. The predicate specifies an entity w and a set of entities O as input values, and returns true, if at least one entity in O is of kind ontology object and contained within the types set of w.

Definition 62 *[IsType] Let $w \in \Omega$, and $O \subseteq \Omega$, then the predicate $IsType : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

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

Similar to the predicate *IsType*, the predicate *EntityContains* focuses on the identification of entities of a specific type. The difference between the two predicates is that predicate *EntityContains* identifies the container units, i.e. EMMOs, which contain at least one entity of a specific type. The predicate specifies an entity e and a set of entities O as input values. It returns true, if e

is an EMMO which contains at least one ontology object in O within the types set of at least one entity from its nodes.

For example, the search for all EMMOs within the database containing entities of type "Novel" can be answered by executing the query expression

$$\begin{aligned} & Select(EntityContains_{[\$,\{o_{novel}\}]}, AllEncEmm(e_{root})) = \\ = \quad & \{e_{studies}, e_{movies V1}, e_{movies V3}\}. \end{aligned}$$

Definition 63 *[EntityContains] Let $e \in \Omega$, and $O \subseteq \Omega$, then the predicate $EntityContains : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

$$EntityContains(e, O) = \begin{cases} true & if \;\; \exists x \in N_e \, \exists o \in O \quad o \in types(x) \\ false & else \end{cases}$$

Furthermore, EMMA defines two selection predicates dealing with an entity's attribute values. The predicate *HasAttribute* enables the selection of entities that possess a specific attribute, e.g. the search for all entities recursively contained within EMMO "Dracula Studies" (Fig. 5.2) which have a "Director" attribute can be expressed by the query expression

$$Select(HasAttribute_{[\$,\{o_{director}\}]}, AllEncEnt(e_{studies})) = \{l_{caligari}, l_{nosferatu}, l_{salem}\}.$$

The predicate $HasAttValue$ is more specific than the predicate *HasAttribute*, because it enables the selection of entities that possess a specific attribute-value pair, e.g. the request for all entities recursively contained within the nodes of EMMO "Dracula Studies"(Fig. 5.2) specifying a "Creationdate" attribute with a value that is smaller than 1950 can be described by

$$Select(HasAttValue_{[\$,\{o_{creationdate}\},Less_{[\$,1950]}]}, nodes(e_{studies})) = \quad \{l_{vampyre}\}.$$

Definition 64 *[HasAttribute and HasAttributeValue] Let $w \in \Omega$, $O \subseteq \Omega$, and $p \in \mathbb{PRE}$, then the predicate $HasAttribute : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

$$HasAttribute(w, O) = \begin{cases} true & if \;\; \exists x \in attributes(w) \, \exists o \in O \quad o = \pi_1(x) \\ false & else, \end{cases}$$

and the predicate $HasAttValue : \Omega \times \mathcal{P}(\Omega) \times \mathbb{PRE} \longrightarrow \mathbb{BOO}$ as

$$HasAttValue(w, O, p) = \begin{cases} true & if \;\; \exists x \in attributes(w) \, \exists o \in O \\ & \qquad (o = \pi_1(x) \wedge p(\pi_2(x))) \\ false & else \end{cases}$$

Functional Aspect

The query algebra also provides selection predicates for identifying entities that satisfy specific conditions with regard to their functional aspect. The predicate *HasDesignator* enables to select all EMMOs with a specific functionality, e.g. asking for all EMMOs in the database (Fig. 5.5) with a rendering function, i.e.

$$Select(HasDesignator_{[\$,\{o_{render}\}]}, AllEncEmm(e_{root})) = \quad \{e_{movies}, e_{studies}\},$$

returns the two EMMOs "Dracula Movies" and "Dracula Studies".

Definition 65 *[HasDesignator] Let $w \in \Omega$ and $O \subseteq \Omega$, then the predicate $HasDesignator : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

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

Versioning

The predicate *ContainsDirectSuccessor* enables the selection of EMMOs that contain the direct successor of a specific entity. It specifies an entity w and a set of entities W as input parameters and returns true if at least one entity of W is a direct successor of w. For example, asking for all EMMOs in the database (Fig. 5.5) that contain a direct successor of EMMO "Dracula Movies" (Fig. 5.1) within its nodes, i.e.

$$Select(ContainsDirectSuccessor_{[e_{movies}, nodes(\$)]}, AllEncEmm(e_{root})) = \{e_{studies}\},$$

yields EMMO "Dracula Studies", as it contains EMMO "Dracula Movies V1" within its nodes.

For accessing all EMMOs that contain any successor of a specified EMMO, the predicate *ContainsSuccessor* can be used. It specifies an entity w and a set of entities W as input parameters and returns true if at least one entity in W is an arbitrary successor of w. For example,

$$Select(ContainsSuccessor_{[e_{movies}, nodes(\$)]}, AllEncEmm(e_{root})) = \{e_{studies}, e_{research}\}$$

returns again EMMO "Dracula Studies" and, additionally, EMMO "Dracula Research" containing EMMO "Dracula Movies V3" within its nodes.

Definition 66 *[ContainsDirectSuccessor and ContainsSuccessor] Let $w \in \Omega$ and $W \in \mathcal{P}(\Omega)$, then the predicate $ContainsDirectSuccessor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

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

and the predicate $ContainsSuccessor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ as

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

The symmetric counterparts of the predicates *ContainsDirectSuccessor* and *ContainsSuccessor*, the predicates *ContainsDirectPredecessor* and *ContainsPredecessor*, are defined analogously:

Definition 67 *[ContainsDirectPredecessor and ContainsPredecessor] Let $w \in \Omega$ and $W \in \mathcal{P}(\Omega)$, then the predicate $ContainsDirectPredecessor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ is defined as*

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

and the predicate $ContainsPredecessor : \Omega \times \mathcal{P}(\Omega) \longrightarrow \mathbb{BOO}$ as

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

5.5.3 Navigational Selection Predicates

The selection predicate *IsRightOf* is based on the navigational predicate *JumpRight* and can be used for selecting end points of a specific navigation path within an EMMO. The predicate *IsRightOf* takes four input values, i.e. three entities e, w_1, and w_2, and a regular path expression $r \in \mathbb{REG}$. It returns true, if e is an EMMO containing w_1 and w_2 within its nodes, such that the navigation along r in the right direction and with starting point w_1 yields w_2. For example, asking for all entities within EMMO "Dracula Movies V1"(Fig. 5.3) which can be reached by navigating in the right direction starting from logical media part "The Cabinet of Dr. Caligari" along one or several associations of type "inspire", i.e.

$$Select(IsRightOf_{[e_{movies V1}, l_{caligari}, \$, o_{inspire}+]}, nodes(e_{movies V1})) = \{l_{nosferatu}, l_{salem}\},$$

yields the two logical media parts "Nosferatu" and "Salem's Lot".

The operator *Targets* (see Def. 29) enables the identification of all entities contained within an EMMO which are specified as target entity of an association. As the set of target entities within an EMMO is equal to the set of all possible end points which can be reached by navigating in the right direction, using the predicate *IsRightOf* in combination with the operator *Targets* helps to reduce the query response time. Within the above query expression, the second input parameter of the *Select* operator is described by the query expression

$$nodes(e_{movies V1}) = \{l_{caligari}, l_{nosferatu}, l_{salem}, l_{dracula}, a_{ca \rightarrow no}, a_{no \rightarrow sa}, a_{(dr \rightarrow no)2}\}.$$

By replacing this input parameter by the query expression

$$Targets(e_{movies V1}) = \{l_{nosferatu}, l_{salem}, l_{dracula}\},$$

query execution time can be reduced, because only entities which are specified as target entities of associations have to be tested as possible end points of navigation. The potential of improving the query performance depends on the size and the structure of the EMMOs.

Additionally, the predicate *IsRightOf* provides means for the selection of EMMOs containing two specific entities being connected by a specific navigation path. For example, the request for all EMMOs within the database (Fig. 5.5) that contain the two logical media parts "The Cabinet of Dr. Caligari" and "Salem's Lot" being connected by one or several associations of type "inspire" can be answered by the query expression

$$\begin{aligned} & Select(IsRightOf_{[\$, l_{caligari}, l_{salem}, o_{inspire}+]}, AllEncEmm(e_{root})) = \\ = \; & \{e_{movies}, e_{movies V1}, e_{movies V3}\}. \end{aligned}$$

Thus, the predicate *IsRightOf* provides the basis for the selection of EMMOs that contain a specific set of sequences of associations connecting a specific pair of entities, i.e. a pair of entities representing the starting and end point of navigation. . EMMOs are modeled as graph structure of connected entities. The use of the *And* predicate enables the selection of EMMOs that contain not only one, but several sets of sequences of associations. Therefore, the predicate *IsRightOf* can be used for subgraph matching. For instance, asking for all EMMOs in the database (Fig. 5.5) that contain the two logical media parts "The Cabinet of Dr. Caligari" and "Salem's Lot" being connected by one or several associations of type "inspire", and the two logical media parts "Dracula" and "Nosferatu" being connected by an association of type "retell", i.e.

$$\begin{aligned} & Select(And(IsRightOf_{[\$, l_{caligari}, l_{salem}, o_{inspire}+]}, IsRightOf_{[\$, l_{dracula}, l_{nosferatu}, o_{retell}]}), \\ & \quad AllEncEmm(e_{root})) = \\ = \; & \{e_{movies V1}, e_{movies V3}\}, \end{aligned}$$

yields the two EMMOs "Dracula Movies V1" and "Dracula Movies V3".

Definition 68 *[IsRightOf] Let* $e, w_1, w_2 \in \Omega$ *and* $r \in \mathbb{REG}$*, then the predicate* $IsRightOf : \Omega \times \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathbb{BOO}$ *is defined as*

$$IsRightOf(e, w_1, w_2, r) = \begin{cases} true & if \ \ w_2 \in JumpRight(e, w_1, r) \\ false & else \end{cases}$$

The symmetric counterpart of the predicate *IsRightOf*, the predicate *IsLeftOf*, provides means to identify EMMOs by specifying a navigation path for the traversal of associations in the left direction, i.e. from target to source entity. Thus, any query operation using the predicate *IsRightOf* can be equivalently expressed by a query operation described by the predicate *IsLeftOf*, i.e. for all logical media parts l_1 and l_2 and for all ontology objects $o_1, o_2, \ldots o_n$, the following equation holds:

$$Select(IsRightOf_{[\$,l_1,l_2,o_1o_2\ldots o_n]}, AllEncEmm(e_{root})) =$$
$$Select(IsLeftOf_{[\$,l_2,l_1,o_n\ldots o_2o_1]}, AllEncEmm(e_{root})).$$

Thus, the predicate *IsLeftOf* can be used for selecting the starting points of a specific navigation path within an EMMO. Similar to the operator *Targets*, the operator *Sources* helps to identify all entities within an EMMO which are source entity of an association. Therefore, again, using the predicate *IsLeftOf* in combination with the operator *Sources* reduces query execution time.

Definition 69 *[IsLeftOf] Let* $e, w_1, w_2 \in \Omega$ *and* $r \in \mathbb{REG}$*; then the predicate* $IsLeftOf : \Omega \times \Omega \times \Omega \times \mathbb{REG} \longrightarrow \mathbb{BOO}$ *is defined as*

$$IsLeftOf(e, w_1, w_2, r) = \begin{cases} true & if \ \ w_2 \in JumpLeft(e, w_1, r) \\ false & else \end{cases}$$

The predicate *ContainsExpr* is based on the operator *AnchorNodes* and can be used for selecting EMMOs that contain entities which are connected by a specific sequence of associations described by a regular path expression. It takes an entity e and a regular path expression r as input value, and returns true, if e is an EMMO which contains a pair of entities within its set of nodes, such that the pair's second entity can be reached by starting from the pair's first entity and traversing the navigation path r in the right direction. For instance, the request for all EMMOs in the database (Fig. 5.5) that contain two entities which are connected by a sequence of one or several associations of type "inspire" can be answered by the query expression

$$Select(ContainsExpr_{[\$,o_{inspire}+]}, AllEncEmm(e_{root})) =$$
$$= \{e_{movies}, e_{movies V1}, e_{movies V2}, e_{movies V3}, e_{studies}\}.$$

Definition 70 *[ContainsExpr] Let* $e \in \Omega$ *and* $r \in \mathbb{REG}$*, then the predicate* $ContainsExpr : \Omega \times \mathbb{REG} \longrightarrow \mathbb{BOO}$ *is defined as*

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

5.6 Join Operator

The *Join* operator renders it possible to extend queries to multiple EMMOs. It specifies how to relate n sets of entities, possibly originating from different EMMOS, within a query. The *Join* operator takes n entity sets, n operators, and one predicate as input values. We compute the Cartesian product of the n entity sets and select only those tuples that satisfy the predicate after applying the n operators to the n entities. The result set of tuples is projected onto the first entry. For example, asking for all successors of EMMO "Dracula Movies"(Fig. 5.1) which constitute an extended version of the original version, i.e. asking for all succeeding EMMOs which at least contain the entities from the original EMMO "Dracula Movie", corresponds to the query expression

$$\begin{aligned} & Join(AllSuccessors(e_{movies}), \{e_{movies}\}, nodes, nodes, SupersetEq) = \\ = \; & \{e_{movies\,V1}, e_{movies\,V3}\} \end{aligned}$$

and yields the two succeeding EMMOs "Dracula Movies V1"(Fig. 5.3) and "Dracula Movies V3" (Fig. 5.17). EMMO "Dracula Movies V1" includes – in addition to the entities already contained within EMMO "Dracula Movies" – two additional entities, i.e. the "retell" association with the logical media part "Dracula" as source entity. EMMO "Dracula Movies V3" contains all the entities from EMMO "Dracula Movies V1", as well as the "similar audience" association with the logical media part "A Return to Salem's Lot" as target entity.

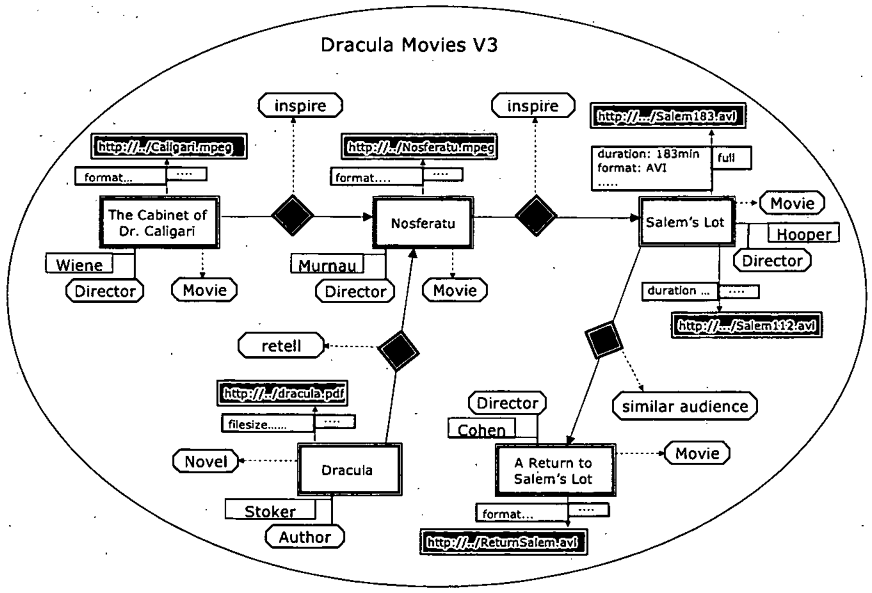


Figure 5.17: EMMO "Dracula Movies V3" ($e_{moviesV3}$)

Definition 71 *[Join] Let* $i \in I = \{1, \dots n\}, n \in IN, W_i \subseteq \Omega, f_i \in \mathbb{FUN}$ *and* $p \in \mathbb{PRE}$*, then the operator Join* : $\prod_{i \in I} \mathcal{P}(\Omega) \times \prod_{i \in I} \mathbb{FUN} \times \mathbb{PRE} \longrightarrow \mathbb{SET}$

is defined as $Join(W_1, \ldots, W_n, f_1, \ldots, f_n, p) = \{\pi_1(w_1, \ldots, w_n) \,|\, \forall i \in I$ $(w_i \in W_i \wedge f_i \in \mathbb{FUN}_{W_i} \wedge p \in \mathbb{PRE}_{\prod_{i \in I} \mathcal{R}(f_i)} \wedge p(f_1(w_1), \ldots, f_n(w_n)))\}$.

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

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

5.7 Summary

In this chapter, we have introduced the formal basis of the query algebra EMMA, which enables the efficient retrieval of the knowledge represented by EMMOs. EMMA's query operators can be divided into five general classes: the extraction operators provide means to query an EMMO's three aspects as well as its versioning information. The navigational operators allow the navigation along an EMMO's semantic graph structure and also facilitate the integration of ontological knowledge. The constructors make it possible to modify, combine, and create new EMMOs. The selection predicates enable the selection of only those entities fulfilling a specific characteristic, and finally, the join operator relates several entities or EMMOs with a join condition. Tables 5.1 and 5.2 summarize the contribution of the EMMA operators and predicates introduced in the preceding sections towards satisfying the requirements of an EMMO query algebra as described in Sect. 4.1.

Table 5.1: EMMA operators addressing the EMMA requirements

general properties	media aspect	semantic aspect	functional aspect	versioning	orthogonality	joins	construction and manipulation	presentation of ontological knowledge
oid	Imp	types	operations	successors	Select	Join	Difference	types
		attributes		predecessors				
name	connectors		Designators		Apply		Union	JumpRight
		asso		AllSuccessors	Elements			JumpLeft
kind	MediaProfile	source	Implementations	AllPredecessors			Intersection	
	MediaProfiles	target	ImpToName		Nest			AnchorNodes
features	Media Instance						Nest	
	Metadata	nodes	Execute					
		emm					Flatten	
	MediaSelector	ont						
	Kind							
	Parameter	Sources						
		Targets						
		AllEncEnt						
		AllEncEmm						

EMMA's operators provide the access to all information and aspects stored within EMMOs and are based on a precise semantics. EMMA features orthogonal and arbitrarily combinable operators. Thus, EMMA offers a formal basis for query rewriting and optimization. Moreover, EMMA operators render it possible to integrate ontological knowledge within queries, such as supertype/subtype

Table 5.2: EMMA predicates addressing the EMMA requirements

general properties	media aspect	semantic aspect	functional aspect	versioning	orthogonality	subgraph matching	presentation of ontological knowledge
		ContainsNode			Not		
HasOid	HasMediaProfileValue	ContainsAllNodes	HasDesignator	ContainsDirectSuccessor	And	ContainsAllNodes	IsType
				ContainsSuccessor	Or		
HasName	HasMediaInstance	IsType			Exists		IsRightOf
		EntityContains			Forall		
IsOfKind	HasMediaSelectorKind			ContainsDirectPredecessor			IsLeftOf
		HasAttribute		ContainsPredecessor	Less/LessEq		
HasFeature		HasAttValue			Greater/GreaterEq		ContainsExpr
					Equal		
					Subset/SubsetEq		
					Superset/SupersetEq		
					Contains		
					Empty		

relationships, transitive or inverse association types. In the following two chapters, we will discuss the issue of integrating ontological knowledge within the authoring, management, and retrieval of EMMOs in more detail.

Chapter 6

Towards the Integration of Ontological Knowledge

Within each application scenario, there exists a shared and common understanding of the domain that can be used for the efficient management of EMMOs, such as authoring of, comfortable access to, or searching for EMMOs. To prevent the generation and storage of useless material, it should be assured during the *authoring* of EMMOs that the interrelations between multimedia resources are compatible with commonly accepted knowledge. In order to facilitate the efficient *access* to EMMOs within a distributed environment, the representation of multimedia content should be flexible in a way that – provided that the semantics of a document is preserved – the user can choose between a compact minimum version being efficient for sending and a maximum version providing optimal browsing and query performance. As user pose imprecise queries, semantic meta information has to be integrated within the *retrieval* process. One way to improve the authoring of, the efficient access to, and the search for EMMOs is to enhance multimedia resources by semantic meta models and to integrate them with *domain ontologies.*

Both, the EMMO model and the EMMA algebra, provide a basis for the integration of a domain ontology, because an EMMO establishes a graph-like *knowledge structure* with associations and nodes being labeled by concepts of the domain ontology, and EMMA defines *navigational operators* to provide means to traverse the ontology-labeled associations within an EMMO's graph structure.

The integration of ontology knowledge into the EMMO model and the EMMA algebra has three appealing benefits:

1. Ontological knowledge can be used for checking integrity constraints during the design and *authoring process of EMMOs*, e.g. to store only associations in the database which conform to the specified types for source and target entities.

2. Ontological knowledge can be incorporated within the *EMMO model* by extending the graph structure of an EMMO with additional associations. For example, if two concepts are stated to be inverse to each other, such as *retell* and *is-retold*, then for each association of one of the two types, an association classified by its inverse counterpart can be added.

3. Knowledge inherent in a domain ontology can be seamlessly integrated into *EMMA queries*. Therefore, the user can pose imprecise queries, which are refined by drawing inferences over the ontological knowledge. For example (see Fig. 5.1), if the user asks for all media objects which have been *inspired* by the movie "The Cabinet of Dr. Caligari", he should also receive media objects which have been indirectly *inspired* by the movie, e.g. the movie "Salem's Lot". This can be accomplished if the transitivity of the ontology object *inspire* is known, i.e. defined in the ontology.

By providing a shared and common understanding of a domain that can be communicated between people and application systems, ontologies facilitate the sharing and reuse of knowledge [Fen01]. Ontologies describe concepts, relationships, and constraints in the domain of discourse. Ontologies can be compared to conceptual schemas in database systems. By defining relations on data, conceptual schemas provide a logical description of shared data enabling application programs and databases to collaborate without specifying the same data structures, whereas ontologies define the vocabulary for composing complex expressions, i.e. a large number of coherent sentences representing the shared knowledge of a particular domain, to be used as additional "background knowledge" input by knowledge-based applications [Gru93].

For the representation and description of ontologies, there exist a large number of representation languages and systems using different syntax and semantics, and diverging in their degree of expressivity. Within a content sharing environment, it is necessary to agree to some common formalism for representing the background knowledge. However, different application domains need an ontology formalism to fulfil different aspects, such as the level of expressivity or the kinds of supported reasoning services. In the following section, we will analyze the requirements for an ontology description language appropriate for the collaborative and distributed authoring and management of multimedia content, i.e. an ontology description language suitable for the EMMO model, before we discuss related approaches and standards for the representation of ontologies.

6.1 Requirements

The EMMO infrastructure establishes a framework for the collaborative and distributed authoring and sharing of multimedia content that can be used by different kinds of application domains, such as the management of cultural knowledge, eLearning platforms, or multimedia enhanced task management systems (see Chapter 9). Although the application domains can be quite different, we identified three common requirements that should be addressed by an ontology description language used within the EMMO infrastructure, i.e. it should maximize its *expressiveness* while still supporting *efficient reasoning* and enable the *exchange and sharing* of ontologies:

1. An ontology description language has to be expressive enough for serious use. Thus, it should provide modeling primitives that enable the description of classes and of *hierarchical structures of classes*, the assignment of things to classes by using a type relationship ensuring that a thing can be member of more than one class and that classes can be again instances

of classes. In addition, it has to provide means for describing relationships between things and *hierarchical structures of relationships*. As one needs to say anything about anything, relationships have to be treated as first-class objects. For avoiding the inappropriate use of relationships, one requires to formulate *constraints* on the usage of classes and relationships, such as domain and range. Besides the formulation of constraints, an ontology description language should allow for the *assignment of properties* to relationships, such as inverse, symmetric, or transitive relationships, and support the formulation of *axioms*.

2. By establishing objects that can be used for logical reasoning, ontologies provide the basis for specifying rules that support certain logical inferences. For enabling an efficient reasoning support, an ontology description language has to be formally defined with *precise semantics* and, if necessary, *limit the expressiveness* of the language.

3. For enabling the exchange and sharing of ontologies, the *syntax* of an ontology description language has to maximize its compatibility with existing and commonly accepted standards. Moreover, the ontology description language should provide a system of unique *identifiers* referring to the established classes, relationships, and instances. In addition, the ontology itself needs to be represented as resource that has its own identifier. Finally, an ontology description language requires some *versioning* support enabling to compare and relate different versions of the same ontology.

6.2 Related Approaches

There exists a long history of research related to knowledge representation. Related approaches and standards for knowledge representation adhere to different language paradigms, such as *first-order logic languages*, *frame-based approaches*, and *Description Logics* [B$^+$03a]. Recently, several *Web standards* for the description of ontologies, such as RDFS, DAML+OIL, or OWL, have emerged.

In the following, we will first introduce one representative example of a first-order logic language, i.e. the Knowledge Interchange Format (KIF), and one representative example of a frame-based approach, i.e. Frame Logic. As the emerging Web Standards are more suitable for fulfilling the requirements of an ontology description language for EMMOs, we will discuss them in more detail.

The *Knowledge Interchange Format (KIF)* [Gen95] is a first-order logic language that was originally developed for the exchange of knowledge between computer programs. It has declarative semantics and is a logically comprehensive language, i.e. it enables to express any arbitrary logical sentence. By supporting meta level statements, KIF establishes an extension of first-order logic. Due to its strong expressiveness, efficient reasoning support cannot be realized. As its derived ontology instances are not serialized in a commonly accepted exchange syntax, such as XML, its concepts have no unique identifiers, and it provides no versioning support, KIF is no appropriate candidate for an ontology representation language within a content sharing and distributed environment.

The frame-based language *Frame Logic* [K$^+$95] can be used for specifying object-oriented databases, Frame Systems, and logical programs. It enables the integration of conceptual modeling constructs, such as classes, subclasses, attributes, domain and range restrictions, or axioms, into a coherent logical framework. Its central modeling primitives are classes, i.e. *frames*, with certain properties called *attributes*. As those attributes are only applicable to one class, they do not have a global scope. Frame Logic provides more complex semantics than first-order logic. However, as in Frame Logic entire formulas cannot be bound to variables, Frame Logic is less expressive than KIF. Frame Logic specifies no global attributes, i.e. relationships are no first-class objects, therefore, it provides the required expressiveness only to some extent. By establishing an extension of first-order logic, efficient reasoning support cannot be realized. Moreover, without commonly accepted serialization syntax, unique identifiers, and versioning support, Frame Logic is not adequate as ontology description language in distributed, content sharing scenarios.

Resource Description Framework Schema (RDFS) [BG04] defines an RDF-based language for describing very simple ontologies, i.e. it establishes basic ontological modeling constructs to be used within RDF. Using the RDFS type system, one can talk about classes and subclasses, about properties and subproperties, and domain and range restrictions of properties. Using the `type` property, RDF objects can be defined as instances of one or more classes. The definition of axioms remains unaddressed. Due to its limited set of modeling primitives, RDFS provides rather restricted expressive power. RDF provides means for reifying statements, i.e. a feature that is difficult to handle within inference services, however, due to its limited expressive power, reasoning support still remains possible. RDFS is based on XML syntax and its objects are uniquely identifiable, but versioning support and unique identifiers for ontology instances are missing. Thus, RDFS is only to some extent appropriate for distributed and content sharing applications. Nevertheless, RDFS provides a standardized syntax and a standard set of basic modeling primitives, such as `subclass of`, `domain` or `range`, for writing ontologies.

DAML+OIL [C$^+$01] evolved by merging two ontology description languages, i.e. *DAML-ONT* [M$^+$03], an early *DARPA Agent Markup Language (DAML)* ontology description language and *Ontology Inference Layer (OIL)* [F$^+$00], a description logic. *Description Logics (DL)* [B$^+$03a] establish logical languages encompassing first-order logic fragments with high expressive power, such that decidability and efficient inference procedures can still be provided, i.e. through the restriction of permitted interactions and combinations of constructors a tractable reasoning can be achieved. DAML+OIL is a Web-based ontology description language that integrates all RDFS modeling constructs and provides additionally all modeling primitives that are commonly used within frame-based languages, i.e. classes, subclasses, axioms, etc. Moreover, it defines relationships not as attributes of classes but as independent classes that can have domain and range restrictions, and can be arranged in a hierarchy. DAML+OIL provides some primitives for describing properties of relationships, such as transitive or inverse relationships, but not for describing symmetric relationships. DAML+OIL axioms are written in KIF format. Although DAML+OIL is equivalent to a very expressive description logic, only partial reasoning support can be provided. This is due to the fact that DAML+OIL has no *unique name assumption*, i.e. individuals with different names not necessarily need to refer to different individuals. DAML+OIL provides XML serialization and identifiers for objects and ontology instances, but an adequate versioning support for ontology instances is missing.

DAML+OIL was a proposed starting point of the W3C project to standardize a Web ontology framework language, i.e. the *Ontology Web Language (OWL)* [SHH04]. OWL specifies three sublanguages:

- *OWL Lite* is suitable for users primarily requiring a classification hierarchy with simple constraints, i.e. thesauri and taxonomies can be expressed in OWL Lite.
- *OWL DL* supports maximum expressiveness by still providing computational completeness and decidability. OWL DL integrates all OWL language constructs, but defines restrictions on their usage, e.g. classes cannot be defined as instances of other classes.
- *OWL Full* achieves maximum expressiveness without computational guarantees, consequently, reasoning support for OWL Full cannot be provided.

Each sublanguage represents an extension of its predecessor language, e.g. any OWL Lite ontology is also an OWL DL ontology. DAML+OIL is very similar to OWL DL, however, we identified three additional features of OWL DL with regard to our requirements. OWL DL adds the modeling primitive for describing symmetric relationships, enables versioning support for ontology instances, and supports the unique name assumption, i.e. by using the modeling constructs `AllDifferent` and `distinctMembers`, one can state that all members of a list are distinct and pairwise disjoint. Thus, OWL DL provides the required expressiveness, is suitable for distributed and content sharing scenarios, and facilitates reasoning support.

6.3 Summary

The integration of ontology knowledge within the management of EMMOs has three appealing benefits, i.e. it can be used for checking integrity constraints during the design and authoring process of EMMOs, enables efficient access to EMMOs, and it provides a basis for query refinement. To represent a suitable basis for the integration of ontological knowledge, an ontology description language for EMMOs has to provide maximum expressiveness while still enabling reasoning support, and support the exchange and sharing of ontologies.

Table 6.1 summarizes the discussion of related approaches with regard to our requirements.

As both, KIF and Frame Logic, establish an extension of first-order logic, they have strong expressiveness. However, by defining its attributes without global scope, Frame Logic possesses the required expressiveness only to some extent. Both, KIF and Frame Logic, provide neither efficient reasoning support, nor the syntax, identifers, and versioning support required within distributed content sharing scenarios. RDFS enables partial reasoning support and is to some extent suitable for content sharing scenarios. However, by only defining 13 ontological modeling primitives, its expressiveness is not sufficient. DAML+OIL provides no means for describing symmetric relationships, therefore its expressiveness is only partly sufficient. It has no unique name assumption and provides no unique identifiers and versioning support for ontology instances, thus reasoning support and the support for distributed content sharing scenarios are only

Table 6.1: Fulfilment of requirements by ontology description languages

Requirements + := support, (+) := limited support, – := no support	KIF	Frame Logic	RDFS	DAML+OIL	OWL DL
Expressiveness	+	(+)	–	(+)	+
Reasoning support	–	–	(+)	(+)	+
Knowledge sharing	–	–	(+)	(+)	+

partially addressed. OWL is a successor of DAML+OIL and has three sublanguages OWL Lite, OWL DL, and OWL Full. OWL Lite is not sufficiently expressive whereas OWL Full has maximum expressiveness, therefore reasoning support is very unlikely. OWL DL is very similar to its predecessor DAML+OIL and a compromise of the two other sublanguages. Through the added modeling primitives, it provides the required expressiveness and reasoning support, and, in addition, is suitable for content sharing and distributed applications. Thus, OWL DL seems to be the most appropriate candidate of an ontology description language within the EMMO infrastructure.

In the following chapter, we will show how ontological knowledge can be used for integrity constraint checking within the design and authoring process of EMMOs, the enhancing of the semantic expressiveness of the knowledge structures defined by EMMOs, and the refining of EMMA query expressions.

Chapter 7

Integrating Ontological Knowledge into the EMMO Model

In the following, we define an ontology structure suitable for the EMMO model, describe how the most common modeling constructs used in standard ontology languages like DAML+OIL [C+01] or OWL [SHH04] can be represented within this structure, and exemplify how the ontology knowledge can be integrated into the authoring process of EMMOs (Sect. 7.1), into the knowledge structures described by the EMMO model (Sect. 7.2), and into the EMMA queries (Sect. 7.3). We will use EMMO "Dracula Movies V3" (see Fig. 7.1) as running example throughout this chapter.

The definition of an ontology structure for EMMOs was inspired by the ontology structure definition in [Mae02]. Any concept of the ontology which is used for labeling entities in the EMMO model is represented as ontology object within the EMMO model. As the EMMO model treats associations as first class objects, ontology objects can be used for labeling both, the nodes and the edges, in an EMMO's graph structure. We specify an *ontology structure* suitable for the EMMO model as 3-tuple $\mathcal{O} = \{\Theta, \mathcal{H}^{\Theta}, \mathcal{A}^{\mathcal{O}}\}$ consisting of

- a *set of ontology objects* Θ, representing the concepts of the ontology,
- a *concept hierarchy* $\mathcal{H}^{\Theta}$ describing the subclass relationship between ontology objects, i.e. $\mathcal{H}^{\Theta}$ is a directed relation $\mathcal{H}^{\Theta} \subseteq \Theta \times \Theta$ with $\mathcal{H}^{\Theta}(o_1, o_2)$ expressing that o_1 is a subconcept of o_2,
- a set of *ontology axioms* $\mathcal{A}^{\mathcal{O}}$, expressed in first order logic.

Figure 7.2 illustrates a small portion of the Ontology of Intertextual Studies used in the CULTOS project (see Sect. 9.1) as defined in [B+03b].

An ontology suitable for the integration into the EMMO model and the EMMA algebra has to distinguish between *object concepts* and *relational concepts.* Object concepts are used for labeling the nodes, relational concepts for labeling the associations within an EMMO's graph structure. For example, the Ontology of Intertextual Studies defines object concepts for describing media

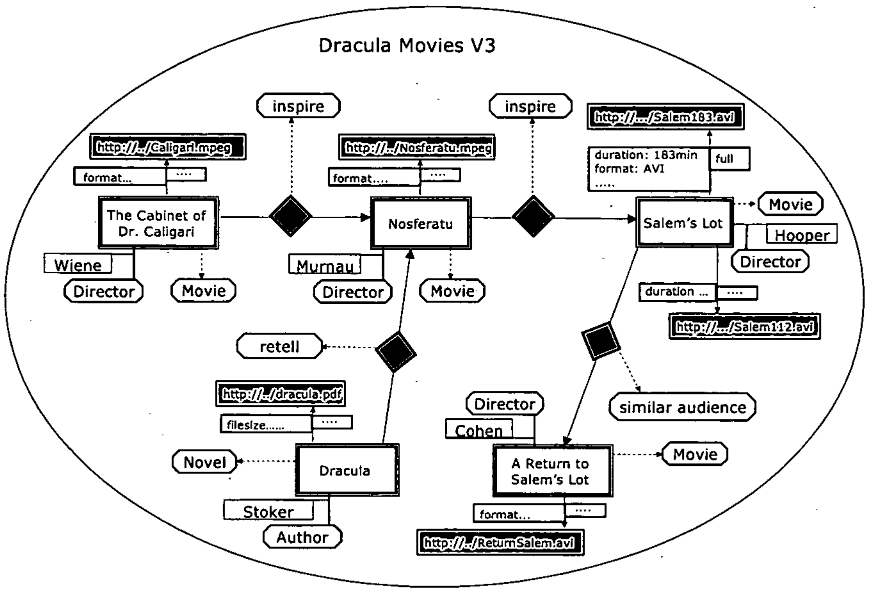


Figure 7.1: EMMO "Dracula Movies V3" ($e_{moviesV3}$)

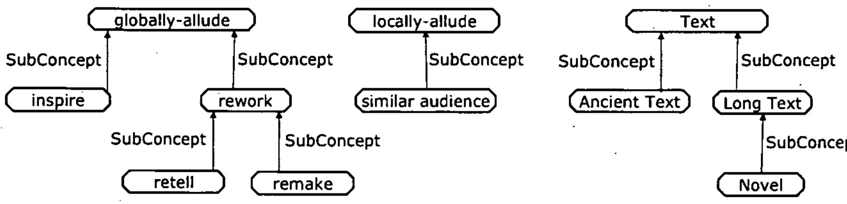


Figure 7.2: Extract from the Ontology of Intertextual Studies

objects, e.g. the concepts *Novel* or *Text*, and relational concepts for describing relationships holding between media objects, e.g. the relational concept *inspire* can be used for describing the fact that an ancient source text *inspires* a particular movie.

The set of ontology axioms $\mathcal{A}^{\mathcal{O}}$ allows to specify properties and restrictions of concepts, and to define properties of relationships between concepts. Thus, we can specify that some specific ontology objects are dedicated to describe associations within the EMMO model, i.e. representing relational concepts, e.g. (see Fig. 7.2)

$$\begin{aligned}(\{&o_{globally-allude}, o_{inspire}, o_{rework},\\ &o_{retell}, o_{remake}, o_{locally-allude}, o_{similar}\}) \subseteq \mathcal{RC}) \in \mathcal{A}^{\mathcal{O}},\end{aligned} \tag{7.1}$$

with $\mathcal{RC} = \{o \in \Theta \mid \forall w \in \Omega \wedge o \in types(w) \rightarrow w \in \Lambda\}$ describing the set of *relational concepts*, Ω the set of all entities, Λ the set of all associations, and $types(w)$ the set of ontology objects labeling the entity w. In a similar way, we can specify that some ontology objects are used exclusively for describing nodes of the EMMO graph structure, e.g. (see Fig. 7.2)

$$(\{o_{text}, o_{ancienttext}, o_{longtext}, o_{novel}\} \subseteq \mathcal{OC}) \quad \in \quad \mathcal{A}^{\mathcal{O}}, \tag{7.2}$$

with $\mathcal{OC} = \{o \in \Theta \mid \forall w \in \Omega \wedge o \in types(w) \rightarrow w \notin \Lambda\}$ describing the set of *object concepts*.

Furthermore, within the set of ontology axioms, we can define the transitivity of the concept hierarchy, i.e.

$$(\forall o_1, o_2, o_3 \in \Theta \quad \mathcal{H}^{\Theta}(o_1, o_2) \wedge \mathcal{H}^{\Theta}(o_2, o_3) \rightarrow \mathcal{H}^{\Theta}(o_1, o_3)) \in \mathcal{A}^{\mathcal{O}}. \tag{7.3}$$

Based on this axiom, we can now infer from the ontology that the concept *Novel*, which is a direct subconcept of *Long Text*, is also a subconcept of *Text*, the superconcept of *Long Text*.

7.1 Ontology-enhanced Authoring of Multimedia Content

During the design and authoring process of EMMOs, we can use ontological knowledge for checking integrity constraints, i.e. only associations that coincide with the specified types regarding source and target entity can be stored in the database. For instance, within an ontology structure one can specify that associations of type *retell* describe binary relationships pointing from entities of type *Text* to entities of arbitrary type, i.e.

$$(Domain(o_{retell}) = \{o_{text}\}) \quad \in \quad \mathcal{A}^{\mathcal{O}}, \tag{7.4}$$

with $Domain(o) = \{x \in \Theta \mid \forall a \in \Lambda \wedge o \in types(a) \rightarrow x \in types(source(a))\}$ describing the set of *domain concepts*, i.e. concepts which can be used for classifying the source entity of associations of type o, and $source(a)$ denoting the source entity of association a (see Fig. 7.3).

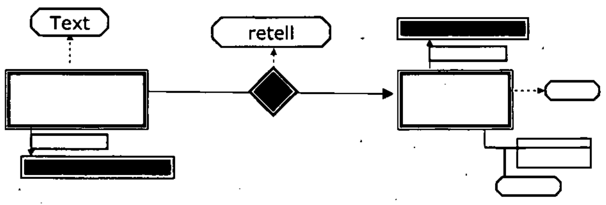


Figure 7.3: Association "retell" with source entity of type "Text"

Let us assume that a user intends to store the EMMO "Dracula Movies V3" (see Fig. 7.1) in the database. Due to integrity constraints checking based on Axiom 7.4, the EMMO's *retell* association, which in this example specifies a source entity of type *Novel* but not of type *Text*, will be removed from the EMMO's nodes before storing the EMMO.

In order to reflect the ontological knowledge about the concept hierarchy, such as the knowledge that *Novel* is a subconcept of *Long Text* which again is a subconcept of *Text*, during the checking of integrity constraints, we can specify the following axiom

$$\begin{aligned}&(\forall o \in \mathcal{RC} \quad \forall O \subseteq \Theta \quad \mathit{Domain}(o) = O \\ &\quad \Longrightarrow \mathit{Domain}(o) = \mathit{SubConcepts}(O)) \quad \in \quad \mathcal{A}^{\mathcal{O}},\end{aligned} \tag{7.5}$$

with $\mathit{SubConcepts}(O) = \{x \in \Theta \mid \exists o \in O \quad (x, o) \in \mathcal{H}^{\Theta}\}$ describing the set of all subconcepts of a set of ontology objects O.

By integrating Axioms 7.3–7.5 within the authoring process of EMMOs, EMMO "Dracula Movies V3" can now be stored in the database without removing the *retell* association (see Fig. 7.4).

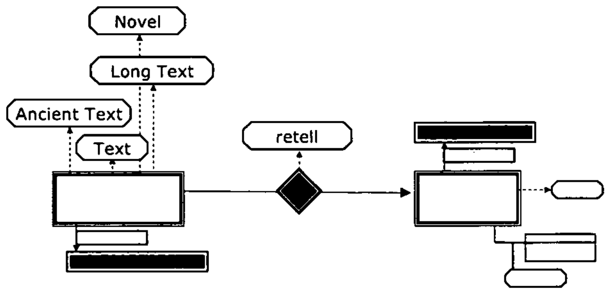


Figure 7.4: Association "retell" with source entity of type $SubConcepts(\{o_{text}\})$

In a similar way, we define axioms for constraints on *range concepts*, i.e. concepts used for classifying the target entity of associations, e.g.

$$(\mathit{Range}(o_{retell}) = \{o_{movie}\}) \quad \in \quad \mathcal{A}^{\mathcal{O}} \quad \text{and} \tag{7.6}$$

$$\begin{aligned}&(\forall o \in \mathcal{RC} \quad \forall O \subseteq \Theta \quad \mathit{Range}(o) = O \\ &\quad \Longrightarrow \mathit{Range}(o) = \mathit{SubConcepts}(O)) \quad \in \quad \mathcal{A}^{\mathcal{O}},\end{aligned} \tag{7.7}$$

with $\mathit{Range}(o) = \{x \in \Theta \mid \forall a \in \Lambda \land o \in \mathit{types}(a) \rightarrow x \in \mathit{types}(\mathit{target}(a))\}$ describing the set of concepts which can be used for classifying the target entity of associations of type o and $target(a)$ denoting the target entity of association a.

7.2 Ontology-enhanced Management of Multimedia Content

Ontological knowledge can be used within the management of multimedia content by integrating it into the EMMO model by extending the graph structure with additional associations. In the following, we will exemplify how the ontological knowledge about the subconcept hierarchy, as well as about inverse, transitive, and symmetric relational concepts can be integrated within the EMMO model. Furthermore, we will introduce the axioms describing this ontological knowledge. All the examples are based on the ontology structure illustrated in Fig. 7.2 and the EMMO "Dracula Movies V3" depicted in Fig. 7.1.

Integrating the Knowledge about the Subconcept Hierarchy

In Axiom 7.3 we have defined the transitivity of the concept hierarchy, and by adding the axiom

$$(\{(o_{rework}, o_{globally-allude}), (o_{retell}, o_{rework}), (o_{remake}, o_{rework}), \ldots\} \subseteq \mathcal{H}^{\Theta}) \in \mathcal{A}^{\mathcal{O}}, \tag{7.8}$$

we specify that the concepts *remake* and *retell* are subconcepts of *rework*, which is again a subconcept of *globally-allude*. The explicit integration of this knowledge into the EMMO model can be realized by adding the associations $(l_{dracula} \overset{o_{rework}}{\longrightarrow} l_{nosferatu})$ and $(l_{dracula} \overset{o_{globally-allude}}{\longrightarrow} l_{nosferatu})$ to the EMMO "Dracula Movies V3" (see Fig. 7.5).

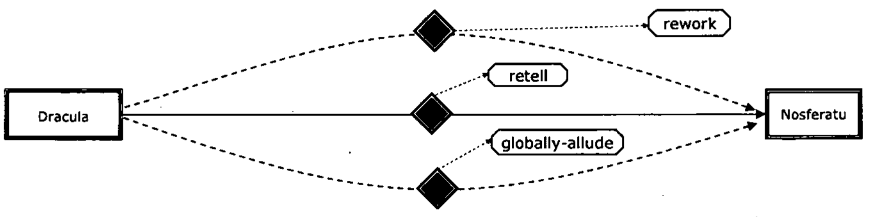


Figure 7.5: Integrating the knowledge about *retell*'s superconcepts

Integrating the Knowledge about Transitive Concepts

Within the ontology axioms, we can also define *transitive concepts*, i.e. relational concepts for which an iteration of the navigation along the corresponding associations can be defined without changing the semantics of the concept, e.g.

$$(o_{inspire} \in \Theta_{\text{TRANS}}) \in \mathcal{A}^{\mathcal{O}}, \tag{7.9}$$

with $\Theta_{\text{TRANS}} = \{o \in \mathcal{RC} \,|\, \forall a_1, a_2 \in I(o) \quad target(a_1) = source(a_2) \rightarrow \exists a_3 \in I(o)\,(source(a_3) = source(a_1) \wedge target(a_3) = target(a_2))\}$ describing the set of all transitive ontology objects, $I(o) = \{w \in \Omega \,|\, o \in types(w)\}$ the set of all entities labeled by the ontology object o, and $source(a)$ and $target(a)$ the source and target entities of association a.

To integrate the knowledge that the concept *inspire* is a transitive concept into the EMMO model, we add the association $(l_{caligari} \overset{o_{inspire}}{\longrightarrow} l_{salem})$ to the EMMO "Dracula Movies V3" (see Fig. 7.6).

Integrating the Knowledge about Symmetric Concepts

In a similar way, we express *symmetric concepts*, i.e. relational concepts for which all associations can be traversed in both directions, i.e. source and target entities can be exchanged without changing the semantics of the concept, e.g.

$$(o_{similar} \in \Theta_{\text{SYM}}) \in \mathcal{A}^{\mathcal{O}}, \tag{7.10}$$

with $\Theta_{\text{SYM}} = \{o \in \mathcal{RC} \,|\, \forall a_1 \in I(o) \exists a_2 \in I(o)\,(source(a_1) = target(a_2) \wedge source(a_2) = target(a_1))\}$ describing the set of all symmetric ontology objects.

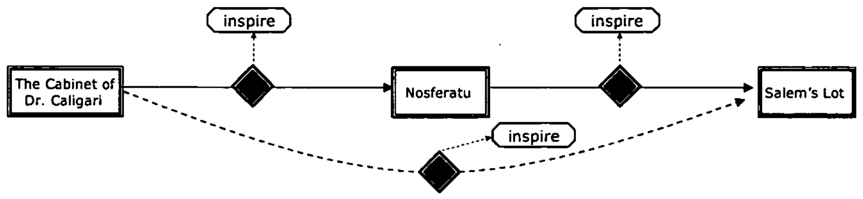


Figure 7.6: Integrating the knowledge that *inspire* is a transitive concept

To explicitly incorporate the knowledge about *similar audience* being a symmetric concept, we add the association $(l_{return} \overset{o_{similar}}{\longrightarrow} l_{salem})$ to the EMMO "Dracula Movies V3" (see Fig. 7.7).

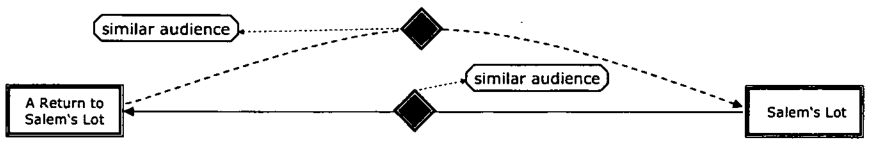


Figure 7.7: Integrating the knowledge that *similar audience* is a symmetric concept

Integrating the Knowledge about Inverse Concepts

Finally, we can also express that two relational concepts are *inverse* to each other, i.e. if an association is labeled with the inverse concept, then source and target entities have to be exchanged to keep the semantics intact, e.g.

$$((o_{retell}, o_{is-retold}) \in \Theta_{\text{INV}}) \in \mathcal{A}^{\mathcal{O}}, \tag{7.11}$$

with $\Theta_{\text{INV}} = \{(o_1, o_2) \in \mathcal{RC} \times \mathcal{RC} \,|\, \forall a_1 \in I(o_1) \exists a_2 \in I(o_2)(source(a_1) = target(a_2) \wedge source(a_2) = target(a_1))\}$ describing the set of all pairs of inverse ontology objects.

The fact that the concepts *retell* and *is-retold* are two inverse concepts can be expressed by adding the association $(l_{nosferatu} \overset{o_{is-retold}}{\longrightarrow} l_{dracula})$ to the EMMO "Dracula Movies V3" (see Fig. 7.8).

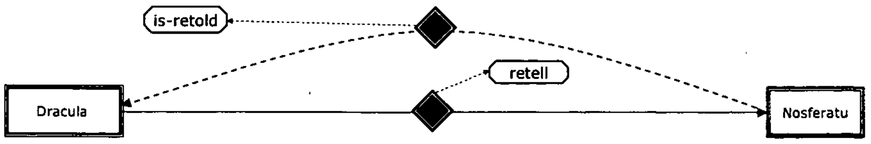


Figure 7.8: Integrating the knowledge that *retell* and *is-retold* are inverse concepts

Inflated versus Deflated EMMOs

To improve the access to EMMOs by integrating ontological knowledge into the EMMO model, an EMMO's graph structure has to be extended by additional associations according to the ontological axioms, i.e. the EMMO is *inflated*.

On the other hand, for enabling efficient authoring and exchanging of EMMOs within a distributed environment, EMMOs are required to be of compact size. To realize this, similar to the inflation of EMMOs, ontological knowledge can be used for *deflating* EMMOs, i.e. any redundant association within an EMMO which can be inferred from the ontology, is removed from an EMMO's nodes.

Thus, there are two different ways of representing EMMOs, i.e. a compact *minimum version* being optimal for exchanging EMMOs and a *maximum version* to provide efficient access to EMMOs. The decision which of the two versions is preferable for a given situation depends on several factors, such as the requirements of the application scenario, the depth and size of the ontology, or the size of an EMMO. Typical decision criteria are response time constraints for retrieval, storage limitations in mobile devices, or restricted bandwidth for network transmissions.

This leads to the problem how to exploit ontological knowledge for the retrieval of multimedia content in situations that would demand the use of an EMMO's minimum version. To solve this dilemma we have extended the query algebra EMMA. We provide means for integrating ontological knowledge within the processing of EMMA's navigational queries by adding alternative navigation paths (see Sect. 7.3). In this way, instead of *explicitly extending* an EMMO's graph structure, now, the graph structure is *implicitly extended* during query execution.

7.3 Ontology-enhanced Retrieval of Multimedia Content

EMMA's navigational operators enable the navigation along an EMMO's semantic graph structure, and thus provide the basis for ontology-based query refinement. The integration of ontology knowledge into EMMA query processing allows to refine user queries. Therefore, a user can pose imprecise queries, which are refined by drawing inferences over ontological knowledge. Instead of extending the EMMO graph structure (see Sect. 7.2), we now extend the navigational operators in EMMA by adding alternative navigation paths. In the following, we will illustrate how the knowledge captured by the ontology structure illustrated in Fig. 7.2 and described in more detail within Axioms 7.8–7.11 can be used for ontology-based query refinement.

Integrating the Knowledge about the Subconcept Hierarchy

To include ontological knowledge about the concept hierarchy we add all subconcepts as alternative branches to any ontology object in the regular path expression of a navigational EMMA query. For example, the hierarchical structure defined in Axiom 7.8 specifies that the concept o_{rework} has two subconcepts

o_{remake} and o_{retell}. Therefore, the query

$$JumpRight(e_{moviesV3}, l_{dracula}, o_{rework}) = \emptyset$$

can be expanded to search also for all subconcepts of *rework*, i.e. *retell* and *remake*:

$$JumpRight(e_{moviesV3}, l_{dracula}, o_{rework} \,|\, o_{retell} \,|\, o_{remake}) = \{l_{nosferatu}\}.$$

A user requesting all entities which were *reworked* by the novel "Dracula", now receives a useful answer, i.e. the logical media part "Nosferatu", because the novel *retells* the movie.

Integrating the Knowledge about Transitive Concepts

To integrate the knowledge about transitive concepts we add the operator "+" to any transitive ontology object in the regular path expression of a navigational EMMA query. For example, as the ontology object $o_{inspire}$ is defined as a transitive concept (see Axiom 7.9), we can expand the EMMA query

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

to the query

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

Therefore, if a user asks for all entities which were *inspired* by the movie "The Cabinet of Dr. Caligari", the user now receives not only the incomplete result consisting of one logical media part "Nosferatu", which is directly *inspired* by the movie "The Cabinet of Dr. Caligari", but also the logical media part "Salem's Lot" because it is indirectly *inspired.*

Integrating the Knowledge about Symmetric Concepts

By incorporating the knowledge about symmetric concepts, we extend any symmetric ontology object in the regular path expression of a navigational EMMA query by adding its inversion as alternative branch. For instance, as the ontology object $o_{similar}$ references a symmetric concept (see Axiom 7.10), the EMMA query

$$JumpRight(e_{moviesV3}, l_{return}, o_{similar}) = \emptyset$$

is expanded to

$$JumpRight(e_{moviesV3}, l_{return}, o_{similar}) \cup JumpLeft(e_{moviesV3}, l_{return}, o_{similar}) =$$
$$JumpRight(e_{moviesV3}, l_{return}, o_{similar} \,|\, o_{similar}-) = \{l_{salem}\}.$$

Thus, a user asking for all entities which address a *similar audience* as the movie "A Return to Salem's Lot" can now retrieve the information that the movie "Salem's Lot" was made for a *similar audience* although the corresponding association in the EMMO "Dracula Movies V3" points in the opposite direction.

Integrating the Knowledge about Inverse Concepts

The integration of the knowledge that an ontology object has an inverse concept is achieved through the extension of the ontology object in any regular path expression of a navigational EMMA query by adding the inversion of the inverse concept as alternative branch. For example, the knowledge about the two concepts o_{retell} and $o_{is-retold}$ being inverse to each other (see Axiom 7.11), is reflected by expanding the EMMA query

$$JumpRight(e_{moviesV3}, l_{nosferatu}, o_{is-retold}) = \emptyset$$

to the query:

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

As a result, if a user wants to know about any entity that *is-retold* by the movie "Nosferatu" we can now provide the satisfactory answer "Dracula", because the EMMO "Dracula Movies" specifies that this novel *retells* the movie.

7.4 Representation of Ontology Structures

There exist different ways of representing ontology structures, which, although having the same expressiveness, are designed for different purposes. In the following, we will show three different ways of representing the ontology structure illustrated in Fig. 7.2 and specified in more detail by Axioms 7.1–7.11:

1. the *graphical representation* enhances human readability,
2. the *OWL representation* addresses the standardization efforts in the context of the Semantic Web initiative [BHL01],
3. the *EMMO representation* of the ontology structure enables the seamless integration of ontological knowledge into the EMMO model.

Figure 7.9 shows the graphical representation of the ontology structure from Fig. 7.2. The relational concept *inspire* is marked as transitive, the relational concept *similar audience* as symmetric, and the relational concepts *retell* and *is-retold* as being inverse to each other. Additionally, the object concept *Text* is specified as domain concept for the relational concept *retell*.

Since DAML+OIL does not provide modeling constructs for symmetric properties, it is not adequate as representation language for ontology structures. Therefore, we used OWL DL, which specifies all the modeling constructs used within an ontology structure, i.e. constructs for expressing transitive, symmetric, and inverse concepts. We used Protege-2000 [Sta04] as authoring tool for creating the domain ontology, and imported the resulting OWL description into the EMMO environment. Figure 7.10 shows the OWL representation for the ontology in Fig. 7.9.

However, by representing the ontology in a standard format, such as OWL DL, more complex inferences drawn from the ontology knowledge cannot be integrated into EMMA queries. Therefore, we plan to develop our own ontology description language compatible with the EMMO model allowing for sophisticated reasoning on EMMOs. Figure 7.11 shows the EMMO representation

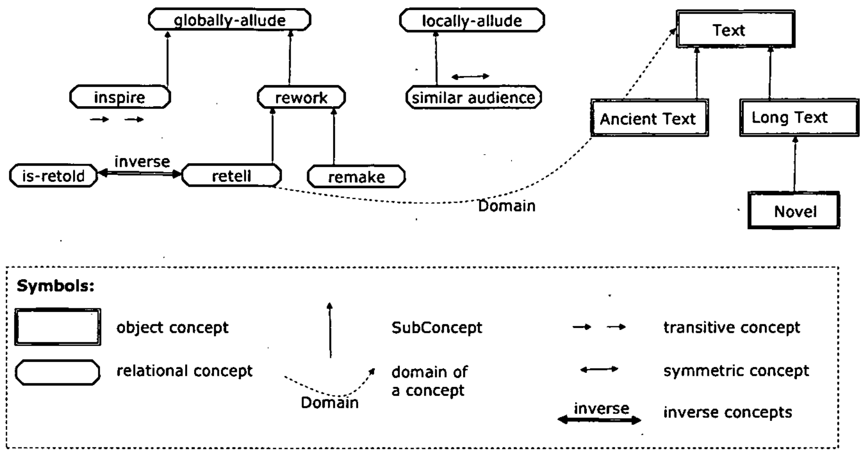


Figure 7.9: Graphical representation of the Ontology of Intertextual Studies

```
<rdf:RDF
     xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#" >
    <rdf:Class rdf:ID="Text"/>
    <rdf:Class rdf:ID="AncientText">
        <rdfs:subClassOf rdf:resource="#Text"/></rdf:Class>
    <rdf:Class rdf:ID="LongText">
        <rdfs:subClassOf rdf:resource="#Text"/></rdf:Class>
    <rdf:Class rdf:ID="Novel">
        <rdfs:subClassOf rdf:resource="#LongText"/></rdf:Class>
    <rdf:Property rdf:ID="globally-allude"/>
    <rdf:Property rdf:ID="rework">
        <rdfs:subPropertyOf rdf:resource="#globally-allude"/></rdf:Property>
    <owl:TransitiveProperty rdf:ID="inspire">
        <rdfs:subPropertyOf rdf:resource="#globally-allude"/></owl:TransitiveProperty>
    <rdf:Property rdf:ID="remake">
        <rdfs:subPropertyOf rdf:resource="#rework"/> </rdf:Property>
    <rdf:Property rdf:ID="retell">
        <rdfs:subPropertyOf rdf:resource="#rework"/>
        <rdfs:domain rdf:resource="#Text"/></rdf:Property>
    <rdf:Property rdf:ID="is-retold">
        <owl:inverseOf rdf:resource="#retell"/></rdf:Property>
    <rdf:Property rdf:ID="locally-allude"/>
    <owl:SymmetricProperty rdf:ID="similar">
        <rdfs:subPropertyOf rdf:resource="#locally-allude"/></owl:SymmetricProperty>
</rdf:RDF>
```

Figure 7.10: OWL representation of the Ontology of Intertextual Studies

of the ontology structure, i.e. how the ontology structure can be represented within the EMMO model. It is important to mention that the EMMO "Ontology" uses some "predefined" *meta ontology objects* corresponding to classical ontology constructs, such as *subconcept*, *relational concept*, *inverse concept*, or *domain*, which are again used to classify other ontology objects. For instance, to indicate that the ontology object *inspire* is a transitive, relational concept, it is typed by the meta ontology objects *transitive concept* and *relational concept.* To express that there is a subconcept relationship between the ontology objects *Text* and *Ancient Text*, an association of type *subconcept* between those two

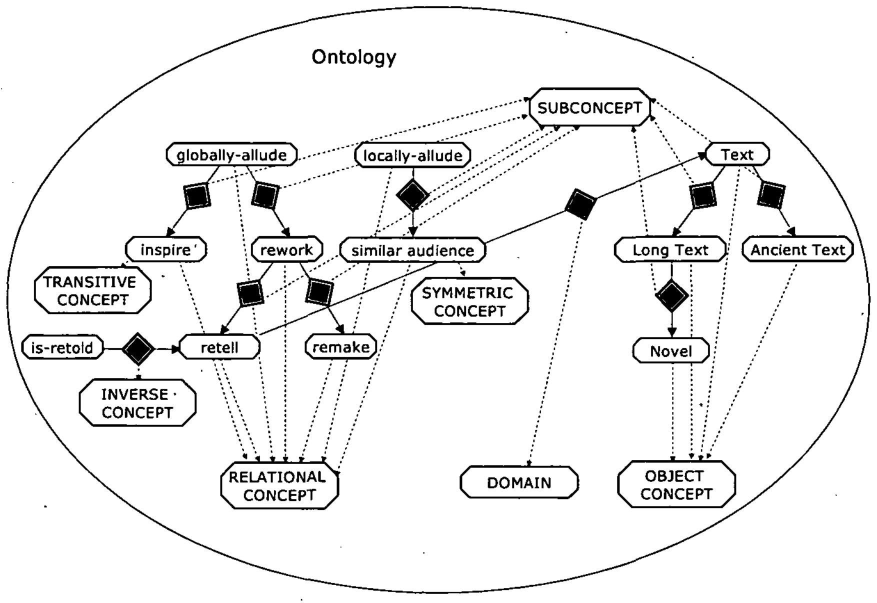


Figure 7.11: EMMO representation of the Ontology of Intertextual Studies ($e_{ontology}$)

ontology objects is established. Finally, to describe that any *retell* association only allows entities of type *Text* as its source entity, an association of type *domain* pointing from the ontology object *retell* to the ontology object *Text* is contained within EMMO "Ontology".

The contribution of an ontology description language is to define the semantics of those and many more meta ontology objects. The representation of the ontology structure within the EMMO model bears two major advantages. First, instead of having to rely on Protege-2000 as external ontology authoring tool, we can now use the EMMO authoring environment also for the development of the domain ontology. Second, EMMA operators can now be used to draw inferences from the ontological knowledge, and thus, the seamless integration of ontological knowledge within the authoring, processing, and querying of EMMOs can be realized. For example, the operator *IsType*, which specifies a set of ontology objects as second input parameter, enables the integration of supertype/subtype relationships within queries. The ontological knowledge about a subtype relationship, e.g. the subtype relationship between the ontology objects "Novel" and "Text", can be reflected within the query expression

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

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

$$IsType(l_{dracula}, JumpLeft(e_{ontology}, o_{novel}, o_{subconcept}*)) = true.$$

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

7.5 Summary

Both, EMMOs and the query algebra EMMA, provide a sound basis for the integration of ontology knowledge into multimedia knowledge management. In this chapter we have illustrated how ontological knowledge can be used for checking integrity constraints within the design and authoring process of EMMOs, how ontological knowledge can be used to inflate and deflate the knowledge structures described by EMMOs, and, finally, how ontological knowledge can be used for refining EMMA query expressions. Thus, EMMOs and EMMA provide a basis for the distributed and collaborative ontology-enhanced authoring, management, and retrieval of EMMOs. In the following chapter we show how we have realized the implementation of the EMMO container infrastructure and the EMMA query processing architecture.

Chapter 8

Implementation and Evaluation

EMMOs and EMMA provide a basis for the distributed and collaborative authoring, management, and retrieval of semantically enhanced multimedia content. To support this functionality, the implementation of the EMMO model and the EMMA algebra had to fulfil several requirements. In the following two sections, we will first introduce the implementation of the EMMO model, i.e. the *EMMO container infrastructure* supporting platform independency and scalability, and providing export and import facilities, and tools for displaying and authoring EMMOs. Then, we describe how we have designed and implemented the *EMMA query processing architecture*, and, finally, we discuss some query evaluation results.

8.1 EMMO Container Infrastructure

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

- The users of EMMO containers are manifold, i.e. ranging from individual users running a home PC to multimedia content publishers. In other words, the systems running the EMMO containers are very heterogeneous servers with different sizes, operating systems, capabilities, and requirements. Therefore, the implementation of the EMMO container infrastructure needs to be *platform independent* and *scalable*.

- As EMMOs are intended to enable the sharing and collaborative authoring of multimedia content, EMMOs must be *transferable* between the different EMMO containers. Therefore, *export* and *import* facilities for EMMO containers, reflecting an EMMO's content, i.e. its three aspects and versioning information, are required.

As already mentioned, we have established EMMO containers that provide the basis for the management and persistent storage of EMMOs. An EMMO container bundles an arbitrary number of EMMOs and is supplied with an application programming interface that permits external applications the access, manipulation, traversal, and retrieval of its contained EMMOs. Compared to the query facilities provided by the EMMA query algebra, the application programming interface is very limited in its expressivity, i.e. its main purpose is the management and handling of the contained EMMOs.

8.1.1 Platform Independency and Scalability

For supporting platform independency and scalability, we have implemented the EMMO containers in *Java* and used the object-oriented DBMS *ObjectStore* for their persistent storage. By using Java we achieve platform independency. The decision for using ObjectStore for the persistent storage of EMMOs was motivated by two factors. First, ObjectStore enables the scalability of EMMO containers, i.e. besides a full-fledged database server implementation suitable for larger content providers, ObjectStore also provides a code-compatible file-based in-process variant *PSEPro* that better suits the limited capabilities and needs of home users. Second, ObjectStore is an object-oriented DBMS, which are in general well suited for handling complex graph structures as specified by the EMMO model. If only the scalability of EMMOs had been of importance, the use of a similarly scaleable relational DBMS would have been an alternative. However, due to the graph structures inherent to the EMMO model, we favored the object-oriented DBMS ObjectStore.

8.1.2 Exporting EMMOs

For enabling adequate export and import facilities, an EMMO container must be able to completely export EMMOs into bundles encompassing the media, semantic, and functional aspect, as well as the versioning information. As EMMOs are capable of describing quite complex knowledge structures, different *export modes* reflecting only selected parts of an EMMO's content, should be provided. For avoiding the storage of duplicates when importing EMMOs, we have developed some basic decisions rules.

EMMO containers export their EMMOs to a bundle structure. An *export bundle* is basically a ZIP archive that captures an EMMO's three aspects, versioning information, and export options. Figure 8.2 illustrates the export bundle of EMMO "Dracula Studies" (see Example 5 and Fig. 8.1).

Example 5

$$\begin{aligned} e_{studies} =(&\text{"}e3811\text{"}, \text{"Dracula Studies"}, \text{"emm"}, \epsilon, \epsilon, \emptyset, \emptyset, \emptyset, \\ &\{l_{vampyre}, l_{dracula}, l_{nosferatu}, o_{miller}, e_{moviesV1}, a_{va\rightarrow dr}, a_{dr\rightarrow no}, a_{mi\rightarrow(va\rightarrow dr)}, a_{dr\rightarrow moV1}\}, \\ &\emptyset, \emptyset, \emptyset, \{(o_{render}, f_{render})\}). \end{aligned}$$

The name of the export bundle reflects the OID of the bundled EMMO "Drac-

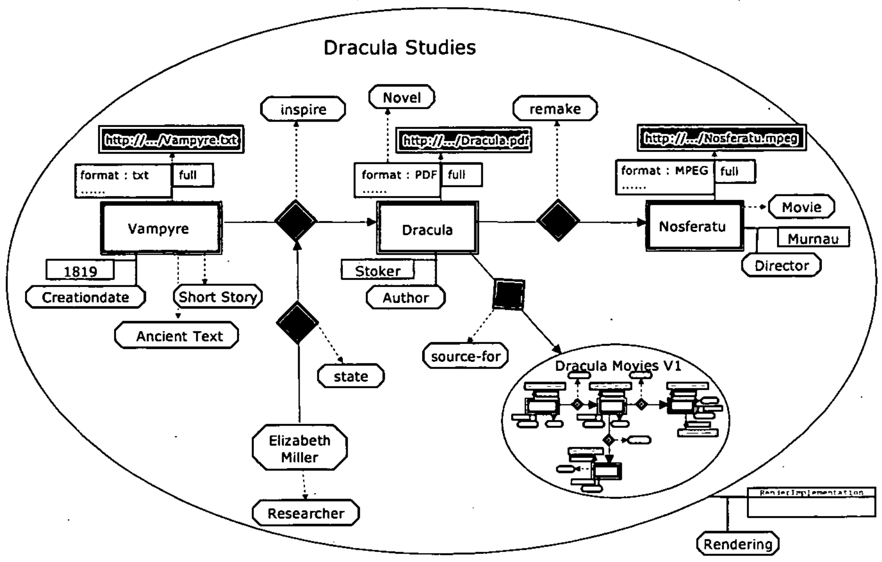


Figure 8.1: EMMO "Dracula Studies"($e_{studies}$)

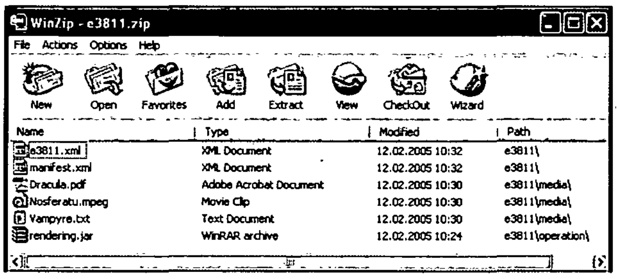
ula Studies". As EMMOs can describe quite complex structures, it is important for the users to specify the parts of the EMMO they want to export. To realize this, the export bundle encompasses a *manifest file* (see Fig. 8.3) indicating the options for the export. There are four different *export options* to specify:

- "includeMedia" indicates whether the associated media files referenced by the logical media parts within the EMMO should be packaged into the bundle,
- "versioning" indicates whether the versions of the entities belonging to the EMMO should be included,
- "recursive" indicates whether all recursively contained entities should be part of the bundle,
- "operations" indicates whether the EMMO's operations should be included within the bundle archive.

The media aspect of the EMMO is reflected by the folder "media". When specifying the "includeMedia" attribute as true, the folder "media" encompasses all associated media files contained within EMMO "Dracula Studies", i.e. the text documents "Vampyre.txt" and "Dracula.pdf", and the video "Nosferatu.mpeg".

The functional aspect is reflected by the folder "operation". As the attribute "operations" is specified as true, the folder encompasses the jar file of the attached rendering operation.

The semantic aspect and the versioning information is described within the *transfer file* whose name is again derived from the OID of the bundled EMMO. In accordance with the export options specified in the manifest file, the transfer


WinZip - e3811.zip
File Actions Options Help
New Open Favorites Add Extract View CheckOut Wizard

Name	Type	Modified	Path
e3811.xml	XML Document	12.02.2005 10:32	e3811\
manifest.xml	XML Document	12.02.2005 10:32	e3811\
Dracula.pdf	Adobe Acrobat Document	12.02.2005 10:30	e3811\media\
Nosferatu.mpeg	Movie Clip	12.02.2005 10:30	e3811\media\
Vampyre.txt	Text Document	12.02.2005 10:30	e3811\media\
rendering.jar	WinRAR archive	12.02.2005 10:24	e3811\operation\

Figure 8.2: Export bundle for EMMO "Dracula Studies"

```
<?xml version ="1.0" encoding ="iso-8859-1"?>
<Manifest>
    <Emmo ID="e3811">
          <ModifiedDate>........
          .............

    </Emmo>
    <ExportOptions includeMedia="true" operations="true"
          versioning= "false" recursive="false"/>
</Manifest>
```

Figure 8.3: Manifest file of the export bundle for EMMO "Dracula Studies"

file describes all of the EMMO's contained entities along with their oids, names and types, all associations with their source and target entities, and all preceding and succeeding versions.

To fulfil the requirements of different application scenarios the export functionality distinguishes different export variants. The chosen export variants are recorded in the manifest file and specify whether the EMMO is exported with or without media files, with or without recursively contained entities, with or without operations, and with or without versions. For the implementation of the different export variants, EMMO containers distinguish three different *export modes* that are additionally reflected within the transfer file. Figure 8.4 shows the transfer file with the export modes derived from the values for the export options in the manifest file in Fig. 8.3.

The export modes specify how entities are placed in the bundle:

- The *strong* mode represents the default export mode for entities and specifies that the bundle covers the complete information about an entity, i.e. an entity's OID, name, types, attributes, direct succeeding and preceding versions, media profiles, operations, and encapsulated entities. For example, the logical media part "Dracula" is transferred in strong mode.

- The *hollow* mode is only used for EMMOs and indicates that the bundle includes all information about the EMMO except the entities it contains, i.e. the entities contained within the EMMO are excluded from the export. For instance, as within the manifest file it is expressed that the value for the export option "recursive" is false, EMMO "Dracula Movies V1" contained within EMMO "Dracula Studies" is exported in hollow mode and all the entities in EMMO "Dracula Movies V1" are not included in the bundle.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<emmo xmlns="http://www.cultos.org/emmos"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  ......>
      <componenents>
              <entities>
                      <entity xsi:type="LogicalMediaPart" mode="strong">
                                 <oid>13892</oid>
                                 <name>Dracula</name>
                                 <creationDate>Montag,  14. Februar 2005 09:45 Uhr GMT+02:00</creationDate>
                                 <modifiedDate>Montag,  14. Februar 2005 09:45 Uhr GMT+02:00</modifiedDate>
                      </entity>
                      <entity xsi:type="EMMO" mode="hollow">
                                 <oid>e7390</oid>
                                 <name>Dracula Movies V1</name>
                                 <creationDate>Freitag,  04. Februar 2005 12:53 Uhr GMT+02:00</creationDate>
                                 <modifiedDate>Freitag,  04. Februar 2005 12:53 Uhr GMT+02:00</modifiedDate>
                      </entity>

                      ........

                      <entity xsi:type="OntologyObject" mode="strong">
                                 <oid>o4302</oid>
                                 <name>Movie</name>
                                 <creationDate>Freitag,  20. Februar 2004 12:31 Uhr GMT+02:00</creationDate>
                                 <modifiedDate>Freitag,  20. Februar 2004 12:31 Uhr GMT+02:00</modifiedDate>
                      </entity>

                      ........

                      <entity xsi:type="EMMO" mode="weak">
                                 <oid>e7921</oid>
                                 <name>Dracula Movies </name>
                                 <creationDate>Montag,  24. Januar 2005 10:25 Uhr GMT+02:00</creationDate>
                                 <modifiedDate>Montag,  24. Januar 2005 10:25 Uhr GMT+02:00</modifiedDate>
                      </entity>
                      <entity xsi:type="EMMO" mode="weak">
                                 <oid>e3225</oid>
                                 <name>Dracula Movies V3</name>
                                 <creationDate>Freitag,  11. Februar 2005 08:15 Uhr GMT+02:00</creationDate>
                                 <modifiedDate>Freitag,  11. Februar 2005 08:15 Uhr GMT+02:00</modifiedDate>
                      </entity>

                      ........

             </entities>
            <mediaProfiles>
                         ........
            </mediaProfiles>
      </components>
             ......

      <links>
              <types>
                     <typeLink entity="13892" type="o4302"/>
                     ........

              </types>
               <attributeValues>
                       ........
               </attributeValues>
              <associations>
                     <assoLink association="a1289" sourceEntity="13892" targetEntity="e7390"/>
                     ........

              </associations>
               <connectors>
                        .........
               </connectors>
               <predVersions/>
               <succVersions/>
               <encapsulation/>
               <operations>
                        .........
               </operations>
      </links>
</emmo>
```

Figure 8.4: Extract of the transfer file `e3811.xml` of the export bundle for EMMO "Dracula Studies"

- The *weak* mode indicates that the bundle only contains the basic information about an entity, such as its OID and name; all the remaining characteristics of an entity, such as its types, attributes, versions, media profiles, operations, and encapsulated entities, are excluded from the export. When EMMOs are chosen to be exported without the versioning information, i.e. the manifest file specifies the export option "versioning" as false, any immediate preceding and succeeding versions of the exported entities are placed within the bundle in weak mode; all indirect predecessors and successors are left out. For example, as the manifest file specifies "versioning" as false, EMMO "Dracula Movies V1"'s direct successor EMMO "Dracula Movies V3" and direct predecessor EMMO "Dracula Movies" are exported in weak mode.

Receiving a request for exporting an EMMO with specified export options, the export tool first collects all required files into the corresponding folder structure, packs the folder into a zip file, and subsequently deletes the directory structure from the local filesystem; it just keeps the compressed file that can now be used for export and exchange.

8.1.3 Importing EMMOs

When importing EMMOs, one has to consider that EMMOs can be authored in a distributed and collaborative way, i.e. EMMOs and entities might have been concurrently modified without creating new entities. Thus, when importing an EMMO bundle into an existing database, the bundle might contain entities carrying the same OIDs as the corresponding entities in the local database. To avoid duplicates, one needs to define some precise decision rules determining whether the local or the imported entity prevails.

The decision rules are based on two parameter values, i.e. an entity w's export mode $mode(w)$ and modified date $date(w)$. For the entities in the local database, these two values are stored within the set of features. The basic strategy of importing EMMO containers is that the greater export mode (stronger $\geq$ hollow $\geq$ weak), the more recent modified date, and the local copy prevail. Additional data contained in an imported entity, such as recursively contained entities or included media data, is added to a local entity by simultaneously raising the export mode. If there are two entities w_{local} and w_{bundle} with the same OID, with w_{local} stored in the local database and w_{bundle} contained within the imported bundle structure, then we distinguish four different cases:

1. If $mode(w_{local}) \geq mode(w_{bundle})$ and $date(w_{local}) \geq date(w_{bundle})$, then w_{local} prevails and w_{bundle} is ignored.

2. If $mode(w_{local}) < mode(w_{bundle})$ and $date(w_{local}) < date(w_{bundle})$, then w_{bundle} prevails and w_{local} is ignored.

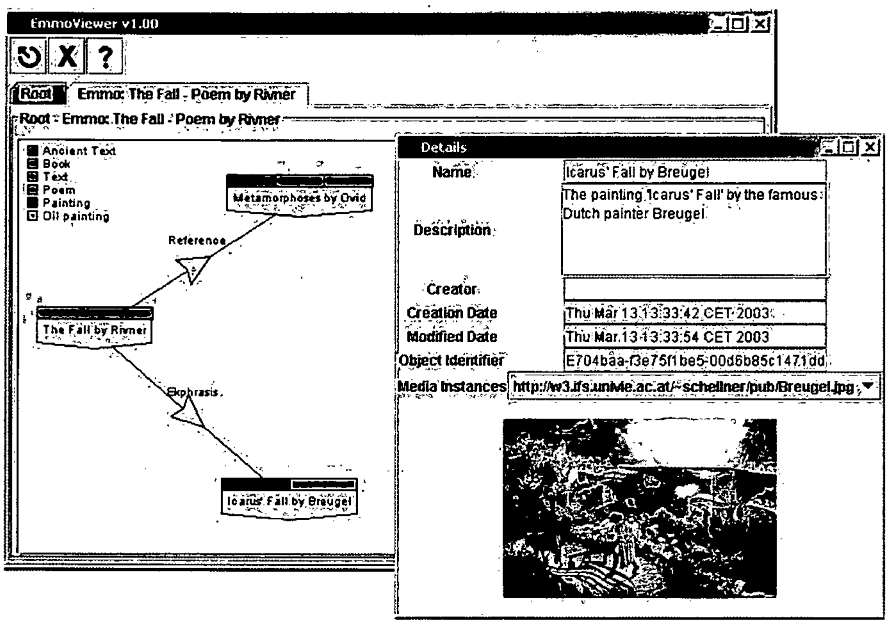
3. If $mode(w_{local}) < mode(w_{bundle})$ and $date(w_{local}) \geq date(w_{bundle})$, then any additional content captured by w_{bundle} is added to w_{local}, and subsequently the export mode of w_{local} is changed to the export mode of w_{bundle}.

4. If $mode(w_{local}) \geq mode(w_{bundle})$ and $date(w_{local}) < date(w_{bundle})$, then w_{local} prevails and w_{bundle} is ignored.

Reflecting the export mode and modified date when transferring EMMOs guarantees that EMMOs are maintained in a consistent manner and capture maximum and most recent information about their content.

8.1.4 Displaying and Authoring of EMMOs

For displaying EMMOs to the user, the implementation is supplied with a simple graphical EMMO Viewer. Figure 8.5 shows a screenshot of the EMMO Viewer displaying EMMO "The Fall – Poem by Rivner" that was authored within the context of the CULTOS project (see Sect. 9.1).



Name	Icarus' Fall by Breugel
Description	The painting 'Icarus' Fall' by the famous Dutch painter Breugel
Creator	
Creation Date	Thu Mar 13 13:33:42 CET 2003
Modified Date	Thu Mar 13 13:33:54 CET 2003
Object Identifier	E704baa-f3e75f1be5-00d6b85c1471dd
Media Instances	http://w3.ifs.univie.ac.at/~schellner/pub/Breugel.jpg

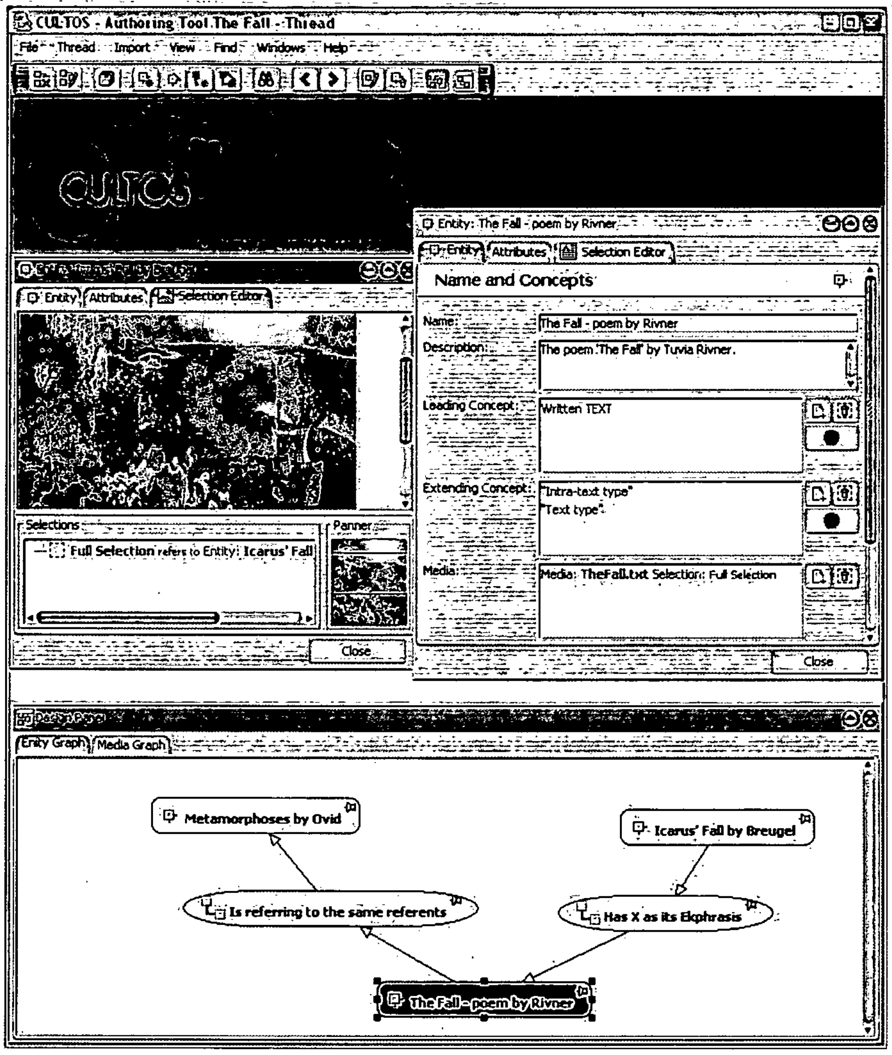
Figure 8.5: Screenshot of the EMMO Viewer

Logical media parts are represented as pentagons by employing color codes to their top section to denote their respective types, and associations are illustrated as labeled edges. The dialog window at the right hand side of Fig. 8.5 provides further details on the logical media part, e.g. the media instance and a human readable description is displayed for the painting "Icarus' Fall by Breugel".

For the construction of EMMOs, the *EMMO Authoring Environment* – a tool which was developed by one of our partners within the CULTOS project (see Sect. 9.1) – can be used. The EMMO Authoring Environment consists of a *Media Import Tool* and an *EMMO Authoring Tool.* The Media Import Tool provides means for the import and the definition of segments from media objects, and the construction of logical media parts by associating them with ontology objects, attributes, and connector information. The EMMO Authoring Tool enables the construction of EMMOs and facilitates the rendering of an EMMO's content in HTML format.

Figure 8.6 shows a screenshot of the EMMO Authoring Tool displaying again the EMMO "The Fall – Poem by Rivner".

In future work, we aim to integrate all those tools into one single *EMMO Development Environment* providing a profound basis for the integration of ontological knowledge within the EMMO management as described in Chapter 7.


CULTOS - Authoring Tool The Fall - Thread
File Thread Import View Find Windows Help
Entity Attributes Selection Editor
Selections
Full Selection refers to Entity: Icarus' Fall
Panner
Close
Entity: The Fall - poem by Rivner
Entity Attributes Selection Editor
Name and Concepts
Close
Entity Graph Media Graph
Metamorphoses by Ovid
Icarus' Fall by Breugel
Is referring to the same referents
Has X as its Ekphrasis
The Fall - poem by Rivner

Name: The Fall - poem by Rivner
Description: The poem 'The Fall' by Tuvia Rivner.
Leading Concept: Written TEXT
Extending Concept: "Intra-text type"
"Text type"
Media: Media: TheFall.txt Selection: Full Selection

Figure 8.6: Screenshot of the EMMO Authoring Tool

8.2 EMMA Query Processing

Because of an EMMO's capability of encapsulating other entities and EMMOs, its versioning mechanism, and its graph structure, the EMMA query algebra, which enables the access to the information stored within EMMOs, can produce quite complex query operations. Depending on the structure and size of the EMMO containers, as well as the size of the integrated ontology, the query execution time might become too long for many practical applications. Therefore, it is important to limit or, if possible, to reduce query execution time. This can be realized by query optimization. Thus, for fulfilling this important requirement, the EMMA query processing environment has to provide a basis for query optimization through query rewriting.

We have implemented the *EMMA query processing architecture* with query optimization in mind, however, the realization of a query optimizer is subject of future work. The EMMA query processing architecture, which is depicted in Fig. 8.7, is based on the implementation of the EMMO container infrastructure described in the previous section. Its focus is the extraction and navigation of information stored within the EMMO containers. The EMMA query processing architecture takes syntactically well-defined query expressions as input. The processing of the query expressions reflects the definition of the EMMA query operators and produces a set of EMMO objects in a pre-defined output format.

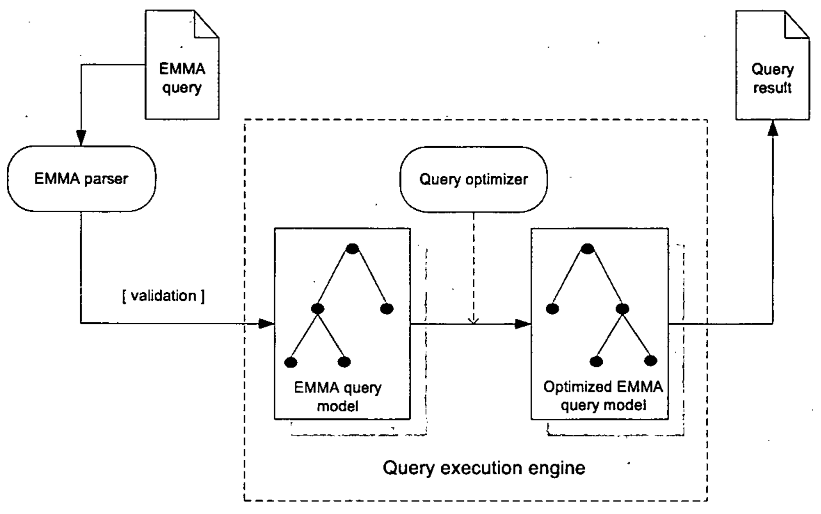


Figure 8.7: The EMMA query processing architecture

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

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

The design of the EMMA query processing architecture was inspired by the work of [Gra93] who proposes five important steps for the efficient and fast retrieval of query results: parsing and evaluation, query resolution, optimization, plan compilation, and query evaluation. In the following, we illustrate how to realize the five steps within the EMMA query processing architecture:

1. For each EMMA query expression that is sent to the query processor, the *EMMA parser* checks whether it is syntactically well-formed.

2. The EMMA parser *resolves* the logical *query expression* into physical, system-specific operators. Each modular operator is specified by an algorithmic implementation, and complex queries, i.e. nested sets of operators, are resolved by building up the EMMA query model.

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

4. A *query plan* assures maximum efficiency by defining the chronological order for evaluating single operators in complex query trees. So far, we have implemented a rather simple query plan. Using evaluation results, we will refine the query plan in future work.

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

As already mentioned, the current implementation covers the building and evaluation of the EMMA query model as part of the *query execution engine*. Currently, we are working on query optimization techniques that are tailored to the physical EMMA algebra. As starting point for the design of the query optimizer, we have done some evaluation experiments illustrated in the following subsection. We expect our future implementation to evaluate query requests on an *optimized EMMA query model*. Moreover, as in our current implementation each algebraic operator is implemented in terms of an ObjectStore function, we plan to integrate query optimization strategies as outlined in [L$^+$97], i.e. we want to translate the EMMA queries into an object algebra, such as [O$^+$95], for which query optimization techniques are readily available.

8.2.1 Evaluation

In order to evaluate the efficiency of the EMMA query execution engine and the potential for query optimization, we carried out some first experiments to compare the performance of a selected number of representative EMMA operators.

Table 8.1: Response times for selected operations (in ms)

	Query performance evaluation								
	DB 1 (100 EMMOs)			DB 2 (300 EMMOs)			DB 3 (500 EMMOs)		
	flat	normal	deep	flat	normal	deep	flat	normal	deep
AllEncEnt(e_{root})	8.67	17.44	14.11	15.56	36.33	74.56	138.86	118.29	160.71
AllEncEnt(e_x)	15.4	14.2	18.6	61.1	31.2	45.3	54.7	79.7	104.7
AllSuccessors(e_x)	14.2	3.2	8	3.2	11.2	6.4	9.6	11.2	6.4
JumpRight(e_x, l_y, o_z+)	8	11.2	6.4	11.2	9.6	12.8	11.2	16	11.2

All experiments were run on a PC with Intel Pentium 4 processor with 2.6 Ghz and 512 MB DDR memory running Windows XP SP1 and the Java Development Kit 1.4.2.

We have implemented an evaluation environment that allows the creation of EMMO data sets with definable size and structure. In all our experiments the EMMO e_{root} encapsulates all the other generated EMMOs e_x. Each EMMO has between 0 and 50 successors and can contain up to 100 entities. In order to create navigation paths of arbitrary length, we connect the entities with associations to establish a graph structure.

The values displayed in Table 8.1 represent the mean of the results obtained after performing the experiments ten times. The columns "flat", "normal", and "deep" indicate the maximum level of nesting: in a *flat* hierarchy the maximum level of nesting is 1, in a *normal* hierarchy 5, and in a *deep* hierarchy 10.

Unsurprisingly, the response time of all operators that access the hierarchical structure of EMMOs increases with the number of EMMOs contained in the database. Regarding the influence of the maximum level of nesting on the response time of queries that access the hierarchical structure, the evaluation results were inconclusive. This might be caused by a trade-off between the additional workload and efficiency gains during the recursive processing of the queries. For the two other query types the response time remains almost constant. Further tests will be required to obtain a more detailed analysis and understanding of the various effects on query performance. One important point we learned from the experiments is that the development of a query optimization strategy cannot be separated from the semantics determined by a specific application scenario. In the experiment above, we artificially constructed databases of different size and structure. However, as there are too many factors influencing the query performance, such as the total number of entities, the percentage of EMMOs, level of versioning, level of recursion, and the size and structure of the ontology, we could not identify characteristic classes of databases showing similar query performance. Consequently, we will carry out further experiments based on real-world EMMO data derived from the application scenarios described in the following chapter. By using application-specific data as starting point for the EMMA query optimizer, the results of restructuring the EMMA query model enable the reformulation of hierarchical queries through rewriting that is customized to the particular application scenario.

8.3 Summary

The platform independent and scalable EMMO container infrastructure provides adequate export and import facilities and is supplied with the EMMO Viewer for displaying EMMOs and the EMMO Authoring Tool for the authoring of EMMOs. In future work, we plan to integrate all these tools into one EMMO Development Environment providing a profound basis for the integration of ontological knowledge within the management of EMMOs. The EMMA query processing architecture provides a basis for query optimization. First evaluation tests based on artificially constructed databases showed inconclusive results. Thus, further evaluation tests in real-world applications to obtain a more detailed analysis and understanding of the various effects on query performance are required. In the following chapter, we introduce application scenarios that will produce the required data for future evaluation tests.

Chapter 9

Application Scenarios for the EMMO Model

After having introduced and described the EMMO approach for the semantic modeling of multimedia content and the algebra EMMA for ontology-enhanced querying of EMMOs, we want to illustrate in this chapter by means of three application scenarios how our approach may be practically applied and what benefits it offers. The first scenario is the development of a platform for the exchange of cultural knowledge (Sect. 9.1). The second scenario addresses the domain of eLearning (Sect. 9.2), and the third scenario introduces a multimedia task management system (Sect. 9.3). All three application scenarios have in common that they require an infrastructure providing means for the distributed and collaborative authoring of multimedia content.

9.1 Management of Cultural Knowledge

The EMMO model has been originally developed to fulfil the requirements of the EU-funded project CULTOS. CULTOS lasted from September 2001 until October 2003 and was carried out by 11 partners from different EU-countries, Israel, and Estonia. CULTOS addressed the need of researchers in the domain of intertextual studies for an integrated view on individual and culture-dependent perceptions of interrelationships between cultural artefacts, such as literature, artworks, movies, etc. For that purpose, we had to develop an Internet-based multimedia collaboration platform for authoring, managing, retrieving, and exchanging so-called *InterTextual Threads (ITTs)* [B^+02],[SWZK03] – knowledge structures that semantically interrelate and compare cultural artefacts. ITTs represent trails of associations between different "texts", where "text" can be anything ranging from paintings, advertisements, melodic patterns in songs, theater recordings, etc. The main aim of ITTs is to make the cultural memory of Europe more attractive and appealing to its citizens. For instance, the development of reverse chronological threads that start with well-known elements of the popular culture, such as movies, pop songs, cartoons, or novels, and subsequently connect those elements to artefacts originating from the traditional Western cultural canon, such as allusions or parodies, supports and improves the comprehensibility of the European cultural heritage. Moreover, ITTs render it possible for the community of intertextual studies to create and exchange pieces of a world-wide collection of semantically connected and multimedia-enriched artifacts incorporating the community's different cultural backgrounds – an important contribution to the preservation of European cultural heritage.

ITTs are basically graph structures that describe semantic relationships between pieces of literature, artworks, and other kinds of cultural artefacts. They can take a variety of forms, ranging from spiders over centipedes to associative maps. The example ITT depicted in Fig. 9.1 highlights several relationships of the poem "The Fall" by Tuvia Ribner to other works of art. It states that the poem makes reference to the 3rd book of Ovid's "Metamorphoses" and that the poem is an ekphrasis (an ekphrasis is defined as "literary representation of visual art") of the painting "Icarus' Fall" by the famous Flemish painter Breugel.

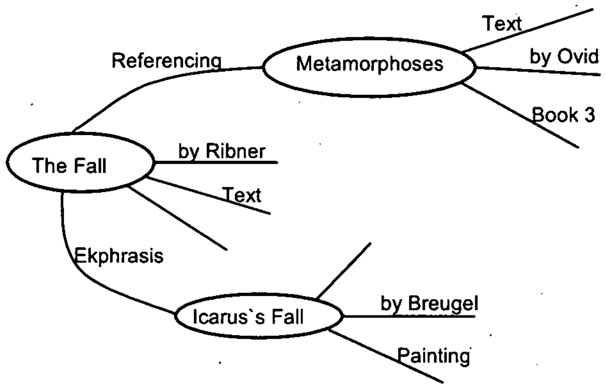


Figure 9.1: Simple InterTextual Thread

The graphical representation of the ITT bears strong resemblance to techniques from the domain of knowledge engineering like conceptual graphs and semantic nets, though it lacks their formal rigidity. However, the complexity of ITTs should not be underestimated. ITTs 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 affects the kind of relationships between pieces of literature he discovers and are of value to him. For instance, Fig. 9.2 illustrates two ITTs that manifest two different points of view on Ribner's poem by describing that the author of the ITT on the left hand side emphasizes the context of the poem to other works of art, whereas the author of the ITT on the right hand side stresses the relationship of the poem to the religious concept of the "Fall of Adam and Eve" in "Genesis". Furthermore, ITTs themselves can be relevant subjects of discourse and thus be contained as first-class artifacts within other ITTs, as, for instance, the two ITTs in Fig. 9.2 are described as opposed representations.

Reification of statements is also occurring frequently 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. Figure 9.2, for instance, expresses through reification that the statement

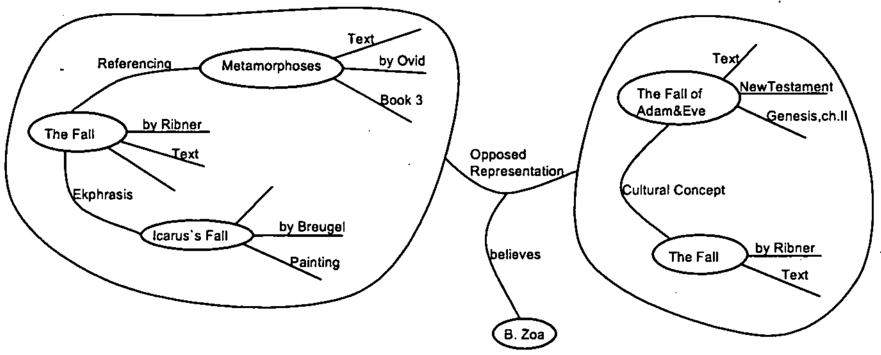


Figure 9.2: Complex InterTextual Thread

describing the two depicted ITTs as opposed representations is only the opinion of the researcher B. Zoa.

Given these characteristics of ITTs, we find that EMMOs are indeed very well suited for their representation within the Internet-based multimedia collaboration platform for intertextual studies:

- The *semantic aspect* of EMMOs offers sufficient expressiveness to capture ITTs. Figure 9.3 shows how the complex ITT of Fig. 9.2 could be represented using EMMOs. Due to the fact that associations as well as EMMOs themselves are entities, it is no problem to deal with reification of statements as well as with encapsulation of ITTs.

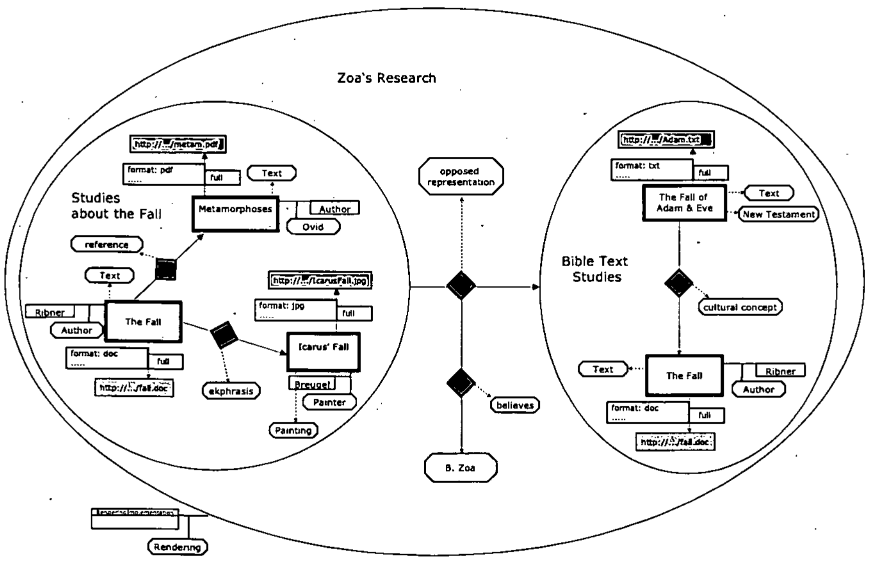


Figure 9.3: An ITT represented by an EMMO

- The *media aspect* of EMMOs allows to enrich ITTs that so far expressed interrelationships between cultural artefacts only on an abstract level with digital media about these artefacts, such as a JPEG image showing Breugel's painting Icarus' Fall, a Word document with the poem of Tuvia Ribner, etc. The ability to consume these media while browsing an ITT certainly enhances the comprehension of the ITT and the relationships described therein.
- The *functional aspect* of EMMOs permits to attach functionality to ITTs. For instance, an EMMO representing an ITT might know how to render itself as a centipede, associative map, or spider via SVG, or might offer rights clearing functionality for the media by which it is enhanced.
- By using *EMMA's query facilities*, the researcher can access all the information captured by EMMOs. For example, a researcher can first create a new EMMO by unifying all EMMOs representing ITTs that discuss Breugel's painting "The Fall", and subsequently navigate along the semantic graph structure of the newly created EMMO.
- By *integrating ontological knowledge* within EMMA query processing, imprecise queries can be refined by drawing inferences from ontological knowledge. For example, a researcher who is looking for all ancient texts serving as "cultural concept" for the poem "The Fall", should also receive the ancient text "The Fall of Adam and Eve", which is not directly classified as ancient text but certainly falls into this category as it is published in the New Testament.
- The *EMMO container infrastructure* provides a suitable foundation for the realization of the cultural knowledge platform. By enabling the persistent storage of EMMOs and providing interfaces that allow applications to traverse and manipulate the stored EMMOs as well as to invoke their operations, EMMO containers establish an ideal ground for the implementation of authoring, browsing, and querying applications for ITTs that had to be realized within the CULTOS project. Furthermore, their decentralized approach renders it possible to set up independent EMMO containers at the sites of different researchers, and by providing means for the import and export of EMMOs through bundling all their aspects and versioning information, the exchange of ITTs including their media data and offered functionality can be realized. This allows researchers to share and collaboratively work on ITTs in order to discover and establish new links between artworks as well as different personal and cultural viewpoints, thereby paving the way to novel insights into a subject. Moreover, regarding the integration of ontological knowledge, the EMMO container provides the basis for the inflation and deflation of EMMOs. For instance, by removing redundant associations from an EMMO, its original representation is transformed into a deflated, i.e. compact minimum version for efficient transfer.
- *Versioning* within the EMMO model supports this kind of collaboration, allowing to create different versions of an ITT independently and concurrently at different sites, to merge these versions, as well as to highlight differences between these versions. By versioning, it is also possible to

reflect the evolution of a researcher's viewpoint onto a subject that might change over time.

Within the CULTOS Project, groups of Cultural Studies specialists based in 11 different countries, formulated a commonly agreed ontology – the Ontology of Intertextual Studies [B+03b]. The ontology is divided in three parts:

- The *upper ontology* encompasses the concepts that are not specific to intertextual studies, such as "agent", "event", "object", and "place".
- The *middle ontology* consists of hundred major *concepts* and more than hundred specified *relationships*. The relationships are hierarchically structured and define all types of connections that can occur between "texts". As the knowledge model is intended to answer the needs of different approaches to intertextuality, it will be subject to evolution.
- The *lower ontology* establishes an intertextual taxonomy, i.e. it specifies properties of relationships, such as symmetric, transitive, or inverse relationships, relates relationships to other concepts or relationships by describing their domain and range values, and associates attribute values to relationships.

Concepts and relationships provide the semantics of the application domain and can be used for describing the interrelations between cultural artefacts.

By using the EMMO Authoring Tool (see Sect. 8.1.4), the researchers in intertextual studies produced 14 individually authored case studies of intertextual relations across various areas of European cultural production [B+03c]. The realization of the case studies as EMMOs served as a first practical evaluation to show the feasibility of the EMMO approach in a real-world setting.

9.2 eLearning

eLearning is currently a very broadly discussed topic in academia as well as in industry [Sto04]. In a globalized world with increasingly faster innovation cycles, people are confronted with the challenge of lifelong learning. The basic hope of eLearning is to be able to tackle this challenge by providing individuals with access to multimedia learning and training material via the Internet, independent from time and location and as much as possible personalized to the individual's context, such as prior education and knowledge, information needs, language, preferred learning style, etc.

To fulfil this vision, it is certainly not sufficient to simply put pre-canned coarse-grained learning material such as the slides, notes, or video recordings of a whole lecture onto a Web server. Instead, it is necessary to modularize learning material into smaller, reusable units that can be flexibly assembled into highly individual presentations that satisfy a user's current information needs and context. As a prerequisite to this, it is necessary to semantically describe the learning units and their interrelationships by means of a suitable eLearning domain ontology so that these units can be matched with a user's profile, preferences, and information needs.

The situation is similar for the realization of the mobile learning approach, i.e. the representation of learning content by means of mobile devices, to meet the requirements of learners studying on their own. By modularizing the learning material into small units covering different formats, levels of details, and quality ranges, device-aware presentations can be generated.

Systems that follow such a semantic modeling approach for eLearning material already exist, e.g. the eLearning Portal introduced in [SSS01] or MultiBook [S^+99]. These systems define ontologies which allow to apply content, context, and structure relations for the description of learning units: *content relations* establish a classification scheme for concepts and notions of the selected domain – e.g. "dealsWith", "partof"– *context relations* put learning units into their pedagogical context – e.g. "exampleOf", "discussionOf"– and *structure relations* describe structural relations between learning units – e.g. "basedOn", "next".

The implementations of these systems typically follow centralized server architectures. Learning units can be uploaded onto a server and then described using the ontology defined by the respective system. The server, among others, offers functionality for the search and retrieval of suitable learning material using these descriptions as well as personalized rendering functionality, which automatically selects learning material and composes it to an individual multimedia presentation that suits the context of the user, including information need, educational level, and preferred learning style.

While systems, like the eLearning Portal or MultiBook, that follow a semantic approach to the modeling of learning material definitely constitute important steps towards modern context-aware eLearning environments, they suffer from their centralistic architectures. On the one hand, they have not been designed with the sharing and exchange of content in mind. Instead, they expect all learning material to be uploaded on central servers. However, learning material is inherently distributed on the Internet and authors of such material, even though they are often very well willing to share their material with others, might be reluctant to publish it by uploading it onto a central server not under their control. This insight gave rise to recent developments of distributed, peer-to-peer infrastructures for the sharing of learning materials such as Edutella [N^+02]. On the other hand, these centralistic systems ignore the fact that people not just want to share material in the sense of publishing but also to collaboratively work on it. Moreover, they offer no support for collaborative authoring, such as versioning, etc. However, today, it is common practice, for instance, that professors in charge of preparing a new lecture typically borrow slides from their colleagues and adapt them to their purposes at the price of handing over their adaptations back to the lender so that he might profit from some of these changes as well.

We believe that the EMMO model and container infrastructure offers the prospect of realizing appealing distributed and collaborative eLearning environments that follow the semantic modeling approach but avoid the problems of the centralized systems. Figure 9.4 gives a sketch of an example EMMO which – roughly following the organization of the learning material on the HyperPhysics system Website [Ron03] – models some of the learning material from the domain of physics, in particular material covering mechanics formulas. The material consists of two EMMOs which represent learning units about the Law of Work and Newton's 2nd Law.

Using the example EMMO, we can demonstrate the benefits provided by the EMMO infrastructure:

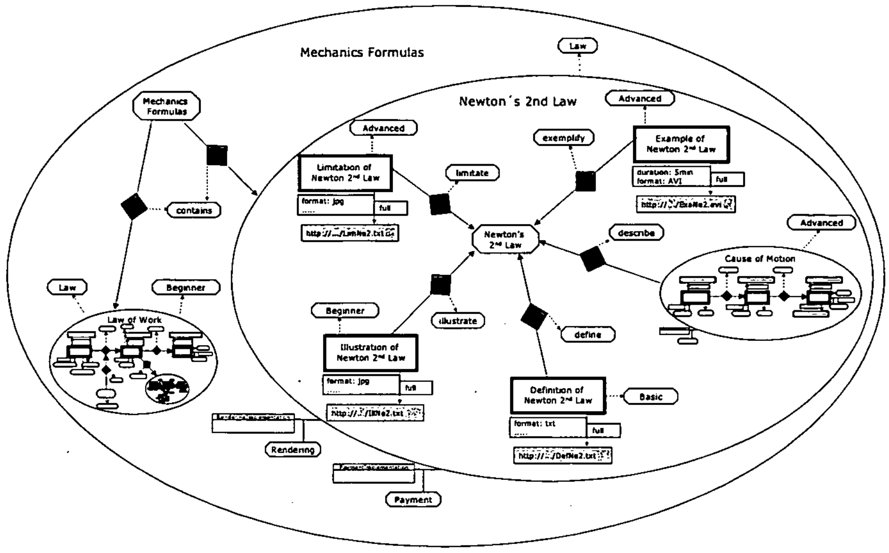


Figure 9.4: EMMO "Mechanics Formulas" ($e_{mechanics}$)

- By capturing the *media aspect*, an EMMO which models eLearning material encompasses not just a semantic description of the material but also all the media content of which the material consists readily available for consumption by the student. For instance, our example EMMO dealing with Newton's 2nd Law consists of a textual definition of the law, two JPEG images graphically illustrating the law and its limitations, and a video that exemplifies the law.

- The *semantic aspect* of an EMMO can provide a semantic description of the learning material it represents according to an appropriate eLearning domain ontology. By the EMMO model's support of encapsulation, learning material can be logically organized into hierarchies of self-contained modular units with well-defined boundaries. Thus, even complex content can be structured in a way that enables the reuse of learning modules in different contexts. In the example of Fig. 9.4, the EMMO "Newton's 2nd Law" encapsulates and refers to the EMMO "Cause of Motion", which may again have a complex structure. This embedded EMMO constitutes a logical unit on its own, which may be addressed, accessed, reused, and transferred independently from the outer EMMO. The ability to reify associations also comes handy: it could be applied to denote, e.g. the source and ownership of an association, copyright issues, etc.

- By its *functional aspect*, an EMMO can ship with its own rendering functionality for adequately presenting the learning material to the user and device. For instance, the operation "Rendering" attached to the EMMO about Newton's 2nd Law might create an HTML document containing the definition of the law, and two images illustrating the definition of the model and its limitation for a student who prefers a textbook approach for learning. For another student who prefers a visual style of learning, the same operation might generate an HTML document that makes use of these images as well but also includes the exemplifying video. Finally, a third student using a mobile device could receive only text documents and the low level images illustrating the definition. As rendering functionality is attached to the EMMO to which it applies and is no global system functionality, it is also possible to supply different EMMOs with different rendering operations tailored to their particular requirements avoiding the need of developing "one size fits all" rendering algorithms. Another functionality important in the eLearning context, are payment operations which check whether payment demands are fulfilled before allowing to consume the material represented by the EMMO.

- The expressivity of the *EMMA query algebra* enables the generation of multimedia representations of the physics content reflecting a specific task, context, and background knowledge of teachers or students. For example, an advanced student in physics might require for the final preparation of his exams a summary of only the definitions of mechanics laws, i.e. a document listing all media objects described by logical media parts of type "Definition" which are contained within EMMOs of type "Law".

- By *integrating ontological knowledge* described within the eLearning Ontology, imprecise queries can be refined. For instance, the knowledge that the domain of mechanics is just one part of the domain of physics can be used for refinement. If a student is asking for all media objects exemplifying a physics law, then he should also receive those media objects which are assigned to the examples of mechanics law, but not explicitly to the examples of physics law.

- The *EMMO containers* with their ability to persistently store EMMOs and their interfaces that permit the fine-grained traversal and manipulation of EMMOs as well as the execution of EMMO operations provide an ideal ground for the realization of an eLearning environment based on semantically modeled learning material. The infrastructure provides visual tools for the authoring of, the search for, as well as the personalized presentation of such material. By being further able to export and import whole EMMOs encompassing their media and semantic aspects as well as their functionality, EMMO containers may form the cornerstone of a distributed content sharing infrastructure for semantically modeled learning material that easily can be extended to a peer-to-peer network similar to Edutella.

- Given the profound *versioning* support incorporated into the EMMO model, a distributed network of EMMO containers would also support the efficient collaborative authoring of learning material.

9.3 Multimedia Task Management

Every employee in an enterprise or organization is expected to accomplish a specific set of tasks and jobs. Although some of these tasks might be simple standalone tasks, most tasks are of a more complex nature and consist of subtasks, which may have dependencies, have to be executed in a certain order, run in parallel, etc. To make things worse, subtasks are frequently under the authority of different persons. To get things done correctly, it is important for employees to know how these complex tasks are organized and which people are involved. A documentation of these workflows is therefore essential. However, in many organizations, such as a typical university, a system for managing the individual tasks is lacking. People simply know how to get their work done. If an employee is assigned a task he has never performed before, he will typically look for people who know how to perform the task and talk to them. However, if these people are unavailable, e.g. due to illness, and a deadline is approaching, things get stressful.

[GH95] define a workflow as collection of tasks organized to accomplish some business process, where a task can be performed by one and more software systems, one or a team of humans, or a combination of these. According to [Hol95], workflow management systems are then systems that define, manage, and execute workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic. Existing workflow management systems, such as [V+04] or [C+04], often focus on the design of complex distributed applications reusing existing software components, and rather neglect the integration of human tasks. Therefore, in many cases technical instructions for individual tasks are not provided. Within our approach, we focus on the workflow of tasks that are performed by humans. To ensure that the person who is responsible for performing a task has all the necessary knowledge available, tasks need to be described by multimedia-enhanced technical instructions.

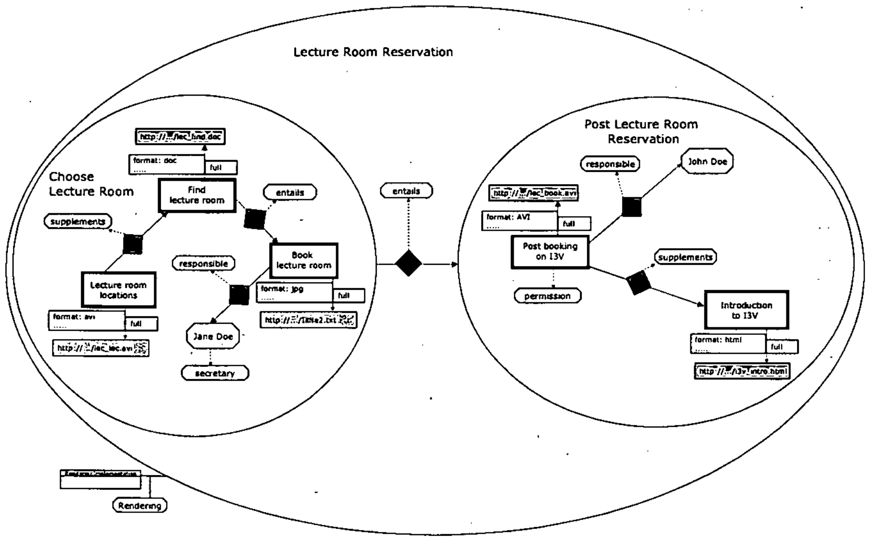


Figure 9.5: EMMO "Lecture Room Reservation" ($e_{reservation}$)

In the following, we want to illustrate how the EMMO infrastructure can be employed to realize a multimedia task management system. Figure 9.5 gives a simplified example of an EMMO that documents the task of reserving a room for a lecture at the University of Vienna, which represents just a small task possibly described within an university administration system. Using the example EMMO, we can demonstrate the benefits provided by our approach:

- By addressing the *semantic aspect* of multimedia content, EMMOs are very well suited for the description of tasks by means of an appropriate workflow ontology. Via associations, it is possible to extensively describe the interrelationships and dependencies between tasks, to describe the objectives achieved by a task, and to relate tasks to the persons responsible for them. With their container-like nature and their support for encapsulation, EMMOs further provide a natural means of structuring even very complex tasks. Each complex task, i.e. a task which consists of subtasks, can be represented by a corresponding EMMO; as these EMMOs can be included in arbitrary other EMMOs, it is possible to assemble more complex tasks from other tasks and to reuse a task in different contexts. The task in Fig. 9.5 consists of two subtasks each represented by corresponding EMMOs, namely the task "Choose Lecture Room" which "entails" the task "Post Lecture Room Reservation". Inside these subtasks primitive tasks exist that have no further substructure and are modeled as logical media parts. It is stated that choosing a lecture room involves identifying the most appropriate room and calling the person in charge of the rooms, Jane Doe, to find out whether it is available at the desired time slot to book it. It is also stated that after this, the reservation must be made known to the students by posting it on the university's central course administration system I3V. There is again a single person responsible for this (John Doe) who must be called to do so. It is also specified by further associations that supplementary information on how to reach lecture rooms and how to use the I3V system is available.

- By means of their *media aspect*, EMMOs are not just capable to describe tasks on an abstract level but also to enrich these abstract descriptions with illustrating media. In our example in Fig. 9.5, we have modeled primitive tasks as logical media parts, which can be augmented with profiles of media explaining how to perform these tasks, e.g. through textual descriptions, videos, etc.

- The *functional aspect* of EMMOs once more allows the realization of a context dependent and personalized rendering of task descriptions. For example, in case that the secretary gets ill and a visiting professor needs to reserve a lecture room himself, he executes the rendering operation associated to the EMMO "Lecture Room Reservation". The implementation of this operation could consider his user profile which might say that the visiting professor has limited knowledge of the locations of the lecture rooms and also no idea what the I3V system is. Therefore, the operation might produce a SMIL presentation, which not just includes the basic descriptions of the tasks but also the supplementary material available. Since the dependencies between the tasks are explicitly modeled within the EMMO, the implementation might also be able to take account of the progress of the task, explaining only those things that still need to be finished.

- By using the *EMMA* operators, one can access very detailed and personalized information captured by the task management system. For example, an employee who is only interested in the task he is responsible for asks for all associations of type "responsible" that specify the ontology object representing his person as target entity.

- By describing each employee and the hierarchical structure of the enterprise within the underlying *ontology*, this *knowledge* can be used for refining a query request. Thus, for example, a head of group can now easily access all the tasks the employees of his group are responsible for. As another example, if an employee responsible for a task is not available, the head of group can search for the employees responsible for the corresponding supertask and assume that they are also competent to attend to this task.

- The *EMMO container infrastructure* constitutes a suitable foundation for the realization of a task management system. EMMO containers provide persistent storage of and fine-grained access to EMMOs modeling tasks and permit the execution of EMMO operations. Moreover, through their ability to import and export EMMOs (and thus task descriptions) they render it possible to move and share EMMOs between different containers. Therefore, they fulfil essential prerequisites to establish a task management system in a distributed setting. This is particularly useful since different units within an organization might show a considerable autonomy, such as the different schools and departments of a university.

- Finally, versioning support of EMMOs, for instance, allows to express the differences in the structure of similar tasks occurring in different autonomous units, which is a very important functionality for the strategic management in many organizations.

9.4 Summary

In this chapter, we have illustrated the appealing prospects offered by the EMMO approach to multimedia content modeling by illustrating three attractive application scenarios. All three scenarios are distributed content sharing applications which benefit from the essential characteristics of the EMMO infrastructure, i.e. EMMOs covering the semantic, media, and functional aspect of multimedia content, providing versioning support, and implemented as exchangeable EMMO containers supported by an ontology-enhanced query mechanism. The first application scenario describes how the EMMO infrastructure has been successfully used for the modeling and exchange of multimedia enhanced intertextual threads in the CULTOS project. The second and third scenario, although not yet implemented, show the usefulness of an EMMO infrastructure for the realization of a distributed eLearning environment and a multimedia task management system.

In our future work, we will focus on a case study in the domain of eLearning to verify the feasibility and scalability of our approach in a real-world application. Additionally, we plan to extend the eLearning scenario by integrating the features and functionality established by a multimedia task management system for educational administration to demonstrate the benefits of the seamless integration of multiple ontology instances – possibly originating from different domains – within one realization of the EMMO approach.

Chapter 10

Conclusion

In this thesis we have introduced EMMOs as a novel approach for the collaborative and distributed modeling of multimedia content. An EMMO establishes a self-contained piece of multimedia content that unifies three aspects of multimedia content. The media aspect reflects that an EMMO aggregates the basic media objects of which the multimedia content consists, the semantic aspect enables the specification of semantic associations between an EMMO's media objects, and, finally, the functional aspect provides means for the definition of arbitrary, domain-specific operations on the content that can be invoked by applications. Moreover, EMMOs are versionable – they can be modified concurrently in a distributed environment – and tradeable, i.e. all three aspects of the multimedia content and the versioning information can be bundled into one unit and serialized into an exchangeable format. The formal basis of EMMOs are entities, which are characterized by thirteen properties. Entities occur in four different kinds, i.e. logical media parts representing the media data, ontology objects representing concepts of an ontology, associations describing binary relationships, and EMMOs aggregating semantically related entities.

For enabling the efficient retrieval of the knowledge represented by EMMOs, we have developed the query algebra EMMA. EMMA is based on the formal definitions of the EMMO model and defines five general classes: the extraction operators provide means to query an EMMO's three aspects as well as its versioning information. The navigational operators enable the navigation along an EMMO's semantic graph structure and also facilitate the integration of ontological knowledge. The constructors render it possible to modify, combine, and create new EMMOs. The selection predicates allow the selection of only those entities fulfilling a specific characteristic, and finally, the join operator relates several entities or EMMOs with a join condition. EMMA's operators enable the access to all information and aspects stored within EMMOs, are based on precise semantics, orthogonal, and arbitrarily combinable. Thus, EMMA offers a formal basis for query rewriting and optimization.

Both, the EMMO model and the EMMA algebra, establish a sound foundation for the integration of ontological knowledge. In this thesis, we have illustrated three different ways of using ontological knowledge within the management of EMMOs, i.e. for checking integrity constraints within the design and authoring process of EMMOs, for inflating and deflating an EMMO's graph structure, and for refining EMMA query expressions.

Moreover, we have implemented the platform independent and scalable EMMO container infrastructure that is supplied with adequate export and import facilities, the EMMO Viewer for displaying EMMOs, the EMMO Authoring Tool for the authoring of EMMOs, and the EMMA query processing architecture for the efficient retrieval of EMMOs.

By outlining three attractive application scenarios, i.e. a platform for the management of cultural knowledge, an eLearning environment, and a multimedia task management system, we have highlighted the benefits of using the EMMO infrastructure for distributed, collaborative applications. The first scenario has already been successfully deployed, which proved the feasibility and scalability of our approach in a real-world application.

In our future work, we will focus on the realization of the eLearning scenario. Additionally, we plan to extend the eLearning scenario by integrating the features and functionality described in the third scenario, i.e. the multimedia task management system. By integrating the underlying workflow ontology, we want to demonstrate the benefits of seamlessly integrating multiple ontology instances – possibly originating from different domains – within one EMMO application.

Based on real-world data gathered from this use case, we will carry out further experiments for performance evaluation, in particular to achieve a more detailed analysis and understanding of the effects of the various factors on query performance. We will use the application-specific data as starting point for the development of an EMMA query optimizer.

Furthermore, our future work will focus on the development of an ontology description language that is fully compatible with the EMMO model. To enable the integration of any arbitrary ontology structure, our intention is the development of an ontology description language with the same expressiveness as the standard ontology description language OWL DL, and to provide a converter between the two languages. This will guarantee full compatibility with other ontology creation initiatives.

Moreover, we plan to establish an EMMO Development Environment by integrating the EMMO container infrastructure, the EMMA query processing environment, and the ontology engineering tools to provide a profound basis for the seamless integration of ontological knowledge to unleash its full potential for efficient authoring, management, and retrieval of multimedia content.

Bibliography

[A$^+$97] S. Abiteboul et al. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[A$^+$01] J. Ayars et al. Synchronized Multimedia Integration Language (SMIL 2.0). W3C Recommendation, World Wide Web Consortium (W3C), August 2001.

[ASS99] S. Adali, M. Sapino, and V. Subrahmanian. A Multimedia Presentation Algebra. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, USA, 1999.

[B$^+$02] M. Benari et al. Organising the Knowledge of Arts and Experts for Hypermedia Presentation. In *Proc. of the Conference of Electronic Imaging and the Visual Arts*, Florence, Italy, May 2002.

[B$^+$03a] F. Baader et al., editors. *The Description Logic Handbook - Theory, Implementation, and Applications.* Cambridge University Press, Cambridge, UK, 2003.

[B$^+$03b] M. Benari et al. Proposal for a Standard Ontology of Intertextuality. Public Deliverable Version 2.0, CULTOS Consortium and Project Planning, June 2003.

[B$^+$03c] F. Billiani et al. Demonstrator of Intertextual Cultural Threads – Standard Ontology-Extended Ontology. Public Deliverable Version 2.0, CULTOS Consortium and Project Planning, January 2003.

[B$^+$05a] A. Berglund et al. XML Path Language (XPath). W3C Working Draft Version 2.0, World Wide Web Consortium (W3C), February 2005.

[B$^+$05b] S. Boag et al. XQuery 1.0: An XML Query Language. W3C Working Draft, World Wide Web Consortium (W3C), February 2005.

[Bau02] S. Baumeister. Enterprise Media Beans^{TM} Specification. Public Draft Version 1.0, IBM Corporation, March 2002.

[Bec04] D. Beckett. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[BFS00] P. Bruneman, M. Fernandez, and D. Suciu. UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. *The VLDB Journal – The International Journal on Very Large Data Bases*, 9, 2000.

[BG02] R. Barta and J. Gylta. XTM::Path – Topic Map Management, XPath Like Retrieval and Construction Facility. Online Article, available under http://cpan.uwinnipeg.ca/htdocs/XTM/XTM/Path.html, 2002.

[BG04] D. Brickely and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, World Wide Web Consortium (W3C), February 2004.

[BHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[BKW00] S. Boll, W. Klas, and U. Westermann. Multimedia Document Formats – Sealed Fate or Setting Out for New Shores? *Multimedia – Tools and Applications*, 11(3), 2000.

[Bog04] D. Bogachev. TMPath – Revisited. Online Article, available under http://homepage.mac.com/dmitryv/TopicMaps/TMPath/TMPath Revisited.html, 2004.

[BT99] C. Beeri and Y. Tzaban. SAL: An Algebra for Semistructured Data and XML. In *Proc. of the Second International Workshop on the Web and Databases (WebDB 99)*, Philadelphia, Pennsylvania, USA, 1999.

[C$^+$95] H. Chang et al. Tele-Action Objects for an Active Multimedia System. In *Proc. of the International Conference on Multimedia Computing and Systems (ICMCS 1995)*, Ottawa, Canada, 1995.

[C$^+$01] D. Connolly et al. DAML+OIL (March 2001) Reference. W3C Note, World Wide Web Consortium (W3C), December 2001.

[C$^+$02] M. Champion et al. Web Services Architecture. W3C Working Draft, World Wide Web Consortium (W3C), November 2002.

[C$^+$04] L. Costa et al. WebComposer: A Tool for Composition and Execution of Web Service-Based Workflows. In *Proc. of the Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, Ribeirao Preto-SP, Brazil, 2004.

[Cat94] R. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan, Kaufmann, San Francisco, CA, 1994.

[CZ01] S. Chang and T. Znati. Adlet: An Active Document Abstraction for Multimedia Information Fusion. *IEEE Transactions on Knowledge and Data Engineering*, 13(1), 2001.

[DLP98] R. Daniel, C. Lagoze, and S. Payette. A Metadata Architecture for Digital Libraries. In *Proc. of the Advances in Digital Libraries Conference*, Santa Barbara, California, 1998.

[DWG94] A. Duda, R. Weiss, and D. Gifford. Content Based Access to Algebraic Video. In *Proc. of the IEEE First International Conference on Multimedia Computing and Systems*, Boston, MA, USA, 1994.

[F$^+$00] Dieter Fensel et al. OIL in a Nutshell. In *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW '00)*. Springer-Verlag, 2000.

[F$^+$02] F. Frasincar et al. RAL: An Algebra for Querying RDF. In *Proc. of the Third International Conference on Web Information Systems Engineering (WISE 2000)*, Singapore, 2002.

[Fen01] D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce.* Springer, Heidelberg, 2001.

[FHH03a] R. Fikes, P. Hayes, and I. Horrocks. DAML Query Language (DQL). Abstract Specification, DAML Joint Committee, 2003.

[FHH03b] R. Fikes, P. Hayes, and I. Horrocks. OWL-QL – A Language for Deductive Query Answering on the Semantic Web. Technical report, Knowledge Systems Laboratory, Stanford University, Stanford, CA, 2003.

[FJJ03] J. Ferraiolo, F. Jun, and D. Jackson. Scalable Vector Graphics (SVG) 1.1. W3C Recommendation, World Wide Web Consortium (W3C), January 2003.

[Gar03] L. Garshol. Tolog 0.1. Ontopia Technical Report, Ontopia, 2003.

[Gen95] M. Genesereth. Knowledge Interchange Specification. Working Draft for an American National Standard, X3T2 Ad Hoc Group on KIF, March 1995.

[GH95] D. Georgakopoulos and M. Hornick. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[Gra93] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Survey*, 25(2):73–169, 1993.

[Gru93] T. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2), 1993.

[Hol95] D. Hollingsworth. Workflow Management Coalition – The Workflow Reference Model. Technical Report, January 1995.

[ISO86] ISO/IEC JTC 1/SC 34. Information Processing – Text and Office Sytems – Standard Generalized Markuplanguage (SGML). ISO/IEC International Standard 8879:1986, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), July 1986.

[ISO96] ISO/IEC IS 13522-5. Information Technology – Coding of Hypermedia Information – Part 5: Support for Base-Level Interactive Applications. International Standard, ISO/IEC, 1996.

[ISO97] ISO/IEC JTC 1/SC 34/WG 3. Information Technology – Hypermedia/Time-Based Structuring Language (HyTime). International Standard 15938-5:2001, ISO/IEC, September 1997.

[ISO00a] ISO/IEC JTC 1/SC 34/WG 3. Information Technology – SGML Applications – Topic Maps. ISO/IEC International Standard 13250:2000, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), February 2000.

[ISO00b] ISO/IEC JTC1 SC34 WG3. New Work Item Proposal, Topic Map Query Language (TMQL). New Proposal, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), December 2000.

[ISO01a] ISO/IEC JTC 1/SC 29/WG 11. Information Technology – Multimedia Content Description Interface – Part 5: Multimedia Description Schemes. Final Draft International Standard 15938-5:2001, ISO/IEC, October 2001.

[ISO01b] ISO/JTC 1/SC 32/WG 2. Conceptual Graphs. ISO/IEC International Standard, International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), April 2001.

[JH98] D. Juke and I. Herman. A Standard for Multimedia Middleware. In *Proc. of the ACM Multimedia Conference*, Bristol, UK, 1998.

[K$^+$95] M. Kifer et al. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the Association for Computing Machinery*, May 1995.

[K$^+$02] G. Karvounarakis et al. RQL: A Declarative Query Language for RDF. In *Proc. of the 11th International World Wide Web Conference (WWW 2002)*, Honolulu, Hawaii, 2002.

[KKS92] M. Kifer, W. Kim, and Y. Sagiv. Querying Object-Oriented Databases. In *Proc. of the ACM SIGMOND Conference on Management of Data*, San Diego, CA, 1992.

[L$^+$93] T. Leung et al. The Aqua Data Model and Algebra. In *Proceedings of the Fourth International Workshop on Database Programming Languages – Object Models and Languages*, Manhattan, New York City, 1993.

[L$^+$97] J. Li et al. MOQL: A Multimedia Object Query Algebra. In *Proc. of the Third International Workshop on Multimedia Information Systems*, Como, Italy, 1997.

[L$^+$99] T. Lee et al. Querying Multimedia Presentations Based on Content. *IEEE Transactions on Knowledge and Data Engineering*, 11(3), 1999.

[Lea98] P. Leach. UUIDs and GUIDs. Network Working Group Internet-Draft, The Internet Engineering Task Force (IETF), February 1998.

[LLD96] C. Lagoze, C. Lynch, and R. Daniel. The Warwick Framework: A Container Architecture for Aggregating Sets of Metadata. Technical Report TR 96-1593, Cornell University, Ithaca, New York, 1996.

[M+03] D. McGuinness et al. DAML-ONT: An Ontology Language for the Semantic Web. In *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, Cambridge, Massachusetts, February 2003.

[Mae02] A. Maedche. *Ontology Learning for the Semantic Web.* Kluwer Academic Publishers, Massachusetts, USA, 2002.

[MH98] V. Matena and M. Hapner. Enterprise Java Beans^{TM}. Specification Version 1.0, Sun Microsystems Inc., March 1998.

[Mic98] Microsoft Corporation. Distributed Component Object Model Protocol. Internet Draft Version 1.0, Microsoft Corporation, January 1998.

[Mil02] L. Miller. Inkling: RDF Query Using SquishQL. Online Article, available under http://swordfish.rdfweb.org/rdfquery/, 2002.

[Mil03] L. Miller. SWAD-Europe Deliverable 7.2: Databases, Query, API, Interfaces Report on Query Languages. Deliverable, Semantic Web Advanced Development for Europe (SWAD-Europe), IST-2001-34732 Project, 2003.

[MS98] M. Marchiori and J. Saarela. Query + Metadata + Logic = Metalog. In *Proc. of the Query Languages Workshop (QL'1998)*, Boston, Massachussets, USA, 1998.

[MSR02] L. Miller, A. Seaborn, and A. Reggiori. Three Implementations of SquishQL, a Simple RDF Query Language. In *Proc. of the First International Semantic Web Conference (ISWC2002)*, Sardinia, Italy, 2002.

[MSS02] B.S. Manjunath, P. Salembier, and T. Sikora, editors. *Introduction to MPEG-7.* John Wiley & Sons, West Sussex, UK, 2002.

[N+02] W. Nejdl et al. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proc. of the Eleventh International World Wide Web Conference (WWW 2002)*, Honolulu, Hawaii, 2002.

[NPS02] D. Newman, A. Patterson, and P. Schmitz. XHTML+SMIL Profile. W3C Note, World Wide Web Consortium (W3C), January 2002.

[O+95] M. Özsu et al. TIGUKAT: A Uniform Behavioral Objectbase Management System. *The VLDB Journal*, 4:100–147, 1995.

[OMG02] OMG Object Management Group. Common Object Request Broker Architecture: Core Specification. Specification Version 3.0, OMG Object Management Group, November 2002.

[OP98] I. Ounis and M. Pasca. Effective and Efficient Relational Query Processing Using Conceptual Graphs. In *Proc. of the 20th Annual BCS-IRSG Colloquium on IR*, Autrans, France, 1998.

[PE02] F. Pereira and T. Ebrahimi, editors. *The MPEG-4 Book.* Pearson Education, California, 2002.

[PGW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *Proc. of the Eleventh International Conference on Data Engineering*, Taipei, 1995.

[Pin04] R. Pinchuk. Toma. Online Article, available under http://www.space applications.com/toma/Toma.html, Space Applications Services, 2004.

[PM01] S. Pepper and G. Moore. XTM Topic Maps (XTM). TopicMaps.Org Specification, TopicMaps.Orgs, August 2001.

[PS04] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Working Draft, World Wide Web Consortium (W3C), 2004.

[RG03] A. Reggiori and D. Gulik. RDFStore. Online Article, available under http://rdfstore.sourceforge.net/, SourceForge.net, 2003.

[RHJ99] D. Raggett, A. Le Hors, and I. Jacobs. HTML 4.01 Specification. W3C Recommendation, World Wide Web Consortium (W3C), December 1999.

[Ron03] C. Ron. HyperPhysics. Website available under http://hyperphysics. phy-astr.gsu.edu/hbase/hph.html, 2003.

[S$^+$99] A. Steinacker et al. Multibook: Metadata for Webbased Learning Systems. In *Proc. of the Second International Conference on New Learning Technologies*, Bern, Switzerland, 1999.

[SD01] M. Sintek and S. Decker. TRIPLE – An RDF Query, Inference, and Transformation Language. In *Proc. of the Deductive Database and Knowledge Management Workshop (DDLP'2001)*, Japan, 2001.

[Sea04] A. Seaborne. RDQL – A Query Language for RDF. W3C Member Submission, World Wide Web Consortium (W3C), 2004.

[SHH04] P. Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, World Wide Web Consortium (W3C), February 2004.

[Sow00] J. Sowa. *Knowledge Representation – Logical, Philosophical, and Computational Foundations.* Brooks/Cole, Pacific Grove, USA, 2000.

[SSS01] L. Stojanovic, S. Staab, and R. Studer. E-Learning Based on the Semantic Web. In *Proc. of the World Conference on the WWW and Internet(WebNet2001)*, Orlando, Florida, USA, 2001.

[Sta04] Stanford Medical Informatics. What is Protege-2000? Online Documentation, available under http://protege.stanford.edu/, Stanford University School of Medicine, 2004.

[Sto04] D. Stockely. E-learning Definition and Explanation. Website available under http://derekstockley.com.au/elearning-definition.html, February 2004.

[SWZK03] K. Schellner, U. Westermann, S. Zillner, and W. Klas. CULTOS: Towards a World-Wide Digital Collection of Exchangeable Units of Multimedia Content for Intertextual Studies. In *Proc. of the Conference on Distributed Multimedia Systems (DMS 2003)*, Miami, Florida, 2003.

[V+04] T. Vieira et al. An Ontology-Driven Architecture for Flexible Workflow Execution. In *Proc. of the Joint Conference 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress*, Ribeirao Preto-SP, Brazil, 2004.

[WZSK05] U. Westermann, S. Zillner, K. Schellner, and W. Klas. EMMOs: Tradeable Units of Knowledge Enriched Multimedia Content. In U. Srinivasan and S. Nepal, editors, *Managing Multimedia Semantics.* IDEA Group Publishing, Hershey PA, USA, 2005.

[ZW04a] S. Zillner and W. Winiwarter. Integrating Ontology Knowledge into a Query Algebra for Multimedia Meta Objects. In *Proc. of the Fifth International Conference on Web Information Systems Engineering (WISE 2004)*, Brisbane, Australia, 2004.

[ZW04b] S. Zillner and W. Winiwarter. Ontology-Based Query Refinement for Multimedia Meta Objects. In *Proc. of the Sixth International Conference on Information Integration and Web Based Applications & Services (iiWAS 2004)*, Jakarta, Indonesia, 2004.

[ZW05] S. Zillner and W. Winiwarter. Integration of Ontological Knowledge within the Authoring and Retrieval of Multimedia Meta Objects. *International Journal of Web and Grid Services (IJWGS)*, 1(4), 2005.

[ZWW04a] S. Zillner, U. Westermann, and W. Winiwarter. EMMA – A Query Algebra for Enhanced Multimedia Meta Objects. In *Proc. of the Third International Conference on Ontologies, Databases and Applications of Semantics (ODBASE 2004)*, Larnaca, Cyprus, 2004.

[ZWW04b] S. Zillner, U. Westermann, and W. Winiwarter. EMMA – Towards a Query Algebra for Enhanced Multimedia Meta Objects. In *Proc. of the Fourth International Conference on Computer and Information Technology (CIT 2004)*, Wuhan, China, 2004.

Lebenslauf

Name:	Sonja Zillner
Wohnhaft:	Mittersteig 2/26 A-1040 Wien
Geboren:	21. Juni 1971 in Starnberg, Deutschland
Vater:	Hermann Zillner
Mutter:	Ingeborg Zillner, geb. Schnitzler
Nationalität:	deutsch
Familienstand:	ledig

Schule, Ausbildung und Studium

1978 - 1982	Volksschule Pöcking, Deutschland
1982 - 1991	Gymnasium Starnberg, Deutschland Abschluß: Abitur
1991 - 1999	Studium der Mathematik mit Nebenfach Psychologie an der Albert-Ludwig Universität Freiburg, Deutschland Abschluß: Diplom-Mathematikerin
Seit 2002	Doktoratsstudium an der Technischen Universität Wien

Beschäftigungsverhältnisse

1992	Praktikum Datenverarbeitung im Deutschen Zentrum für Luft und Raumfahrt (DLR), Oberpfaffenhofen, Deutschland
1993 - 1999	Programmiertätigkeit am Institut für Informatik und Gesellschaft, Universität Freiburg, Deutschland
1999 - 2000	Wissenschaftliche Projektarbeit am FWF-Projekt "BHUTAN - Ein virtuelles Museum" an der Universität Wien
2000 - 2002	Vertragsassistentin am Institut für Informatik und Wirtschaftinformatik der Universität Wien
2002 - 2003	Wissenschaftliche Projektarbeit am EU-Project CULTOS an der Universität Wien
2002 - 2004	Externe Lektorin am Institut für Informatik und Wirtschaftinformatik der Universität Wien
2004 - dato	Wissenschaftliche Assistentin am Institut für Distributed and Multimedia Systems der Universität Wien