INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

EOS: An Epistemological Ontology-driven System for Knowledge Processing

Dipl.-Inform. Univ. Wolfgang Wohner

Institut für Informatik
der Technischen Universität München

EOS: An Epistemological Ontology-driven System for Knowledge Processing

Dipl.-Inform. Univ. Wolfgang Wohner

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Helmut Krcmar
Prüfer der Dissertation:	1. Univ.-Prof. Rudolf Bayer, Ph.D.
	2. Univ.-Prof. Dr. Manfred Paul, em.

Die Dissertation wurde am 27. Februar 2003 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 22. Oktober 2003 angenommen.

Title of the Thesis:

EOS: An Epistemological Ontology-driven System for Knowledge Processing

Author:

Dipl.-Inform. Univ. Wolfgang Wohner

Keywords:

Knowledge Representation and Processing, Ontologies, Epistemology, Concept Theory

Abstract:

This thesis introduces EOS, a new approach to knowledge representation and processing that is based on Concept Theory, a unicategorical formalism we developed for defining semantically enriched formal ontologies. Generally, ontologies are a means for knowledge representation and reuse. They provide a shared understanding about a knowledge domain. EOS extends the scope of ontologies by offering means to also formalize epistemological processes, i.e. guidelines for knowledge processing that may be employed in knowledge acquisition, generation and retrieval tasks. This general idea is molded into an EOS framework for epistemological ontology-driven systems for knowledge processing. On this formal basis we will show how an actual EOS system can be designed, implemented and successfully put to work.

Table of Contents

Chapter 1 A Reader's Guide

This thesis introduces the **EOS** approach to knowledge representation and processing. EOS reads '**E**pistemological **O**ntology **S**ystem', indicating the essential aspects of designing computerized management of human knowledge using EOS – epistemological and ontological properties, as well as their reconciliation in a unifying framework. Epistemology[1] and Ontology[2] are terms affiliated with using and representing knowledge, that have been borrowed from philosophy where theories of knowledge have a long tradition. In order to provide a better access to the basic ideas behind these terms, as well as their influence on current research activities in the IT community, we will subsequently offer a short introduction to their origins, the principles they embody and how they are reflected in different areas of computer science. This perspective will then be used for motivating the goals and requirements of the EOS approach.

Ontology (literally: *the study of being*), in its traditional meaning, is used in two senses. [46] distinguishes the general meaning of ontology depicting the "department of metaphysics concerned with the nature of existence" as a whole from the more particularized usage of the term referring to a "specific theory" thereof. In this thesis we are looking at ontologies exclusively in the latter sense of the word, taking the information science perspective, i.e. we understand ontologies as theories, or models, describing the semantics of a particular knowledge domain using a specific syntax. In information science, recent research activities have led to the notion of *formal ontologies*, providing knowledge representations of different fields of expertise, i.e. domains. Formal ontologies are used to describe the terms used, as well as their semantics, implications and interrelationships within such domains, e.g. an "enterprise ontology" that outlines a company's structure and the internal vocabulary used by its employees. From this perspective, any knowledge representation may be regarded as a formal ontology. Thus an ontology engineer can look at a relational Entity/Relationship (E/R) model and an UML diagram alike, identifying both of them as formal knowledge representations, i.e. as formal ontologies. Their individual representations will certainly differ in format and UML

[1] The term epistemology is built from the Greek words *episteme* (engl. *scientific knowledge*) and *logos* (engl. *word*, *definition*).

[2] The term *ontology* is derived from the Greek words *ontos* (engl. *to be*) and *logos* (engl. *word, definition*). Apart from depicting different metaphysical theories *Ontology* (then written with a capital '*O*') is often used as a synonym for metaphysics. In order to avoid misinterpretations we refrain from this latter usage of the word.

diagrams serve a different purpose than E/R models, yet the common goal of these formalizations is to provide a conceptualization of a certain problem domain – in the case of an E/R model this will include the entities, attributes and relations for an application scenario of a relational database system, whereas UML diagrams are directed towards developing a specific software application using object-oriented modeling techniques. While E/R models, UML diagrams and comparable modeling techniques serve a specific purpose in their respective fields of application, formal ontologies are a general means for designing domain models, not presupposing any particular application area. A cardinal aim of formal ontologies is to create a *shared understanding* of a given subject area [72]. This shared understanding, as provided by ontologies, may then be used in a variety of contexts. Generally speaking, by presenting a shared understanding, formal ontologies provide a means for communication between or among people, organizations, and/or software systems alike. This thesis will certainly concentrate on the employment of ontologies in the context of software systems, and in particular applications that base decisive parts of their behavior dependent on the formal ontologies they accept as an input.

Epistemology (literally: *the study of knowledge*) focuses on questions about the nature of (human) knowledge, i.e. what is knowledge and what can be known. As pointed out above, formal ontologies are being regarded as shared understandings of subject areas, i.e. *knowledge representations* describing the respective domains. Thus, in the context of formal ontologies epistemological questions center around how formal ontologies should be structured, what they are able to express, and especially how they can be successfully used by software systems. Questions of this kind determine on the one hand the need for a concise formalism for ontologies that act as a basis for *epistemological processes*, i.e. tasks a software system has to manage, which thus influences the behavior of the system. On the other hand, the definition of such system internal processes themselves becomes a significant subject matter. Recognizing this, we designed an ontology formalism we call *Concept Theory*. Concept Theory defines the structural elements of formal ontologies within the EOS approach. Next to domain knowledge EOS also distinguishes different classes of epistemological processes that find their ontological specification within Concept Theory. This means that the EOS approach extends the scope of formal ontologies as used in current systems (cf. [75] and [76]). EOS ontologies based on Concept Theory are not only depicting a shared understanding of a subject domain but are also covering *application semantics*, aspects about the system behavior, e.g. inference rules or guidelines for (personalized) query processing.

In a nutshell, the EOS approach provides the theoretical foundation for formalizing different *kinds* of knowledge that are required for successful knowledge processing. Such formalizations can be passed as an input to an actual EOS system, a technique that is described within the EOS framework. Unlike other knowledge processing systems, an EOS system accepts not only a formal domain model but also schematized application semantics about the manner in which domain knowledge should be handled. As a direct consequence, EOS systems are more flexible than conventional systems because EOS allows for *parameterized* knowledge processing. Encoding domain knowledge in a standardized way in order to create a shared understanding that can be communicated to other parties, belongs to the *ontological* foundation of the EOS approach. Its *epistemological* competency stems from the expressive power of EOS, which allows for an explicit modeling of application semantics based on the same formalism that is used to express domain knowledge, namely Concept Theory. The theoretical framework provided by EOS thus lays the necessary groundwork for designing and implementing a full-fledged knowledge processing system.

1.1 Motivation and Objectives

In order to characterize the important role of knowledge representation and processing in information systems, in this motivating section we will refer to a specific class of such systems, namely *digital libraries*.

1.1.1 Digital Libraries

Digital libraries (DL) provide access to electronic information collections that are made available e.g. using Web and database technology. Collections of this kind may range from publicly accessible Web indices, newspaper archives and library catalogues to collections that are intended for a restricted audience only, e.g. a company's knowledge base consisting of internal project documentation, white papers, best practice sheets, etc. Common to any of these application areas for DL is the basic need for an efficient management of huge amounts of information, often derived from heterogeneous sources. Efficiency in this respect entails both, short response times and high quality of service, especially in terms of assisting users in finding the information they are looking for. Therefore, next to storage and preservation of information the foremost services of DL are concerning solutions for comfortable information retrieval and delivery.

DL typically offer standardized interfaces for querying the information held within their collections. The most common query interfaces use *keyword-based search*, *categorized search* or a combination thereof. Keywords and phrases, such as 'database' and 'database managing system' are eventually transformed into a fulltext search that produces all information items (e.g. documents stored within a DL) that contain matching text passages. Many DL also provide additional features, such as Boolean operators for formulating more complex queries, e.g. 'schema OR model', or a special syntax for specifying text patterns and truncation. Categorized search, on the other hand, offers the user a predefined set of categories and subcategories for navigating through the information space of the DL. These categories, e.g. 'news', 'sports' and 'politics', are representing classes of information items that are intended to provide a structural view of the DL. Finally, combining keyword-based and categorized search yields a query interface where the user can chose keywords that will only be matched against the class of information items as determined by one or several selected categories.

The known drawbacks of these search strategies are the limited expressiveness of the query languages and the insufficient treatment of semantic text properties such as linguistic diversity or contextual semantics. Keyword-based approaches fail to handle e.g. homonyms such as the word 'kiwi', as one example of linguistic diversity, in a satisfactory way, i.e. searching for 'kiwi' may yield documents on the bird 'kiwi', as well as on the fruit 'kiwi' or on New Zealanders who are sometimes nicknamed 'Kiwis'. Categorized search provides limited contextual semantics that must be chosen on a rather broad level in order to remain generally applicable. Yet, exactly for this reason, they cannot account for the specific information needs of users at different times. In order to overcome these drawbacks, DL can benefit greatly from conceptual approaches that have recently emerged in the research fields centering on knowledge representation and processing, e.g. a refined conceptual model that is accessible to the user in combination with an appropriate query language would enable a DL user to formulate queries that exactly meet his or her information needs. In order to be able to discuss questions of this kind in greater detail and on a more practical basis, in the following sections we will refer to an actual DL system and its specific domain background.

1.1.2 The Digital Library Project VD17

The DL project called VD17[3] has been brought to life in order to build, maintain and provide for public electronic access the German National Bibliography of the years 1601-1700. VD17 is funded by the *Deutsche Forschungsgesellschaft* (*DFG*, engl. *German Research Foundation*), and is currently being built in a joint effort of six German partner libraries[4] holding major stocks of 17th century prints. The cataloging phase of this project started in 1996 and will continuously be financed by the DFG for a total period of 10 – 12 years. As a central goal, VD17 aims at establishing an extensive, for the most part complete bibliography of the estimated 300,000 baroque works still existing today that have been originally published in the German language area of the 17th century. Cataloging data within this bibliography is being stored in a central database and supplemented by a number of distributed image databases holding 1.5 million digitized images of key pages, and provided for world-wide access via the Web.

1.1.2.1 Data Integration in the Project VD17

In order to succeed, on the technical side VD17 is dependent on the services of a distributed DL system allowing for cooperative cataloging and advanced searching techniques. For this reason, FORWISS, the Bavarian Research Center for Knowledge-based Systems, has been called into the project as a technical partner. FORWISS designed and implemented a DL system tailored to the particular needs and specifications of the project, including a distributed relational database system holding VD17 records as well as complementary pixel images, along with additional software tools. Key issues that had to be addressed when designing the VD17 software components were (cf. [45], [44]):

- Supporting cooperation and collaboration of libraries located in different federal states of Germany and therefore belonging to different library networks, via the Internet.
- Design and implementation of a data model which meets the needs of German cataloging formats and rules.
- Linkage of the digitized key pages distributed over several databases to their respective bibliographical records within the central VD17 database.
- Providing world-wide access to the VD17 database for Web-based research.
- Integration of already existing machine-readable bibliographical records of other national and international book collections covering German 17th century prints.

While the first four issues of the above list pertain to the immediate realization of the project, which is under the sovereignty of the project partners, integration of foreign records poses demands of a different nature. In order to successfully import data from foreign sources, it must be assured that this data either complies with native formats and concepts, or can be easily translated into them. In case of VD17, the accepted cataloging format is MAB[5], a Ger-

[3] VD17 is an acronym for the German project title "*Verzeichnis der im deutschen Sprachraum erschienenen Drucke des 17. Jahrhunderts"* whose general sense may be translated into English as "*German National Bibliography 1601 - 1700*".

[4] Active partner libraries are the State Library of Berlin, the Bavarian State Library Munich, the Herzog August Library Wolfenbüttel, the State and University Library of Dresden, the Research Library of Gotha, and the State and University Library of Halle. The number of libraries participating in the VD17 project will further increase during the funding period of the DFG. Another three libraries have already been approved. Additionally, bibliographic records of other libraries that meet the high standards of the VD17 records will also be included, as has already been

[5] The original MAB (ger. Maschinelles Austauschformat für Bibliotheken) had been brought forward by the *Deutsche Forschungsgemeinschaft (DFG)* who initiated and financed the development of a standardized machine-readable exchange format for German libraries in the early 1970s. The MAB working group who has ever since been responsible for further development and amendments of MAB was later integrated into the *Deutsche Bibliothek* (German National Library). In 1995 a revised version of MAB called MAB2 was announced by the Deutsche Bibliothek in order to adapt the German exchange format so that it could meet the new requirements of exchanging data in online environments. Thus, MAB2 is a

man library standard (cf. section 1.1.3.1). VD17 records are thus compatible with the vast majority of German library systems. Therefore, integration of bibliographical records from these libraries poses little technical problems, e.g. in 1999 a set of 17,500 MAB records provided by the *Ratsschulbibliothek Zwickau*[6] were successfully imported into the VD17 database. However, records in foreign formats, e.g. MARC[7] as defined by the Library of Congress (LC), are usually not directly convertible into MAB without extensive manual work. Issues that must be taken into consideration in this context are:

- *Cataloging Format*. What a machine-readable cataloging format (e.g. MAB, MARC) describes is a classification schema that defines the structure of parsable bibliographical records. Firstly, a cataloging format specifies a set of entities (e.g. *author*, *title*) that are apt to identify a particular copy of some work, such as a book or a CD. Secondly, it describes the structure of the work itself (e.g. of a multi-volume work). Thirdly, it defines the internal organization of bibliographical records (e.g. how information about single copies of a work and information about the work itself are grouped together, and marked accordingly for automated parsing). Different cataloging formats may thus be incompatible because of several reasons (leaving separate naming schemes for semantically identical entities aside):
  - Entities of one format cannot be matched correctly to entities of another format (e.g. the different understanding of the concepts *work* and *item* in MAB and MARC).
  - Structures cannot be matched correctly from one format to the other (e.g. multi-volume books are treated differently in MAB and MARC).
  - Parsing information cannot be matched correctly (e.g. MAB and MARC are not fully compatible concerning their record structures, i.e. the semantics of their parsing instructions are also differing).
- *Cataloging Rules*. Any cataloging format requires additional cataloging rules on how to insert bibliographical data into the cataloging schema as defined by the standard format, e.g. how names and initials of persons are quoted. As a common practice, cataloging rules are standardized separately from cataloging formats. Thus, records complying to one set of rules defined for a specific format (e.g. German RAK[8] rules for the MAB format, cf. section 1.1.3.1) and records created according to a different cataloging rules standard may be incompatible even if they relate to one and the same format.

As a consequence, importing foreign records into the VD17 database without manual human work must fail if the bibliographical data does not follow the cataloging standards used within the project (MAB and RAK). Even for a notedly German project like VD17 this poses a serious drawback because important stocks of German 17th century prints can be found worldwide, especially in the UK and the US, where other cataloging standards apply. The importance of being able to process non-MAB formats within the bounds of VD17 becomes a vital factor for German libraries regarding their general holdings. Today, the share of Anglo-American books present in German libraries amounts to 54% on average (cf. [55]) which

rather modern format compared to other library standards, even MARC. In this thesis we will not differentiate between the original MAB and MAB2 but subsume them uniformly as MAB.

[6] The *Ratsschulbibliothek Zwickau* is a German library not associated with the project.

[7] MARC defines a data format which originally emerged from a Library of Congress led initiative begun in the 1970s. MARC became USMARC in the 1980s and MARC 21 in the late 1990s. There also exist several other MARC dialects such as the British UKMARC or the internationally maintained UNIMARC that differ slightly from MARC 21 as defined by the Library of Congress. We will not go into detail concerning these minor differences within the rather homogeneous MARC world and simply refer to them as MARC.

[8] RAK is an acronym for the German "*Regeln für die Alphabetische Katalogisierung*" (engl. *Rules for Alphabetic Cataloging*).

turns the ability to parse already existing bibliographical records of Anglo-American libraries (complying to MARC) into an economic factor. At present, German library systems working with MAB are not able to automatically import bibliographical data in foreign formats. Such records still require individual manual labor before they can be integrated into local German systems, which is an inefficient and costly procedure.

This describes a current dilemma within the library world. On the one hand, libraries are heavily using DL systems for maintaining and offering their bibliographical data, which allows for fast and easy data exchange and integration in electronic form. On the other hand, data exchange and integration requires cataloging standards that are universally accepted. As already pointed out in this section, existing cataloging standards, such as MAB and MARC, are not compatible with each other. Several national and international initiatives (cf. [28], [27]) have therefore tried to harmonize library standards. However, to this day neither a universally accepted new standard has evolved, nor has one already existing standard replaced the others. There are different reasons for this status quo, from an economic and from a cataloging point of view. [37] present an impressive list of disadvantages in case German libraries were to abandon their cataloging standards. An immediate solution to this problem on conventional grounds has not been found during long years of debate. Yet, recent developments in the area of defining and using metadata in the context of knowledge representation and processing promise to offer new perspectives. Regarding cataloging standards as metadata on bibliographical records related to the items of library holdings makes advances e.g. in the area of ontology engineering applicable to this problem.

1.1.3 Cataloging Standards and Metadata

This section takes a closer look at some prominent cataloging standards that define formats and rules for bibliographic records such as the information stored and managed by the VD17 DL. The discussion will mainly concentrate on MAB/RAK and MARC/AACR, and how they touch on related metadata and Web standards such as XML and Dublin Core (DC).

MAB (ger. Maschinelles Austauschformat für Bibliotheken) and MARC (MAchine-Readable Cataloging) are standard formats for the representation and communication of bibliographic and related information in machine-readable form. These formats define the structure of bibliographical records, similar to DTDs[9] that are describing the structure of XML[10] documents. From a technical point of view, MAB and MARC thus are metadata schemas for cataloguing books and other media according to a set of predefined categories such as *book*, *title*, *author*, *year*, etc. Bibliographical records complying to these standard formats are (in their electronic form) machine-readable classifications that can be processed and used by library applications, such as the VD17 DL. In particular, data exchange with the VD17 database has been realized via a MAB interface, i.e. the DL system reads in records complying to MAB which are then stored in a relational database, from where records can again be exported in MAB format.

Librarians are creating and maintaining bibliographical records, i.e. standardized descriptions of (mainly) printed publications. This activity originally involved hand-written catalog cards that were held and organized in (physical) catalog shelves inside a library building – usually organized redundantly, sorted once per authors' names, and once per subject area. Since the 1970s bibliographical records were increasingly stored digitally, so that, nowadays, electronic catalogs have substituted former card systems in most institutions. Bibliographical

[9] DTDs (*D*ocument *T*ype *D*efinitions) are used for specifying the structure and elements of XML documents.

[10] XML, the e*X*tensible *M*arkup *L*anguage, is used for generating semi-structured documents containing user defined meta-information (markup) on the textual content of the documents.

records in modern systems are, e.g. CIP-records and tables of content when included in the document, as well as catalog records when stored in separate Online Public Access Catalogs (OPACs) or abstract and index databases. From a technical point of view, MAB and MARC records are digital representations of information used for describing (mainly) print media and thus simply are a specific kind of metadata. Metadata, generally speaking, consists of descriptions of objects, documents or services which may contain information about their form and content. This means that metadata represents "data about data" and is thus semantically independent from the data it is describing. Therefore, it may be part of the resources themselves or kept separately from them. Concerning print media held in stock by a traditional library, bibliographic information is naturally kept separate from the works themselves, either on catalog cards or, nowadays, in electronic records.

With the advent of the Web and the immensely growing supply of electronic documents (e.g. in the formats HTML, XML or PDF) the possibility to store a work along with its accompanying metadata has become ubiquitous. Analogous to the need for effective classification schemas for printed works that resulted into the definition of cataloging formats, the huge amount of digital documents demands corresponding metadata standards in order to facilitate searching and filtering the information offered on the Web. In this context DC, a comparatively new metadata standard for digital documents, became popular.

1.1.3.1 MAB and RAK

The first version of the German MAB standard was published in 1972. MAB defines a cataloging format comprising about 700 different categories, called *fields*. Developed and promoted by the Deutsche Bibliothek, it quickly became the standard format for exchanging bibliographical records among German libraries in electronic form. An excerpt of a MAB classification of the VD17 book "*Der aus dem Grab der Vergessenheit wieder erstandene Simplicissimus*" by Hans J.C. v. Grimmelshausen is given in Figure 1.1. It contains original bibliographical data extracted from the VD17 database.

```
001 23:233330V
005n19961113
100 Grimmelshausen, Hans Jakob Christoffel ¬von¬
101 Schleifheim von Sulsfort, German
198 Boener, J. A.
198 Meyer, Joh.
331 ¬Der¬ Aus dem Grab der Vergessenheit wieder erstandene
    Simplicissimus
335 Dessen Abentheuerlicher und mit allerhand seltsamen/ fast
    unerhörten Begebenheiten angefüllter Lebens-Wandel ...
    vermittelst Scharfsinnigen Lehren/ wohlkommenden Anmerckungen
    und schönklingenden Poetischen Versen/ auch recht lebhafften
    Kupffer-Bildnüssen ... vorgestellet wird
410 Nürnberg
412 Felßecker
413 Nürnberg
414 Felsecker, Johann Jonathan
505 Nebent.: ¬Deß¬ Teutschen Simplicisimi Redi-vivi Lust- und
    Lehr-reicher Schrifftenmarck
511 Erscheinungsjahr im Bd. 1 auf Zwischent.: 1683. - Bd. 3 von
    1699 erschien bei Felsecker, Johann Jonathan <Erben>
523 Erschienen: 1 (1683) - 3 (1684); 3 (1699)
675 Anmerkungen // Kupferbildnissen // Schriftenmarkt
710 Roman
```

Figure 1.1: MAB Example

As indicated in Figure 1.1, MAB is built upon a category schema consisting of 3-digit numbers that are representing predefined concepts, e.g. *author*, *title*, belonging to the same work (usually a book). For example, the line

```
100 Grimmelshausen, Hans Jakob Christoffel ¬von¬
```
[11]

simply states that the string "`Grimmelshausen, Hans Jakob Christoffel ¬von¬`" falls into MAB category `100`. The semantics of category `100` associate record entries in this field with the *first author* of the respective book. Accordingly, in the above line "`Grimmelshausen, Hans Jakob Christoffel ¬von¬`" must be interpreted as the name of the first author. Similarly, category `331` indicates the *main title* ("`¬Der¬ Aus dem Grab der Vergessenheit wieder erstandene Simplicissimus`") of the same book, category `335` is used for *additional titles* ("`Dessen Abentheuerlicher und mit allerhand seltsamen/ fast unerhörten Begebenheiten angefüllter Lebens-Wandel…`"), and, likewise, all remaining categories are interpreted according to their MAB semantics as defined in the MAB standard.

Most MAB fields are further subdivided into a set of related concepts, or subcategories, using 1-digit MAB *indicators*. In the second line of the MAB record in Figure 1.1, for example, indicator `n` is used with category `005`[12]. Next to classifying single works, MAB can also be used to catalogue hierarchical structures such as multi-volume works, collections (i.e. several works published in one volume), supplements (e.g. maps or CDs), etc. In fact, Figure 1.1 shows an excerpt of a MAB record that, in its entirety, describes a three-volume edition of the novel by Hans J.C. v. Grimmelshausen.

As MAB only defines a cataloging *format*, it does not per se account for any cataloging *rules*, i.e. the exact way in which bibliographical information of a particular work is transferred into the category schema. This is not a trivial question, i.e. bibliographical records may take on very different forms depending on the cataloging rules applied. Referring to the above example, the line

```
100 Grimmelshausen, Hans Jakob Christoffel ¬von¬
```

suggests that MAB category `100` must be of the form

```
100 <last_name>, <first_name>.
```

Yet, strictly, this is not the case. In fact, MAB itself does not specify the internal structure of category entries. Thus, the above line could as well be written

```
100 Hans Jakob Christoffel ¬von¬ Grimmelshausen
```

and would (according to the format definition by itself) still be MAB compliant. However, for machine processing it is necessary that record entries do follow a predefined schema in order to render applications able to interpret MAB data correctly. This is done using cataloging rules. In general, cataloging rules are defining on the one hand *what* data about a work will be used for creating a bibliographical record, e.g. whether a foreword author is mentioned at all

[11] According to German cataloging rules, using the non-sort markers (ger. *Nichtsortierzeichen*) '¬' causes the enclosed word to be omitted in sorting processes, e.g. '¬von¬' signals that the substring 'von' in '`Grimmelshausen, Hans Jakob Christoffel ¬von¬`' should not be considered by MAB parsers when building e.g. a fulltext index on the respective category, in this case category `100`.

[12] Category `005` within MAB denotes the date of change of the respective records, i.e. whenever the contents of a MAB record are altered (e.g. for correcting or completing bibliographical data), category `005` serves as a time stamp of this transaction. MAB provides two indicators for category `005`, *n* and *v*: `005n` specifies the date of the last transaction on the record, while `005v` denotes the date of the last but one transaction.

within the record or not[13]. On the other hand cataloging rules are describing *how* the data is entered into the category schema, e.g. RAK specifies how names of persons within MAB category `100` are treated.

RAK consists of a body of about 430 paragraphs containing a complete set of cataloging rules for the MAB standard. Originally, RAK was designed for cataloging print media using traditional index cards, but has already been modified for a better support of electronic and online processing techniques (cf. [37]). The RAK standard is subdivided into groups of rules covering basic concepts (§§ 1-35), general rules (§§ 101-193), main headings (§§ 201-208), person names (§§ 301-342), corporate bodies (§§ 401-486), titles (§§ 501-524), main and added entries (§§ 601-696), titles for main and added entries (§§701-715), and additional rules (§§ 801-823).

MAB format definitions and RAK cataloging rules are specified in natural language and intended for guiding librarians when cataloging books and other media. Thus, MAB and RAK cannot be used directly e.g. as an input to a knowledge processing system that would then be able to correctly interpret bibliographical records that comply with MAB/RAK. Consequently, for any system that does work on bibliographical records MAB and RAK definitions have to be translated into the design of the system. This can be achieved either by implementing these definitions into the algorithms of the different system components, or by using a separate knowledge representation of MAB/RAK which can be parsed by the system. In the VD17 project we chose a middle way between these two options. VD17 system components know the hierarchical makeup[14] of MAB and the basic structure of MAB records. The category schema and according processing instructions are defined in a separate so-called "configuration file". The format of this configuration file has been designed so that VD17 components can parse it efficiently. In its simple form, the VD17 configuration file therefore is a knowledge representation of MAB categories and how they should be processed by the VD17 DL.

1.1.3.2 MARC and AACR

The American cataloging format MARC was designed by the LC and put to use in 1969. While MAB is basically limited to German libraries, MARC has found its application area on an international scale, encompassing among other countries the complete Anglo-American language area. The set of about 330 MARC data elements make up the foundation of most library catalogs currently used in these countries. Because of their extensive publishing productivity and economic weight, today, MARC is the most recognized format for exchanging bibliographical records among libraries all over the world. Consequently, even libraries that are using other authoritative bibliographical standards, such as the German MAB, are facing the need for translating their local formats into MARC, be it because they want to integrate foreign records into their own systems or that they are offering their bibliographical data to the international library community.

[13] Neither MAB nor MARC possess a separate category *foreword author*. It depends on the cataloging rules if a foreword author is mentioned at all, e.g. on equal terms with the book authors. With MAB this is not the case, i.e. information about foreword authors is omitted completely, while MARC cataloging rules allow for mentioning foreword authors in the *title* category.

[14] MAB knows a strict hierarchical schema, the "Gesamttitelhierarchie", for cataloging multi-volume works: information that is valid for all volumes (e.g. publisher, general title, etc.) is stored separately from information about the different volumes, and these records are interconnected. The same accounts for bibliographical data about the various copies of these volumes. The resulting hierarchical structure defines the work as a whole, respecting multi-volume works as such within MAB record structures. MARC cannot handle such structures: either the complete information about a work is stored in one single record (including all general and volume information), or for each volume there exists a separate record (without reference to the other records).

```
005 19961113
100 1# $a Grimmelshausen, Hans Jakob Christoph von, $d 1625-1676.
245 14 $a Der Aus dem Grab der Vergessenheit wieder erstandene
    Simplicissimus $b Dessen Abentheuerlicher und mit allerhand
    seltsamen/ fast unerhörten Begebenheiten angefüllter Lebens-
    Wandel ... vermittelst Scharfsinnigen Lehren/ wohlkommenden
    Anmerckungen und schönklingenden Poetischen Versen/ auch recht
    lebhafften Kupffer-Bildnüssen ... vorgestellet wird
260 ## $a Nürnberg, $b Felßecker, Johann Jonathan, $c 1683-99.
312 #4 Deß Teutschen Simplicisimi Redi-vivi Lust- und
    Lehr-reicher Schrifftenmarck.
214 ## Anmerkungen, Kupferbildnissen, Schriftenmarkt
```

Figure 1.2: MARC Example

Figure 1.1 shows a bibliographic record in MARC. Similar to MAB categories, each bibliographic record in MARC is divided logically into *fields*. There is a field for the author, a field for title information, and so on. These fields are subdivided into one or more *subfields*. MARC fields are represented by 3-digit number, or *tags*[15]. A tag unambiguously identifies the field, the kind of data, that follows, e.g. an author or title entry. The tag is followed by one or two one-digit *indicators*, usually numbers, though MARC also allows letters to be used for specifying indicators. Unlike MAB indicators who specify subcategories, MARC indicators mostly denote processing instructions, e.g. the second indicator for the title field `245` displays the number of characters at the beginning of the field (including spaces) to be disregarded by the computer in the sorting and filing process – an instruction that MAB expresses using the non-sort markers '¬' (cf. the discussion following Figure 1.1). As already mentioned, most MARC fields contain several related pieces of data that refer to subfields. For example, the field for a book's physical description (defined by the tag 300) includes a subfield for the extent (number of pages), a subfield for other physical details (illustration information), and a subfield for dimensions (centimeters). Each subfield is preceded by a *subfield code*, commonly a lowercase letter (occasionally a number) preceded by a *delimiter*, i.e. a character used to separate subfields (in the example MARC record shown in Figure 1.2 the delimiter character is '$'). Each subfield code indicates what type of data follows it.

Cataloging rules for MARC records are defined in the AACR standard that has been designed for use in the construction of bibliographic catalogues and other lists in general libraries within the Anglo-American community. The rules cover the description of, and the provision of access points for, all library materials, such as print media, CDs, etc. Similar to the RAK standard for MAB records, AACR rules are the standard way to create and interpret bibliographical data complying to MARC. The AACR standard is divided into two parts, one covering rules dealing with the different types library materials, and the second defining how to handle headings, uniform titles and references. knowledge representation of MAB categories and how they should be processed by the VD17 DL.

1.1.3.3 Dublin Core

The name 'Dublin Core' (DC) is actually a shorthand for the Dublin Metadata Core Element Set, a list of metadata elements that has been agreed upon at the OCLC/NCSA Metadata Workshop in March 1995 (cf. [74]). The objective of DC has been to define a minimal set of

[15] Though on-line catalogs using MARC as their internal format may display the names of the fields, the names are supplied by the system software, not by the MARC record. The same accounts for MAB based systems such as the VD17 DL.

15 data elements so that authors and publishers of Web documents could annotate their products on a common basis. DC can be viewed as a compromise between highly structured bibliographic data such as MAB and MARC records and simple annotations for automatic indexing as it is done by Web crawlers or locator services such as Lycos.

The DC element set consists of (with obvious semantics): TITLE, CREATOR, SUBJECT, DESCRIPTION, PUBLISHER, CONTRIBUTORS, DATE, TYPE, FORMAT, IDENTIFIER, SOURCE, LANGUAGE, RELATION, COVERAGE, and RIGHTS. All elements are optional and repeatable. DC annotations have been mainly used with HTML pages using the HTML META tag. An HTML version of the book by Hans J.C. v. Grimmelshausen as introduced in Section 1.1.3.1 could include annotations as listed in Figure 1.3.

```
<META NAME="DC.title"
      CONTENT="Der Aus dem Grab der Vergessenheit
               wieder erstandene Simplicissimus">
<META NAME="DC.creator"
      CONTENT="Grimmelshausen, Hans Jakob Christoph von">
<META NAME="DC.publisher"
      CONTENT="Felßecker, Johann Jonathan">
<META NAME="DC.type"
      CONTENT="Roman">
```

Figure 1.3: DC Example

Providing metadata for Web documents using the DC element has marked an important improvement. DC allows producers and publishers to annotate their Web documents in a simple way that does not require any specific expertise in cataloging. Yet, the category schema it provides can certainly not meet the high standards of cataloging formats such as MAB and MARC. From a cataloging point of view, DC describes a minimal set of document properties that are required in bibliographical records but do not suffice to catalog library stocks in a satisfactory way. There have been various proposals to extend the DC core element set, but as [20] argues correctly, it is hardly possible and even undesirable to define one single metadata standard that can accommodate all present and future requirements of all communities. As we will argue in more detail in the following sections, even within the highly specialized and restricted library community it has not yet been possible to specify one commonly agreed upon cataloging standard. On the contrary, the existing major cataloging formats have proven to be incompatible to an extent that makes automated format conversions a difficult and in some respects unresolved task, which is the status quo current library systems have to deal with.

1.1.3.4 Library Systems and Cataloging Standards

Library systems, e.g. OPACs and DLs, rely on well-defined data formats, such as MAB and MARC, in order to be able to interpret and manage bibliographic records correctly. In the library world according guidelines are provided by different sets of cataloging rules, e.g. RAK rules for MAB records and AACR rules for the MARC format. Cataloging rules were in the first place meant for catalogers who are using them for creating bibliographical records describing a work. The product of cataloging, however, is eventually a catalog or database of such records, managed by a library system and is usually meant for use by the public. Users of systems providing catalog (meta)data, such as the VD17 DL, expect library catalogs to be intuitively understandable, and this means there has to be a transparent structure with clearly outlined semantics. This is what cataloging formats and rules are providing. [78], [26] give a list of features a library system should offer:

- *Reliability*. Catalog users should be able to ascertain quickly and with certainty if the item is in the collection or not. In an unreliable catalog it may require potentially many trials before one can be sure.
- *Serendipity*. Taking search result sets as a starting point, browsing functions are essential, firstly because one doesn't always have precise search criteria, and secondly because chance findings are often valuable. Catalogs should therefore make related materials browsable[16]. Library systems can, for example, support browsing in the following ways:
  - provide alphabetical indexes of names, terms, titles, etc.
  - present result sets in more than one arrangement for the user to choose
  - make related publications accessible via hyperlinks
- *Depth*. This feature covers two aspects that are not exactly part of cataloging:
  - *a policy saying what materials or objects are subjected to cataloging*. Classically, these are (complete) books. However, a book may consist of several meaningful parts, each of which is representing a unit that might become the subject of a bibliographic record itself, e.g. proceedings volumes or periodicals. Readers will, in many cases, be interested and thus actually be looking for a chapter or parts of a book rather than the whole volume. If cataloging restricts itself to title page information, the catalog will be completely oblivious to all the constituent parts of books. One important case are "multipart publications" with individually titled volumes and how they are to be represented in a catalog: the focus of European cataloging (e.g. in MAB) is concentrating on the parts, whereas American catalogers (using MARC) have more often perceived multipart works as a whole.
  - *a concept for subject indexing*. Usually it does not suffice to simply assign a few subject terms and/or classification symbols to a document to nail down its content matter. The aim, on the other hand, should be to index every or the relevant part of all subjects that is actually dealt with in some part of the publication (cf. [26]).

Considering the above list of requirements and following [19] one can define essential properties for library systems. We will shortly characterize these properties and set them into relation with the services offered by the VD17 DL. The core principles for the definition of cataloging standards should optimally result in a catalog (interface) that:

- *makes the available materials reliably findable*: The user must be able to find with certainty each object in the collection according to specified characteristics that are normally in or on the object, e.g. titles and names of persons. The idea is that, for the majority of search scenarios, a small set of characteristics should suffice for finding a work if it is contained in the catalog.

  Book characteristics in the VD17 DL are realized through a set of 15 so-called *structure fields*. Each structure field pertains to a class of characteristics, e.g. there is a structure field "*title*" encompassing entries for main titles, title propers and secondary titles, and another structure field "*first person*" containing authors, translators, publishers, etc. All of these structure fields are searchable through the VD17 user interface, either separately or combined into complex queries using Boolean operators. If necessary, structure field search can also be used in combination with fulltext searches.

[16] The question what "related" means within a collection is, of course, not a trivial one to answer. The quality of a catalog, however, will depend to an important degree on its capability to inform users about affiliated and correlated works, e.g. concerning authors, subjects, secondary literature, etc.

- *differentiates among the dissimilar*: Each object should be described succinctly but in sufficient detail that it can be distinguished from any other object in the collection. Essentially, the criteria for describing objects are laid out in the cataloging format definition the catalog uses internally.

  The internal data representation format of the VD17 DL is MAB. Accordingly, each VD17 document possesses a unique identifier (stored in MAB category `001`), similar to a ISBD number. VD17 identifiers are organized in an according searchable structure field. Furthermore, VD17 records are such that a combination of the structure fields *first person*, *title*, *year* and *volume* will distinguish any document from all others within the VD17 DL.
- *relates and displays together objects that belong together*: The catalog must be able to provide the user contextual interrelations among its objects, e.g. the works of one and the same author, the available editions or versions of a work, or the components parts of a multipart work or a series.

  Contextual interrelations within the VD17 DL are partly immediately accessible, partly interrelations must be actively queried by the user. Immediately accessible are all hierarchical relations of multivolume works. Works by the same author, different editions or versions of works must be searched using the appropriate structure fields.
- *clearly displays that which is found*: If there are several or more entries in the catalog that meet the search criteria, the catalog should present them in a clear manner that facilitates selection by the user, e.g. using sorting capabilities.

  The VD17 DL allows the user to determine the way a result list is displayed in two ways. Firstly, large result sets are split into several result pages where the user may chose the number of hits shown per page. Secondly, the user can chose whether hits are sorted by first person, title or year (with the option that multivolume works are sorted consecutive by volume number).
- *makes the selected object accessible*: The user is interested in the quickest way of accessing the selected object. Card catalogs display only the location and call number; online catalogs, on the other hand, can lead directly from the record to use: they can facilitate the ordering or reserving of an item, or link one directly to the document online, if available.

  All documents of a VD17 result set listing are accessible directly via HTML links (each book title of the result list acts as a link to the respective document). Within each document page that consists of the bibliographical data of an individual work, there is also a list of links to related digitized key pages of this work.

In order to meet these requirements and provide services as outlined above, a computer system managing catalog data, such as the VD17 DL, must be able to interpret this data correctly. This means that the system depends on an adequate model of the bibliographical records it handles and accepts as an input. In the library world such models are inherent in cataloging formats and rules that provide the syntax and semantics for bibliographical records, MAB/RAK in the case of VD17. This poses no problem as long as all data such a system has to manage complies to one format alone, or at least it must be assured that there exists an algorithm for converting foreign data into the native format. Even with VD17, a project limited to German prints published in the 17th century, a significant portion of relevant works can be found abroad, e.g. in stocks of the LC. Yet LC records are coded in MARC/AACR that are based on a model that differs significantly from MAB/RAK.

Different cataloging standards are unfortunately using models that may be incompatible to an extent that makes it impossible to convert data from one standard to another without manual labor. As mentioned above, MAB/RAK and MARC/AACR records exhibit significant differences, e.g. concerning the treatment of multivolume works. Even within relatively homogeneous library standards such as the family of different MARC dialects, format conversions must not be seen as a trivial task. Even more so, records of more remote formats such as MAB and MARC cannot be easily aligned into a homogeneous data set (pertaining to a single data model). In fact, incompatibilities between MAB/RAK and MARC/AACR have grown to be a well-recognized deficit that poses a significant obstacle to exchanging data between German and Anglo-American libraries. For this reason, the VD17 partners had to refrain from importing LC records into the VD17 DL.

1.1.3.5 Harmonization of Cataloging Standards

Various international projects and conferences have been recently focusing on questions concerning the harmonization of cataloging standards. Harmonization in this context subsumes all efforts to either define a mapping between library standards (e.g. MAB/RAK and MARC/AACR), to propose an outline of (desirably minor) changes to these standards which would allow for such a mapping, or to specify a new standard that acts as a format unification. Finding solutions to these questions is a task that has been addressed in various international initiatives:

- *UseMARCON* (*Telematics for Libraries* Project No. 2054) [65] is a EU funded project that was aimed at developing a toolkit for converting bibliographical records in different MARC variants[17]. It was completed in 1997, resulting in a software package for managing UNIMARC, UKMARC and USMARC records. Yet, concerning bibliographical compatibility, UseMARCON does not go beyond the MARC world, and therefore cannot provide results that would be helpful for more complex tasks such as harmonizing MAB and MARC (cf. reuse_final_report.html]).
- The joint project "*Harmonization of Anglo-American Cataloguing Rules and Russian Cataloguing Rules (RCR)*" [60] was initiated in 1996 by the Russian Library Association (RBA) and OCLC. Financed by the Ministry of Culture (Russia) and OCLC, the project yielded a comparative table of 120 major differences between AACR and RCR. These results were the basis for a first list of recommendations open for discussion concerning possible changes of AACR and RCR in order to achieve a better foundation for exchanging bibliographic records between Russian and Anglo-American libraries. Although the project members could not offer a working solution for this problem, these efforts can be seen as an important step towards harmonization.
- REUSE [55], [28] and REUSE+ [27] are two consecutive projects carried through during the years 1995 – 1998, focusing on enhancing bibliographic compatibility among libraries using MAB/RAK and MARC/AACR. The project partners, members of the State and University Library of Göttingen (SUB Göttingen) and OCLC, studied format and rules differences that impede harmonization between the German and Anglo-American library standards. The goal of the REUSE projects was to identify these differences and to provide guidelines for an enhanced reuse (hence the project names) of MAB and MARC records. For the remainder of this section we will take a closer look at the REUSE efforts that offer a more detailed view on the background of and the difficulties encountered in harmonizing library standards.

As pointed out in [55], German academic libraries acquire more than 60% of their books abroad, and 90% of this material is provided by Anglo-American publishers. These books must be catalogued by German libraries according to their own cataloging formats and rules, mostly MAB/RAK. Furthermore, the bibliographic records of the American Library of Con-

[17] Throughout the world nearly 50 different MARC formats are currently in use, with 10 employed in the national libraries of European Community countries.

gress (LC) and its European counterpart, the British National Bibliography, are offered in most of the German library networks (ger. *Verbünde*). This accounts for the importance of manageable techniques for processing Anglo-American records, i.e. bibliographic data complying to MARC/AARC, in German library systems. However, reuse of these records without considerable manual and intellectual labor is appallingly low. This shortcoming has been examined in several studies (e.g. in [60]) from the perspective of the German library networks. Vice versa, the Library of Congress came to similar conclusions when trying to import German records into their local system (cf. [71]). The different studies showed as a common result that the impending question of finding possible ways to accommodate German records created on the basis of MAB/RAK with bibliographical records generated with MARC/AARC stated an unresolved problem in the international library world. It was against this background that in October 1995, OCLC and the State and University Library of Göttingen (SUB Göttingen), seat of the Regional Library Network for Central and Northern Germany (GBV), agreed to join in the project REUSE with the goal of enhancing international bibliographic compatibility among German and Anglo-American libraries. REUSE+, a continuation of project REUSE has then set its main focus on different types of representations of hierarchical bibliographic structures in MAB and MARC.

With MARC being a de facto standard exchange format on a global scale it seems quite natural to ask why it has not already replaced all other library standards. The answer to this question is not a purely historical one. Obviously, local developments and library initiatives in different cultural and national regions brought forward different cataloguing schemes, such as MAB and MARC. However, these formats save the same purpose within the same application area, i.e. defining a standard machine-readable format for bibliographical records. Thus, cataloging rules and formats *could* be essentially the same, taking minor differences aside, e.g. diverging names for identical concepts, etc. Unfortunately, this is not the case as we will demonstrate by referring to the relationship between MARC and the German MAB format. During REUSE and REUSE+, bibliographical records complying to MAB/RAK and MARC/AACR were systematically analyzed regarding their underlying formats and cataloging rules specifications. As a result, the REUSE working group defined a number of rule and format differences existing between the current versions of MAB/RAK and MARC/AARC. Major discrepancies were observed e.g. in the proper forms (ger. *Ansetzungsformen*) of names, titles and corporate bodies. Three separate classes of differences between the two library standards can be identified:

- *Conceptual differences*. MARC and MAB offer diverging understandings of what is meant e.g. by the terms “work” and “item”, as well as what may be accounted for as “corporate bodies”. These diverging understandings lead to structural differences and different cataloging policies.
- *Structural differences*. Multivolume and multipart editions are treated differently in MAB and MARC. While information on multivolumes and multiparts are integral parts of MAB records, MARC records do not know hierarchical structures. With MARC the work of establishing item records for individual volumes is done locally (in separate records maintained by libraries holding copies of the respective volumes), i.e. several records are needed for a complete description of a work. MAB records, on the other hand, contain the complete information on a work be it multipart or not.
- *Differences concerning cataloging rules*. RAK and AACR define diverging rules for cataloging corporate bodies in work titles. AACR regards more entities as corporate bodies, e.g. projects, expeditions and actions, ships and spacecraft, and buildings such as churches or castles. According to RAK these are not considered to be corporate bodies and will

therefore be treated differently (either they are omitted completely or fall into different categories).

The main result of REUSE has been the global observation that some of the most striking discrepancies between MAB/RAK and MARC/AACR are not caused by the respective cataloging rules or formats but by the differing underlying logical data models (cf. [28]). The REUSE working group came to the conclusion that the different data models inherent in MAB and MARC would require what they call a 'Common Object Model' for representing and a 'Common Functional Model' for processing bibliographical records (cf. also [59]). Although the REUSE working group does not provide such a model they stress its vital importance for further alignment between cataloging standards in general, and for MAB and MARC in particular. The foundations for designing such a data model and putting it into use cannot be found in the limited context of cataloging standards but in information science where research on semantic modeling has become a major field.

1.1.4 Digital Libraries and Semantic Modeling

The general idea behind a conceptual approach to DL using knowledge representations is to define a formal domain model, i.e. a formalization that covers the semantic content of a range of information items held in a DL. This conceptual model is on the one hand intended to provide an unambiguous view on the specific domain, e.g. differentiating between the different senses of homonyms such as 'kiwi', on the other hand it acts as a basis for knowledge processing tasks such as intelligent information retrieval. Research in this field has centered around conceptualizations called *formal ontologies*. Formal ontologies, roughly characterized as *'specifications of conceptualizations'* [40] have experienced an increasing popularity in the last decade manifesting a common trend within the IT community that stresses the importance of conceptual modeling over specific technologies and software architectures. The roots of ontology engineering lie on the one hand in knowledge representation techniques and mathematical logic, and on the other hand find their origins in the treatises of Plato and Aristotle on the philosophical discipline of *Ontology* where ontology engineering got its name from. The conscious choice of the terms formal *ontologies* and *ontology* engineering pays reference to these philosophical foundations and their rich vocabulary, and also stresses the grown awareness within the IT community that research in conceptual modeling is of prime importance for enabling advanced knowledge processing, such as intelligent information retrieval in DL.

	E/R	*OO-Design/UML*	*Prolog*	*EOS Ontology Engineering*
Primitives	entities, relations, attributes	Classes, attributes, methods	Facts, rules	concepts, ontologies
Active Components	integrity constraints	Methods	Rules	rules, laws
Purpose	Designing specific database schemas	implementation of software components	implementation of rule-based systems, e.g. deductive databases	designing a domain model determining the semantic mode of operation of knowledge processing (EOS) systems, e.g. DL
Formalism	relational Algebra	(–) [18]	predicate logic (Horn clauses)	Concept Theory

Table 1.1: Different Areas of Semantic Modeling

As indicated in Table 1.1, developing semantic domain models is not a new idea, e.g. a relational database schema is the result of a conceptualization that has been carried out using the Entity/Relationship (E/R) modeling technique. Other prominent examples for the employment of conceptualizations can be found in the area of object-oriented software design, where the structure and behavior of software components are modeled, e.g. using the Unified Modeling Language (UML), and in the field of logic programming, e.g. using Prolog for implementing rule-based systems. What sets ontology engineering apart from E/R, UML and logic programming is that these latter aim at and result in producing integral parts of software systems (the schema materialized in the system tables of a particular database, particular software components and systems), while the outcome of an ontology engineering process remains conceptual, i.e. an application independent representation of the domain in the form of an ontology. Such representations can take on different forms, e.g. using new technologies such as XML and XML-based languages, e.g. RDF(S) and DAML+OIL, or hybrid formats that include predicate logic or frame logic, e.g. KIF. Regardless of the specific representation format, the purpose of formal ontologies is to provide an application independent conceptual model of a given domain of interest.

This already mentions one of the benefits of formal ontologies, namely that they are not designed with respect to particular systems or application areas but remain on the conceptual level. In this way, ontologies can be exchanged among heterogeneous systems that may also use them differently, e.g. a DL and a workflow management system can operate on the basis of the same ‘enterprise ontology’, yet utilize it to completely different ends. The knowledge that is represented in a formal ontology, e.g. the above mentioned enterprise ontology, is an explicit conceptual model that can be used by both (and more) systems, as opposed to a situation where the different systems hold this knowledge implicitly in their algorithmical implementation. Using explicit and interchangeable conceptualizations in the form of ontologies thus keeps software systems more flexible, as it is fairly easy to alter the formal representation of an ontology (e.g. an XML file) that is passed to the systems as an input, than a costly system redesign that is necessary if these changes pertain to their internal implementation.

This also implies that the semantic knowledge contained in formal ontologies must be interpreted by these systems. In order to enable correct semantic interpretation, these systems must not only be able to parse the specific ontology representation but also need a formal ba-

[18] No satisfactory and uniquely accepted formalism for OO design has yet evolved.

sis for such an interpretation. To the best of our knowledge, an adequate formalism for expressing on the one hand conceptualized domain knowledge and on the other hand the semantic implications for utilizing this knowledge in a unified way has not yet been proposed. This leads to the following objectives for the EOS approach as presented in this thesis:

- *Motivating and presenting a formalism and modeling technique apt to provide a semantical basis for ontologies*. The formalism we have developed in that respect is called *Concept Theory* and builds upon a unicategorical view on conceptual modeling. Concept Theory is based on our notion of *formal concepts* which will be explained in detail in Chapter 2, along with a graphical notation for ontology design. Formal ontologies that are founded in Concept Theory are called *EOS ontologies*.
- *Incorporating application semantics into this formalism, which allows knowledge processing systems to interpret EOS ontologies in a unified way*. This applies in particular to the active components of EOS ontologies, i.e. concepts defining rules and epistemological laws. Thus, EOS ontologies also model the mode of operation of knowledge processing systems. This approach is also new to the ontology engineering community and will be the central topic of Chapter 3.
- *Proposing a general architecture for knowledge processing systems that interpret EOS ontologies*. Such systems are called *EOS systems*. Basically, any information system such as the VD17 DL, search engines, etc. can be regarded as an EOS system, provided it can interpret EOS ontologies according to Concept Theory. The basic architecture of EOS systems is introduced in the discussion on automated knowledge processing in Chapter 3 while further application and implementation details are presented in Chapter 4.
- *Translating EOS ontologies into an easy-to-parse and convenient exchange format*. In this context we will show how EOS ontologies can be expressed in XML compliant to a specific EOS DTD.
- *Storing and querying EOS ontologies efficiently using database techniques*. Here, we will refer to relational database systems and how they can be employed for managing EOS ontologies. In particular, a database schema and a mechanism for accordingly indexing EOS ontologies will be presented.
- *Evaluating and discussing the proposed EOS approach*. This is done by referring to a specific application scenario for EOS systems and how knowledge processing tasks may be defined and put to work in this practical example. Furthermore, we will flesh out the application scenario by discussing a prototypical implementation.

1.2 Orientation

The following reflections give an outline on how different fields of human activity are operating with the term '*knowledge*'. In the course of this synopsis we will briefly shed light on philosophical, business and information science perspectives on knowledge. These very different disciplines all have their own objectives that, evidently, influence the way they approach knowledge. Yet, despite their contrasting motivations, it is only natural that philosophers, scientists, and practitioners find many parallels, analogous questions and arguments that mutually build upon each other when analyzing knowledge. Philosophy, being the original discipline of rational investigation per se, examines the essential nature of its subjects of analysis, thus providing a valuable and mature history of discourse, which is especially true with treatises on knowledge[19]. This preliminary work serves as a theoretical basis for adequate

[19] How knowledge should be ultimately characterized is an unsettled matter in Philosophy to this day. The most influential treatises on the subject have been given by Plato and his scholar Aristotle. Knowledge about some thing in the world, ac-

ways of making use of knowledge in application areas such as organizational knowledge management, or computerized knowledge representation and processing.

Sociologically, knowledge is nowadays being considered a key factor of primal importance. Individual and economical success depend, more than ever, on the ability to acquire, organize, annotate, retrieve, interpret and utilize knowledge in a fast and efficient manner. Enabling technologies brought forth by engineering and information science, therefore, experienced an unparalleled growth as an immediate consequence to the increasing demand for new and better ways of dealing with large quantities of correlated information. Advanced database techniques, especially improved indexing methods, coupled with accomplishments in the fields of e.g. Data Warehousing, Online Analytical Processing (OLAP) and Information Retrieval (IR) help organizing and searching information. Yet, searchable *information* in itself may not be incautiously equated with *knowledge*. All of the techniques mentioned treat information on the basis of a specific semantic model. Databases, for example, store information according to a predefined schema that, on its part, enables applications built on top of them to correctly interpret this information. On equal terms, a thorough understanding of the problem domain, integrated explicitly or implicitly into the computer system, is a necessary prerequisite to any information processing task. Once again, the specification of the algorithmic solutions to these tasks is, too, based on knowledge about problem-solving techniques in the particular area.

These first, exploratory considerations have shown that knowledge and its utilization touch on practically every aspect of modern society, in particular, it has become a valuable economical resource. Starting from this perspective we will further discuss the notion of knowledge within the scope of this thesis. Section 1.2.1 is dealing with an approximation to a working hypothesis of knowledge applicable to practical usage. How knowledge is being approached in organizational knowledge management and arising consequences for knowledge representation and processing will be the subject matter of section 1.2.2. In section 1.2.3 we will then briefly sketch the current status of computerized utilization of knowledge.

1.2.1 What is Knowledge?

Asking the question 'What is knowledge?' might at a first glance seem to be a purely philosophical enterprise. However, even on a day-to-day basis, humans constantly rely on knowledge of their environment, their physical and mental skills to perform the tasks their daily life calls for. Clearly, it is of utmost importance for us to acquire knowledge and skills in order to master anything, from brushing our teeth to conceiving very abstract scientific theories. Knowledge is being articulated in various forms, in written or spoken language, graphics, diagrams, etc., and communicated to others, which is the basis of all learning. In short, acquiring and using knowledge is an essential factor of human life, not just an abstract philosophical inquiry.

Surely, this thesis cannot and does not try to answer the metaphysical question about the nature of knowledge, a problem which has not been satisfactorily solved since the beginnings of Western Philosophy in Ancient Greek – despite its prominent status at the center of philosophical debates ever since. Our focus is application oriented, and therefore we will concentrate on theoretical and technical aspects of *utilizing* knowledge in computer systems. Hence we will certainly not define knowledge but, on the other hand, we must not refrain from clarifying the *notion* of knowledge in our field of research that encompasses *knowledge representation* as well as *knowledge processing*.

cording to Aristotle [2], implies the ability to produce a precise definition of that thing. Plato, too, referred to the notion of definition when he specified knowledge as "justified true belief" [11].

Knowledge representation treats knowledge as a resource that can be communicated to third parties, humans and computer systems alike. Thus, research in this field is concerned with encoding and decoding knowledge in different ways. As a simple example, knowledge about the topic 'knowledge representation' may be encoded in electronic form, e.g. as an XML file. Physically, this XML file is binary *data* and from this point of view equivalent to any other file containing data of any kind, e.g. text, audio, or video, possibly even an arbitrary binary sequence. Syntactically, the XML file contains structured *information* that is expressed in a specific format, namely XML, and text in some artificial or natural language, as the content of XML files is interpreted as textual information. Semantically, the same XML file carries *knowledge* on a particular subject, 'knowledge representation' in this example, that can be understood by humans and processed by computer systems. Consequently, one and the same XML file may be regarded as data ('something'), information ('something containing semantics'), or knowledge ('semantics themselves') depending on whether one adopts a physical, syntactical, or semantic perspective.

Knowledge processing comprises techniques and methodologies for computerized management of knowledge, or more precisely, management of data bearing knowledge, like the XML file mentioned above. The hypothesis of knowledge processing is that the semantic content of data can be analyzed by a computer system if the syntactical features are exploited in an appropriate manner. On this basis, a resulting knowledge processing system should then *behave as if* it understands the semantic content of its input data. Naturally, highly structured data, whose syntactical composition is known in advance and can thus be interpreted accordingly, offers better preconditions for advanced processing techniques than unspecific data like plain text files in natural language. In any case, the computer system will be designed according to presumptions about the format and range of content of the data it should process, i.e. background, or *ontological knowledge*, will necessarily be an important part of the implementation of the system, be it explicitly or implicitly. Secondly, the data has to be processed in a way that a desired result, e.g. the answer to a query about its content, can be computed. Therefore application semantics, or *epistemological knowledge*, have to be integrated into the system. Any knowledge processing system, consequently, is the result of carefully designing and implementing software components according to scientific theories, practical guidelines, and expertise in the application area.

We identified two different *kinds* of knowledge, ontological and epistemological knowledge, that are essentially determining the modalities of knowledge representation and the behavior of knowledge processing systems:

- Following philosophical theories, taking an ontological perspective means to analyze parts or all of reality in order to determine its constituents and inner correlations. Evidently, classification and explication of the entities identified during this analysis is of predominant interest when examining reality. Studying the world ontologically, therefore, is to categorize it according to an elaborate metaphysical system, which is very similar to formalizing knowledge according to a predefined methodology. Hence, the basic considerations of ontological theories and formalizing knowledge are closely coupled, but unfortunately this fact is hardly ever recognized, let alone exploited to benefit knowledge representation formalisms. Against this background it is our expressed concern to incorporate a thorough metaphysical foundation into our EOS framework to overcome these deficiencies and to pave the path for its epistemological applicability.
- Regarding the world epistemologically, then, is to look into ways of acquiring and using ontological knowledge, which presupposes an understanding of the nature of knowledge itself. Different epistemological theories in Philosophy have evolved, most of them shar-

ing as a common denominator that the structure of knowledge is characterized as something strong enough to account for objective truths about entities of reality [70]. Knowledge, thus, relies on an ontological basis and modes for justifying true beliefs, just like mathematical deduction is based on a set of primitives and inference rules.

We will frequently refer to the basic distinction between ontological and epistemological aspects of knowledge throughout this thesis in order to motivate and explain practical as well as technical details.

1.2.2 Knowledge Management

It comes as no surprise that, with knowledge being recognized as a value of its own, knowledge management has nowadays become a vital economic constituent for organizations as commercial success depends on a proper understanding of business and market structures (cf. [79]). This includes internal processes leading to increased productivity and innovation, as well as external processes, e.g. concerning market perspectives, and interactive processes among organizations and their customers, as in e-commerce environments. All of these processes on the one hand produce new information, while on the other hand they require knowledge in order to be mastered successfully. Knowledge, here, is seen in the context of business rules [73] and the organizational memory that comprises the intellectual potential of employees (i.e. skills, experience, expertise), document archives (electronic as well as print media) and all further information relevant to the organization (e.g. inherent in workflow processes) [50]. Therefore, knowledge management tasks concentrate on capturing and organizing these semantics for further usage. Predominant problems in this area are how to implement ways to acquire and handle often large amounts of information, particularly implicit knowledge, and how to incorporate it into an organization's workflows.

The knowledge management business model thus understands knowledge as a valuable resource that should be exploited in order to supplement the success of an organization. Hence knowledge management offers no explicit theory of knowledge nor does it promote a particular methodology for representing knowledge. Rather, it provides guidelines for identifying and using relevant information (about and within business processes) and its actual and potential benefits for an organization. Once the explorative labor of gathering this fundamental information has been carried out, it can be formalized according to some knowledge representation model and subsequently forged for further use. This clearly establishes a close connection between organizational management and knowledge representation techniques that are used as an indispensable auxiliary for efficiently implementing a knowledge environment. In short, representation methods provide the means for computer supported exploitation and utilization of knowledge, thereby enabling successful management of huge amounts of information in the first place.

It is a common practice to organize formalized information in databases where it is stored consistent with a suitable database schema that allows for a syntactical interpretation of the data. Databases provide secure persistent storage and efficient information retrieval techniques, and are as such a valuable aide, if not an essential prerequisite for intelligent searching, knowledge bases, data warehouses and OLAP systems, or more general, for knowledge processing. Knowledge processing techniques are using the syntactical features of knowledge representations in order to make domain knowledge and its implications available to human users. Advanced tasks that can be solved by knowledge processing systems are automatic deduction of new knowledge from information at hand, as well as finding yet undiagnosed interdependencies and correlations among domain entities. Knowledge management on the technical side, thus, heavily relies on such systems for ensuring highest benefits from organizational knowledge, once it has been acquired.

As already mentioned above, organizational knowledge management is closely connected to knowledge representation and processing techniques. The information identified and accumulated during explorative phases of knowledge management can on its part be regarded as an input to knowledge representation methods. Resulting formalizations, then, are the basis of knowledge processing, whose conclusions may be used, again, to support knowledge management processes. Knowledge management therefore displays ontological as well as epistemological requirements that have to be met by supportive computerized systems incorporating knowledge representation and processing.

1.2.3 Knowledge Representation and Processing

Knowledge representation and processing have become major fields of interdisciplinary research. [43] As outlined in the previous sections, human knowledge is being identified, acquired and analyzed, then formalized using various representation models. Resulting representations lay the groundwork for processes that incorporate knowledge, be it on a general level (e.g. for supporting workflows within organizations) or in practical applications (e.g. for assisting specific workflow components). At present, the predominant means for representing knowledge in knowledge processing systems are *formal ontologies* [41]. But despite their widespread use the notion of formal ontologies remains blurred [42]. The commonly accepted but nevertheless subsidiary and rather unspecific definition of a formal ontology as a '*specification of a conceptualization*' [39] does not account for the semantic implications of ontologies, although the object of such conceptualizations is supposed to be knowledge. As an unfortunate consequence, the predicate 'ontology' is attributed to a large variety of formalizations that differ greatly in form and their expressive power. What is obviously lacking is an elaborate formalism for the design of ontologies to account for a satisfying theoretical fundament.

Since ontologies are knowledge representations, explicitly formalized bodies of human knowledge, they can be as such communicated across computer systems. In particular, formal ontologies may be used as an input to knowledge processing systems. Yet, how knowledge held within an ontology is utilized by a computer system to perform its tasks is still part of the algorithmical implementation of the respective system. The actual knowledge processing tasks, just like all other internal processes, thus typically remain encapsulated by the system, that in this respect acts as a black box, abstracting from the details of its realization. While hiding *implementation* particulars from outside components is a well-established and advantageous technique, concealing the pursued application *semantics* when processing knowledge comes close to self-contradiction when *knowledge* is at the center of the enterprise of such systems.

It is an important design decision for any knowledge processing system where to draw the line between internally realized application semantics that are an integral part of its implementation, and input data to the system, i.e. its parameters that have to be interpreted according to application semantics. Note that this decision is certainly not one about the syntactical format of input data, e.g. formal ontologies, but one about the scope this data covers semantically. With respect to formal ontologies, different languages like DAML+OIL, RDFS, etc. for expressing their ontological content are conceivable (cf. Section 4.3), and a respective knowledge processing system must, of course, be able to interpret (at least) one representation format. Yet, the ability to understand an ontology syntactically does not tackle the original question on what kind of knowledge the ontology carries semantically. If it is restricted to ontological knowledge alone (and no other input is given to the system), then application semantics are the sole responsibility of the knowledge processing system. The more epistemological knowledge, e.g. on processing and inference rules, a system can accept, the better and more flexible it can be put to use for assorted and complex tasks.

1.3 EOS Ontologies in a Nutshell

At the core of EOS is Concept Theory, a formalism that provides the syntax and accompanying semantics that are necessary for designing EOS ontologies, highly structured knowledge representations. Concept Theory is the necessary foundation of the EOS approach and offers a uniform modeling paradigm for formal ontologies. As such it provides clear semantics and a corresponding definition for formal ontologies.

Central to Concept Theory is the notion of *formal concepts*. A formal concept in terms of Concept Theory represents a definition of an *existent*, i.e. an entity or relation of a domain of interest that has to be modeled by a formal ontology. A complete EOS ontology, then, consists of a set of formal concepts complying to the structural rules as laid out by Concept Theory. These syntactical principles define, firstly, a hierarchical structure among ontology concepts that mirrors specialization and generalization among existents. Secondly, Concept Theory determines composition patterns of concepts along the specialization hierarchy. From this perspective, an EOS ontology can be thought of as a directed graph of ordered concepts.

Concepts are capable of representing domain objects as well as relationships, which allows to regard them on the one hand as entities that are subject to relations, and on the other hand as relations themselves. As an example, *ownership* is a relation between some *property* and its *owner*. In Concept Theory it is feasible to describe such a relation with a concept OWNERSHIP that relates two other concepts, PROPERTY and OWNER. The concept OWNERSHIP may then in turn be subject to other relations such as a CONTRACT_OF_SALE.

Another specialty of formal concepts is that they are used to describe general domain existents, classes or categories, such as OWNERSHIP, as well as their individual concretizations, instances or occurrences[20], like the specific ownership that exists between a particular person BARRY and the specific HOUSE_OF_BARRY this individual owns. Hence, Concept Theory offers a seamless representation mechanism for both, a definition of the general aspects of a domain, i.e. entities and relations, and concrete instances thereof.

Concept Theory is outlined in Chapter 2 with reference to the current state of research concerning knowledge representation in information science, which is sketched in Section 2.1. An introduction to the theoretical groundwork of the EOS approach is given in Section 2.1.1, where two main categories of domain existents, *universals* and *particulars*, are presented. Section 2.1.2 subsequently discusses the notion of formal ontologies in information science. Based on the results of this treatise, all of Section 2.2 is dedicated to a thorough introduction to the syntactical and semantical features of formal concepts. Section 2.2.1 covers the essential syntax of concepts, while Sections 2.2.2 and 2.2.3 discuss sets and kinds of concepts. Significant sets of concepts are the Universe of Concepts that represents the totality of all intelligible concepts, domains and formal ontologies themselves. Different kinds of concepts can be identified by examining what kinds of existents, universals or particulars, they are describing. The semantics of concepts are motivated and formally defined in Section 2.2.4, which eventually leads to an exact definition of formal ontologies. Ontological semantics are grounded in the properties of the relation *specialization* which is represented by the concept **ISA**. Section 2.2.4 is thus a concise treatment of the syntactical properties and the ensuant semantical im-

[20] Occurrences are specializations of existents, e.g. the species *larch* is an occurrence of *tree,* and an individual larch that is part of a forest is an occurrence of the species *larch*. As these examples indicate, occurrences can be both, species and instances.

plications of this concept. A graphical notation for representing EOS concepts is introduced in 2.3. Finally, Section 2.4 concludes Chapter 2 by presenting an extended example of an EOS ontology.

1.4 The Enterprise of EOS Epistemology

Introducing epistemology to the EOS approach means integrating knowledge processing primitives into the formal body of Concept Theory. This project has far-reaching consequences concerning the understanding of knowledge processing systems. Currently, knowledge processing systems do not allow their internal processes to be subject to conceptional changes without redesigning vital parts of their implementations, which is understandable considering that such systems are commonly conceived and realized for very specific purposes, e.g. for parsing natural language texts, for managing Web pages, or for providing the services of a DL. The EOS approach, on the other hand, pursues a more general goal. An EOS system must be thought of as a versatile machine the actual behavior of which can be defined (and altered) at any time and on a declarative, formal level, i.e. on the level of Concept Theory. Thus, EOS systems do not only accept knowledge representations, formal ontologies, that contain specific domain entities and their interrelations, but also receive formalized epistemic entities that define the epistemological processes specifying the system behavior in accordance with the respective knowledge representation. In other words, an EOS system will accept both, ontological *and* epistemological knowledge.

Epistemological elements of Concept Theory will be discussed in Chapter 3. Section 3.1 starts with an introduction to knowledge processing from the different perspectives of philosophy and information science. Points made there will result into the specification on the EOS framework for knowledge processing systems. This framework defines the general architecture for EOS systems and defines three main epistemological tasks, namely knowledge acquisition (accepting and classifying input information expressed through concepts), knowledge generation (producing new concepts from existing ones) and knowledge retrieval (by querying an EOS ontology). Section 3.2 sets off from this theoretical outset and introduces EOS rules and laws for defining epistemological processes. Rules and logical conditions in Concept Theory are treated in Sections 3.2.1 and 3.2.2. Conceptual laws for epistemological tasks are laid out in Section 3.2.3. In particular, Section 3.2.3.1 presents a generic algorithm for EOS systems that shows the overall structure of EOS knowledge processing. Subsequently, Section 3.2.3.2 discusses the law **ACQUIRE** and its employment in the knowledge acquisition process, Section 3.2.3.3 introduces the law **GENERATE** that used in knowledge generation processes, and lastly Section 3.2.3.4 covers **QUERY**, the law for defining query processing in knowledge retrieval.

1.5 Employment of EOS Ontologies

Chapter 4 presents an application scenario for EOS systems used for classifying Web documents. This discussion will focus on two main issues, the practical employment of EOS ontologies in knowledge processing tasks, and different representation formats for such ontologies. On the one hand, ontologies may be communicated to, from and across EOS systems, i.e. an adequate representation format for exchanging EOS ontologies is needed. On the other hand, EOS systems must persistently store ontologies and provide mechanisms for updating and querying this information. In order to implement these services in an effective way, it is feasible to use database technology. As a prerequisite, an internal representation format for EOS ontologies in the form of a database schema is needed.

Section 4.1 describes the example application scenario for EOS systems where Web documents are being classified and semantically indexed. Several ways of making use of document markup in information extraction are presented. In this context, we will give an outline on how markup elements may be mapped to ontology concepts in order to support the knowledge acquisition procedure. In Section 4.2 we are presenting how the exemplary EOS system implements epistemological processes in the document classification scenario. This includes integrating conceptual representations of documents and their intellectual content (knowledge acquisition), as well as querying this ontological information (knowledge retrieval), with a special focus on how to optimize return values by semantic query rewriting (knowledge generation). In this context Subsequently, in Section 4.3 we will examine possible representation formats for ontologies, a discussion that leads to our preferred format for EOS ontologies that is based on XML. Section 4.4 characterizes how EOS ontologies can be effectively managed using database technology, and 4.5 concludes Chapter 4 by presenting details of an EOS prototype implementation.

This overview of the EOS approach completes the reader's guide that we used to motivate the subject matter of this thesis and the agenda it is committed to. The subsequent chapters will now develop a detailed view on the different aspects of EOS that form the groundwork needed to put an epistemological ontology-driven system for knowledge processing to work.

Chapter 2 EOS Ontologies

The aim of this chapter is to give a concise explanation of the theoretical foundation of the EOS approach and subsequently develop the ontology formalism we designed for our approach. At the heart of this formalism that we call *Concept Theory* is the notion of *concepts*, formalizations of existents, i.e. entities of and facts about a given domain of interest, or, generally speaking, about reality. Concepts are simple but powerful constructs carrying clearly defined semantics as provided by Concept Theory. As such they are an adequate means for developing knowledge representations. In particular, Concept Theory offers a formalism that is versatile enough for a satisfactory clarification of the semantics of formal ontologies, which has remained a pressing, yet unresolved problem since the advent of ontology engineering in the late 1980ies.

2.1 Knowledge Representation

In order to establish a well-founded position apt to motivate and describe a formalism for ontologies, this section will lay out our perspective on knowledge representation. We will take a look at the philosophical foundations of ontologies and use arguments found here for specifying a theoretical basis for formal ontologies. Coming from this perspective we will subsequently stress canonical requirements for formally representing knowledge in a way that it represents a substantial input for knowledge processing.

Knowledge Representation aims at encoding human knowledge in such a way that this knowledge is not only understandable by humans but can also be used by computer systems. A successful representation of some piece of knowledge must, therefore, cause the system using this knowledge *behave* as if it knows it. The surmise that this goal can be achieved has been made popular as the *Knowledge Representation Hypothesis*:

> "Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behavior that manifests that knowledge." [68]

During the past two decades the notion of knowledge in the AI community experienced a notable shift from a primarily functional view that was focused on modeling human rationality towards a new perspective that put emphasis on modeling systems in the world [16], [41]. We call the former understanding of knowledge the *narrow view* as it is foremost task-driven, i.e. solely knowledge relevant to a specific, pre-defined problem is taken into consideration. Opposed to this notion is the *general view* of the latter approach where knowledge is expected to describe not only details for particular tasks but an entire problem *domain*. Thus the general view is closely related to the objective reality of the problem domain and in itself independent of possible applications. It therefore gives a systemic perspective on an application area by providing an adequate formal description of the corresponding domain. In this thesis we are concentrating on this general view, moreover, the EOS framework is precisely aimed at describing systems that operate on this understanding of knowledge that has also been the underlying motivation for developing highly sophisticated knowledge representations such as formal ontologies.

Knowledge representations are generally applied to tasks where computer systems need additional input (domain knowledge) in order to adequately process data, e.g. texts in natural language. The system is regarded as possessing knowledge about a problem domain via its formal representation that it accepts as an input. Knowledge in this sense consists of data while application logics, i.e. knowledge on how this data may be used, is usually not part of such knowledge representations. The manner in which this data is processed by a computer system mainly relies on its own design and implementation, not on processing instructions included in the knowledge representation itself. Thus knowledge representations actually address structural aspects of a domain while the definition of problem-solving processes that use elements of these representations are mostly part of the algorithmic implementation of the system. Yet, at this point we are still lacking the proper arguments concerning what we can and should demand from such knowledge representations. The following sections are therefore intended as a brief outline of a feasible approach to knowledge representation committed to the general view.

2.1.1 Philosophical Foundations

The foundations for the formal models of basically any means of knowledge representation have been laid out in a long tradition of writings in *Metaphysics*, a branch of Philosophy prominent since the time of Aristotle (384 – 322 BC). Metaphysics, roughly speaking, is concerned with the *nature of reality*, and has therefore developed a complex apparatus for examining "being *qua* being" [2]. Following this initial definition of their enterprise metaphysicians have ever since elaborated diverse theories on structuring and explaining the world. As the subject matter of such theories is the nature of reality, a central task they have to address is that of defining *existents* ("things that exist"), the objects they encounter when examining reality. In other terms, given a specific existent a metaphysical theory must be able to produce a satisfactory answer for the question "What is it?". According to Aristotle, the *kinds*, or *categories*, under which a thing falls enable us to say what the thing in question is. This yields immediately to further considerations like "What categories are there?" and "How are categories related to the existents they are attributed to?". Thus categories and their application are fundamental constituents of metaphysic theories and can therefore be found in basically any metaphysical treatise, ranging from Aristotle's catalogue of ten categories[21] to Immanuel Kant's table of twelve categories[22] on to modern category theories (a comprehensive survey on category theory is given in [69], [68]).

Predominant in modern Metaphysics is the distinction between *particulars* and *universals* that form the two fundamental categories of existents (cf. [18], [52]). The category of particulars includes what one typically thinks of as 'things' – individual objects like human beings, animals, plants and inanimate bodies or events. Aside from these *material* objects most theories also know *immaterial* particulars like human minds. A further distinction is drawn between *concrete* particulars that exist in space/time and *abstract* particulars like numbers that do not. Universals on the other hand are commonly divided into two broad groups: *qualities* (or *unary properties, attributes*) like an object's mass or color, and polyadic *relations* (or *relational properties*) such as the dyadic relation *longer-than*. Universals may be exemplified, or *instantiated*, by particulars, i.e. universals are construed as repeatable existents. In fact at any given time, one and the same universal can be instantiated by several different spatially discontinuous particulars. Accordingly several distinct automobiles can simultaneously exhibit the same shape and color, as well as the same virtue may be exemplified by different people. A schematic summary of the notion of universals and particulars is shown in Table 2.1.

Existents					
Universals		*Particulars*			
Qualities (e.g. mass, color)	Relations (e.g. longer-than)	abstract (e.g. numbers)	concrete (e.g. people)	material (e.g. tables)	immaterial (e.g. human minds)

Table 2.1: Outline of Metaphysical Realism

Metaphysics cannot be regarded as a uniform discipline or science. As sketched above many philosophers have suggested various and very different metaphysical models for explaining reality, some even denying the existence of universals, or particulars, respectively. Common to these approaches, nevertheless, is the manner in which metaphysical questions are advanced. Examining metaphysical theories we can ordinarily identify

- a set of fundamental categories
- a commitment to some specific model on how to apply these categories

The set of fundamental categories is traditionally called *ontology* (cf. introduction to Chapter 1). Thus, an ontology claims a certain view on reality, e.g. metaphysical Realism assumes that reality consists of both, universals and particulars, while metaphysical nominalism denies the existence of universals. Consequently, realists and nominalists will develop quite different ontologies for explaining reality in their own ways. The arguments for the internal structures of their ontologies are drawn from their respective *ontological commitments*, i.e. the specific view on reality underlying, e.g. realism or nominalism.

In summary, an ontology is a means for *classifying* and *explaining* all actually occurring and intelligible existents. In order to do so, each ontology provides its own set of fundamen-

[21] Aristotle's categories are: Substance, Quality, Quantity, Relation, Activity, Passivity, Having, Situatedness, Spatiality and Temporality

[22] Kant's categories are: Quantity (Unity, Plurality, Totality), Quality (Reality, Negation, Limitation), Relation (Inherence, Causality, Community), Modality (Possibility, Existence, Necessity)

tal categories and an accompanying theoretical model. Philosophers traditionally stress the importance of the underlying model as they use ontologies predominantly for *explaining* reality, i.e. any resulting classification of existents is usually regarded as a (welcome) side-effect. Knowledge representation on the other hand tries to meet another claim, namely providing coherent *descriptions* of entities and their correlations that may be used as an input for computerized systems. Accordingly, classification and other means for facilitating automated processing and reasoning play a major role in knowledge representation.

For our discussion of methods for and systems using knowledge representation we will adopt the holistic perspective and vocabulary of metaphysical Realism as laid out in this section. A simple graphical view on metaphysical Realism is depicted in Figure 2.1. Reality, i.e. the entirety of all existents, is divided into universals and particulars. This view will serve as a basis for developing and explicating our approach to knowledge representation and, specifically, formal ontologies.

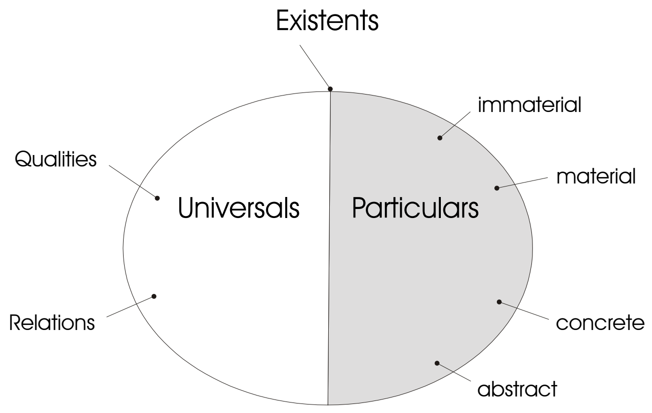


Figure 2.1: Metaphysical Realism

Once human knowledge is formalized and made part of some knowledge representation it is necessary to determine how this knowledge may be used, e.g. in order to infer new knowledge or to perform other tasks that rely on the knowledge held in ontologies. Ontologies are concerned with the nature of reality, i.e. the nature of *objects of knowledge*, and are therefore static in nature. Using knowledge, on the other hand, involves dynamic processes that use and create such objects of knowledge. This observation is examined in philosophical theories of knowledge that study the *nature of knowledge* itself.

Ontological conceptions, such as metaphysical realism, serve as an essentially required basis for theories of knowledge as ontologies give a notion of being and truth, which are fundamental to knowledge. Theories of knowledge have a long tradition in philosophy, in fact, an entire philosophical discipline, *Epistemology*, is dedicated solely to the study of knowledge. Epistemology focuses on questions about the nature of (human) knowledge, i.e. what is knowledge and what can be known. Modern analytical philosophy stresses the propositional structure of knowledge and uses mathematical logic for arguing about propositions. The general idea is that true propositions describe situations in the world, which presupposes objective truth that may be attributed to propositions. In order to turn a true proposition into knowledge its truth has to be proven, or justified (see also Section 3.1.1). This leads to the most prominent definition of knowledge as ‘justified true belief’ that has been given by the Greek philosopher Plato and is still at the center of current debates.

The distinction between objects of knowledge and knowledge itself as demonstrated in Metaphysics and Epistemology may act as a directional role for knowledge representation. Any knowledge representation will necessarily describe solely *objects* of knowledge but the way knowledge is encoded is crucial for its employment in epistemological processes. Just as each epistemic conception is (implicitly or explicitly) built upon its own ontological foundation, so does any knowledge processing system rely on the expressiveness of the knowledge representation it receives as a basis for the tasks it has been designed to carry out. Knowledge representation, therefore, must meet a high standard concerning quality and usability. At present, the most promising examples of high quality knowledge representations used for knowledge processing are formal ontologies [29].

2.1.2 Formal Ontologies in Information Science

Knowledge representations try to mirror the logics of a domain of interest, similarly to a metaphysical ontology that aims at giving an account of the nature of reality in general [67]. Thus it comes as no surprise that metaphysical terminology made its entrance to theories on knowledge representation as the scope and demands in this field of research grew more sophisticated. The aspired goal is to find methods for designing knowledge representations that are on the one hand thoroughly independent of a specific application, i.e. on a very high level of abstraction, and on the other hand a precise and complete, i.e. applicable model of a given domain. This goal and accompanying questions are addressed in information science where *formal ontologies* have been introduced as promising candidates for bridging the gap between general representations and the concise employment of knowledge in computer systems. Formal ontologies comprise abstract knowledge representations that convey a shared understanding of a domain of interest [72]. Necessarily, a formal ontology will entail its own worldview with respect to the domain, defining the existents involved. This is referred to as conceptualization and, accordingly, formal ontologies are often characterized as *specifications of conceptualizations* [40].

Despite their widespread use – formal ontologies are frequently cited as the key technology used to describe the semantics of information exchange [32] – the common understanding of ontologies is still an unsettled matter. Such conceptualizations take on a variety of forms and therefore differ greatly in structure and formality, ranging from highly informal ones expressed loosely in natural language, to rigorously formal ones using precisely defined terms with formal semantics, theorems and proofs of such properties as soundness and completeness [72]. This formal diversity stems from the unfortunate fact that there is no agreed meaning of the term ‘ontology’, even within the ontology engineering community [42].

For our discussion it will suffice to follow the general understanding of a formal ontology as an agreement about a shared conceptualization for a domain of interest, aimed at knowledge sharing and reuse. The quality and usability of an ontology therefore depends on how, regarding adequacy and completeness, it represents the target domain. Yet, as already indicated, there is no commonly agreed upon understanding of ontologies. The reason for this deficiency lies in the, at times, careless usage of the term ‘ontology’. Unfortunately, it has become distressingly fashionable among scientists to call *any* knowledge representation an ontology, hence the vast diversity. This obliterates the fact that formal ontologies are conceptualizations that have been designed for a specific *purpose*, namely to enable and support sophisticated knowledge processing. As an ontology describes a domain of interest its usefulness strongly depends on how the knowledge it encodes can be utilized to perform knowledge tasks such as inference based on the knowledge it provides. This means that an ontology needs to be epistemologically valuable with respect to the knowledge it represents which leads to three basic requirements for formal ontologies:

- *Generality*. A formal ontology should exhibit a high degree of generality concerning its possible uses. Ontologies comprise knowledge about domains and should therefore be designed to be task-independent.
- *Significance*. All significant aspects central to a domain have to be captured by a corresponding formal ontology. This requirement cannot be easily formalized because significance may be subject to personal opinions and expertise (e.g. of ontology designers). Nevertheless it is an important claim that an ontology be a thorough representation of a domain not lacking any essential knowledge thereof.
- *Expressiveness*. The knowledge formalized within an ontology should be useful for a variety of tasks. Therefore it must be capable of defining not only domain terms but also their interrelations and axioms or rules that govern these relations. Hence, simple thesauri and type hierarchies alone cannot be regarded as full-fledged ontologies.

Many knowledge representations cannot meet these requirement, and are in fact not designed to do so as they are tailored to a single, very specific task. Knowledge representations of this kind may be optimized for their specific purpose and thus – in their context – prove more useful than more general approaches. Yet, employing formal ontologies also pursues a different goal, namely describing and communicating knowledge on heterogeneous subject domains across different systems. This requires a broad applicability of ontology formalisms which is mirrored in the generality requirement as already described above. Task-dependent, as well as partial and constricted knowledge representations lack the ontological soundness (in the philosophical sense of the word) that is required as a basis for their general epistemological applicability. For this reason it would be misleading to call such knowledge representations ontologies. In order to clarify the notion of formal ontologies we developed Concept Theory, a formalism that we consider suitable for serving as a theoretical framework for specifying general, significant and expressive EOS ontologies. Epistemological and practical applications for EOS ontologies will then be the subject of the following chapters.

2.2 Concept Theory

The following sections will lay out the formalism we call *Concept Theory*, the theoretical foundation we developed for our approach. It is strongly influenced by the basic considerations and arguments presented in section 2.1.1. Starting from a first (informal) introduction to the notion of concepts based on Definition 2.1 of this section, we will further discuss and concretize what concepts are in terms of Concept Theory. Definition 2.4, then, determines the formal structure of such concepts. Subsequently, we are describing how concepts can be used for designing formal ontologies.

Definition 2.1 (informal): Concept

A *concept* is a formal representation of an existent.

If we assume the totality of all intelligible existents as being represented by a set of universals and particulars (as metaphysical realists do) the claim made here is that for each universal and particular there exists a concept that represents it. In other words, any describable fact about reality is expressed by a corresponding concept.

Example 2.1: Some concepts of the domain *Society*

For explanatory purposes examples given in this section will frequently refer to an illustrative domain, *Society*, that describes people, their environment and social relationships.

The concept FAMILY refers to the universal 'family' and is part of the domain *Society*. A simple definition for FAMILY would be that it depends on other concepts like MOTHER, FATHER and CHILD. We may also know about particular families like 'the Smith family across the road' that can be represented by the concept SMITH_FAMILY. In the same way we may find concepts for all members of the Smith family, e.g. BARRY for Barry Smith, the husband of Mary Smith and father of their son John.

Note: The naming conventions for concepts are, of course, irrelevant to the model itself as concept names are technically not more than identifiers or indices to the set of all concepts. For better readability, though, we will continue to use concept names like FATHER and BARRY, always assuming that these names are uniquely addressing one single existent. Accordingly, in our example the concept BARRY only represents the individual Barry Smith, not 'all persons called Barry', or 'the name 'Barry''.

Being a formal definition of an existent each concept establishes a statement of knowledge about this existent. It follows that concepts promise to be naturally suited for constructing knowledge representations, in particular formal ontologies. In addition, with the notion of concepts being grounded in metaphysical realism, *Concept Theory* possesses a strong theoretical basis that is missing in other approaches. The semantics of concepts will be developed on this basis and are thus clearly tangible.

In concluding these informal considerations on concepts we will accentuate their representational properties that follow immediately from Definition 2.1. Hence, concepts constitute explicitly formalized pieces of knowledge about existents. On this account each concept may act as:

- a *reference* to a specific thing in the world *(referential property)*
- a structural *description* of the thing in the world it refers to *(definitional property)*

In Concept Theory, a complete formal description of any given target domain will generally consist solely of concepts, referring to universals and particulars prominent within the domain. Thus it follows that Concept Theory itself is *unicategorical*, i.e. it knows only one category, namely concepts.

Definition 2.2: Universe of Concepts

The total set of all formally definable concepts is called the *Universe of Concepts* (or *Universe) Φ*.

Analogous to the distinction between universals and particulars in general, we put emphasis on the two elementary subsets of the Universe of Concepts Φ:

- $\Phi_U \subset \Phi$ is the set of all concepts that refer to universals
- $\Phi_P \subset \Phi$ is the set of all concepts that refer to particulars

It follows from the properties of universals and particulars that the Universe of Concepts is disjointedly divided into these two subsets:

- $\Phi_U \cap \Phi_P = \emptyset$, as universals and particulars are distinct categories
- $\Phi_U \cup \Phi_P = \Phi$, as reality is constructed from universals and particulars only.

The Universe of Concepts is the elementary set of Concept Theory (cf. Definition 2.2). All metaphysical existents, universals and particulars, are represented by concepts of Φ_U and Φ_P, respectively. Each concept is a concise formal definition of a single existent, i.e. itself an existent with singular properties and a specific structure. Thus, metaphysically speaking, all concepts are concrete particulars, instantiating from the universal *concept*. This metaphysical status of concepts is depicted in Figure 2.2. Arrows pointing from Φ_U to universals and from Φ_P to particulars are indicating the referential property of concepts towards all existents.

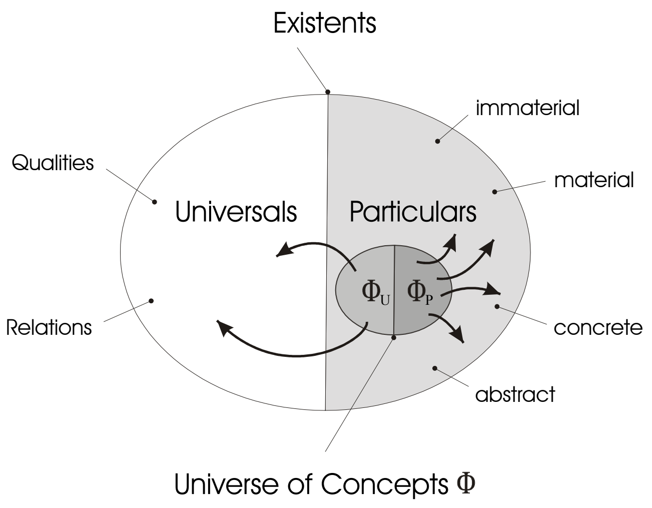


Figure 2.2: Concepts within Metaphysical Realism

On this theoretical basis we will subsequently develop the syntactical and semantical features of Concept Theory. Section 2.2.1 presents the syntax of Concept Theory, including the formal definition of concepts. Sets of concepts, i.e. Φ and significant subsets are discussed in section 2.2.2. In section 2.2.3 we will further examine two kinds of concepts, namely conceptual universals of Φ_U and conceptual particulars of Φ_P. The semantics of concepts, which are established with respect to the ontological relation *specialization* are central to section 2.2.4. This section concludes with a formal definition of ontologies that arises as a direct consequence from the semantic features of Concept Theory.

2.2.1 Syntax

Concept Theory is based on its own notion of 'concepts' that goes well beyond the ordinary usage of the term. Concepts are not merely abstract or generic ideas generalized from particular instances (cf. Merriam-Webster) but explicit descriptions of existents and as such concrete entities exhibiting structure and semantics. Each concept constitutes an object of knowledge and can be addressed by its concept label.

Definition 2.3: Concept Label

A *concept label* is a name (a character sequence) that acts as an unambiguous reference to its corresponding concept, i.e.

- $\exists$ function l: $\Phi \rightarrow$ CHAR*
- $\forall$ $C_1, C_2 \in \Phi$: if $l(C_1) = l(C_2)$ then $C_1 = C_2$ (i.e. function l is injective).

For each concept there exists a corresponding concept label that identifies it uniquely. The relation between concepts and their labels is thus well defined, according to the injection that associates each concept with its corresponding label.

Example 2.2: Concept Label

Let MOTHER, FAMILY $\in \Phi_U$ be concepts describing the universals *mother* and *family*. The corresponding concept labels can be defined as

l(MOTHER) := 'Mother_Concept'

l(FAMILY) := 'Family_Concept'

Note that the only way to talk about concepts is, in fact, by reference. Strictly speaking, 'MOTHER' itself is not a concept but only a name, a reference just like the label 'Mother_Concept' is. Therefore, from a practical point of view, we will not differentiate anymore between the syntactical representation of concepts and their labels. Within Concept Theory, expressions like 'MOTHER' and 'Mother_Concept' are thus treated equally. With concept labels being introduced we can now give a formal definition of concepts themselves, as Concept Theory addresses them.

Definition 2.4 (formal): Concept

A *formal concept* (or *concept*) C is a tuple $C:=(n, L)$ where

- $n \in \mathbf{N}_0$ denotes the arity of concept C, i.e. the number of concepts in L
- L is an ordered tuple of n concept labels $l_1, l_2, ..., l_n$.

The tuple L of concept labels determines the internal structure of concept C, i.e. it accounts for other concepts that are part of C. These concepts contained in C are called its *subconcepts*, or *components*.

Note:

- As a short notation for referring to labels of the tuple L we adopt the one used for designating the elementary relation for sets, i.e. we use

  $$l(\mathrm{CONCEPT}) \in L$$

  instead of

  $$l(\mathrm{CONCEPT}): \exists\ i \in \mathbf{N}: l(\mathrm{CONCEPT}) = L[i].$$

- An integrity rule for conceptual particulars arises immediately from the notion of components, which are representing the constituents (i.e. the parts) of the existents a conceptual particular is describing. As existents cannot be proper parts of themselves, for any conceptual particular $C:=(n, L) \in \Phi_P$ and the set ϕ_C of all of its (recursive) components it follows that:

  $$l(C) \notin \phi_C.$$

A concept is by definition directly related to the existent it describes, i.e. defining a new concept immediately implies that there is a designated part of reality it is a referent of. From a knowledge representational point of view, the semantics of concepts (or any other knowledge representation entity) usually fail to fully describe the existents they are referring to, e.g. it is very hard to completely capture the meaning of the concept FAMILY using a formal model. Yet, this is not the intended use of concepts and formal ontologies. The semantics of *existents* themselves that are referenced and described by concepts are fixed in separate theories of other disciplines – e.g. the natural sciences or metaphysics – which are beyond the scope of this thesis. This thesis concentrates on the use and semantics of *concepts* in Concept Theory, which will be developed throughout this and the following chapter.

As mentioned above, a first approach to the semantics of concepts is that they possess a *referential property* and a *definitional property*. The *referential property* of concepts implies that each concept references *something* (an existent), and does so uniquely in a sense that concepts are – contrary to terms in natural language – unambiguous. There are no two concepts describing the same existent, and one concept always refers to a single existent alone. ‘Frank’, for example, is an English word that has very different meanings, depending on its uses as a noun (the name ‘Frank’), an adjective (being ‘frank’, openly showing thoughts and feelings), or a verb (to ‘frank’ a letter). Yet, as names, properties and activities clearly are different existents the three senses of ‘frank’ would certainly yield three different concepts. A fourth concept may describe the English homonym ‘frank’ that is ambiguous, while the corresponding concept is not, as it is simply a description of a particular word having several meanings. Generally, linguistic ambiguities like homonyms, synonyms, etc. do not affect Concept Theory.

The *definitional property* of concepts is realized by a structural explication of the metaphysical existents they represent. This is achieved by the ordered tuple of concept labels that, metaphysically speaking, denote the constituents of the respective existents. As an obvious integrity rule, only labels referring to previously defined concepts may be used within concept definitions. Thus, at its core, any formal ontology in Concept Theory relies on a fundamental singular concept like EXISTENT[23] that represents the most basic universal *existent*. All other concepts can be defined on that basis. Without loss of generality we will as-

[23] The choice for the name EXISTENT is obvious, yet of course arbitrary. Regardless of this naming convention, the existence of such a fundamental ‘root’ concept plays an important role in the construction of EOS ontologies.

sume EXISTENT as the fundamental concept common to all EOS ontologies within the scope of this thesis.

Example 2.3: The Concepts EXISTENT and FAMILY

The fundamental concept EXISTENT is defined as

EXISTENT := (0, ( )).

Subsequent concepts can now use the label EXISTENT in their own concept definitions. This way, more specific concepts possessing an arbitrary number of subconcepts may be designed that will finally lead to the concept definitions for MOTHER, FATHER and CHILD. Applying these to the specification of the concept FAMILY as mentioned in Example 2.1 will yield

FAMILY := (3, (MOTHER, FATHER, CHILD)).

Note: It may be reasonable to define singular, or *bare* concepts besides EXISTENT, i.e. concepts that do not contain any other subconcepts. Determining subconcepts would mean to set a specific structure which may neither tenable nor wanted for these concepts. A short notation for expressing bare concepts omits the information about arity and concept labels. Thus, a definition of EXISTENT equivalent to the one above would be

EXISTENT := ( ).

Concept labels refer to concepts as a whole, i.e. FAMILY denotes the universal *family*, regardless of its internal structure. If subconcepts of concepts have to be addressed directly this can be done with *concept references*.

Definition 2.5: Concept Reference

Let Φ denote the set of all valid concepts. A *concept reference* is a function

cref: $\Phi \times \mathbf{N} \rightarrow \Phi$

that for a given concept C:=(n, L) $\in \Phi$ and an integer i$\in \mathbf{N}$ returns L[i], the i-th concept of the tuple L.

Note: As an equivalent way of expressing a concept reference *cref(C, i)* we introduce the shorter notation *C[i]*.

Trivially, there are no concept references to bare concepts as these concepts do not contain any components. The semantic difference between concept labels and concept references is that the latter reference (sub)concepts within the *context* of another concept. Thus, concept references pertain to the structure of a concept while labels pertain to the concept as such.

Example 2.4: Concept Reference

Two references for the concept FAMILY := (3, (MOTHER, FATHER, CHILD)) are:

---

cref(FAMILY, 1) = FAMILY[1] = MOTHER

cref(FAMILY, 3) = FAMILY[2] = CHILD

Having introduced the fundamental components of Concept Theory we will briefly interpret them along the lines of metaphysical realism. Concepts are explicit definitions of existents and as such, metaphysically speaking, particulars, more precisely particulars instantiating the universal *concept*. Analogously, concept labels and references are particulars of the universals *concept label* and *concept reference*.

Recurring to the formal definition of concepts we are now in a position to examine mereological[24] and set properties of concepts. A mereological approach to concepts stresses their internal structure, i.e. the relation between concepts and their subconcepts, whereas the set oriented view on concepts regards them as elements of meaningful sets (such as ontologies) that represent parts of reality. Sets of concepts are discussed in section 2.2.2. The subsequent sections will then concentrate on the properties of concepts with respect to their subconcepts.

2.2.2 Sets of Concepts

This section concentrates on sets of concepts significant to Concept Theory. The set comprising all intelligible[25] concepts is the Universe of Concepts Φ (cf. Definition 2.2) and, consequently, all other sets are per definition subsets of Φ. Trivially, the Universe is a (not only enumerable) infinite set. This superset of all intelligible concepts e.g. clearly contains subsets that are themselves infinite sets, like the set {One, Two, ...} of all concepts that describe individual integers. Although we are taking Φ to be an infinite set, it should be clear that all ontologies that are modeled using concepts are of course finite, i.e. consisting of a finite set of concepts. For theoretical reasons we will nevertheless sometimes mention considerations where not only finite ontologies are regarded but Φ as a whole.

Definition 2.6: Denominator

Let $sub(\Phi) \subseteq \Phi$ denote a subset of the Universe of Concepts, let $(sub(\Phi))$ be the lexicographically ordered[26] tuple of all concepts of the subset $sub(\Phi)$ and $|sub(\Phi)|$ the cardinality of this subset. The *denominator den* $\in \Phi_U$ of this subset is the concept

$$den(\, sub(\Phi)\,) := (\, |sub(\Phi)|,\ (sub(\Phi))\,)$$

that acts as a representative of the subset.

Note that from a theoretical point of view, the notion of concepts as formal definitions of existents implies a logical paradox. As any such definition is itself an existent, so is the 'utmost' denominator den(Φ):=(|Φ|,(Φ)) of the Universe of Concepts. Considering that Φ per definition contains the totality of all intelligible concepts this, consequently, must also valid

[24] Mereology is a philosophical discipline embracing all formal theories of part and whole (cf. Simons, Parts)

[25] Concepts and existents correspond with each other, i.e. for any existent we assume a concept that describes it. Different theories may produce different existents they are working with and new theories may introduce new kinds of existents, all being represented by their corresponding concepts. The set of all of these (possible) concepts that are necessarily based on according theories of existents is what we call the set of all 'intelligible' or 'conceivable' concepts.

[26] This ordering is implied by the lexicographical order of the concept names.

for den(Φ), i.e. den(Φ) $\in$ Φ. Thus den(Φ) references itself ad infinitum (cf. [12]). Fortunately, this paradox is not relevant for practical applications as knowledge representations are always finite.

Definition 2.7: Domain, Domain Space

A *concept domain* (or *domain*) *D* is a finite or infinite subset of the Universe Φ: $D \subseteq \Phi$.

The *domain space* Ψ of all *valid* domains, is $\Psi \subset 2^{\Phi}$, a subset of the power set of Φ, where all domains $D \in \Psi$ satisfy the condition that they be *sensible* in terms of modeling reality (a condition that cannot be captured syntactically but has to be evaluated and ascertained by a human expert).

Technically, any subset of the Universe is a valid domain, while the domains interesting for knowledge representation will only amount to a small fraction of all possible subsets. No formal notion of 'interestingness' concerning domains can or should be defined at this level and thus be integrated into Concept Theory. Domains act foremost as abstractions, (possibly infinite) sets that cannot necessarily be described by extension but are better captured by intension. Regardless of infinite sets, extensionally specifying a domain may prove itself inadequate in many cases, and may even be unwanted, as domains are rather mental constructs used for modeling a knowledge representation than an actual part of it. Note in this context that domains can contain several concepts that pertain to the same metaphysical existent, which accounts for the fact that there may be various equally suited ways of describing this existent.

Concept Theory does not restrict domains to be subsets of Φ_U alone, i.e. concepts describing particulars may very well be elements of domains. Consequently, domains may change over time, taking into account that new particulars may come to existence, and already present particulars may be altered or cease to exist. This understanding of domains is an immediate consequence of the epistemological import of domains. Useful knowledge about a meaningful portion of reality – and this is what domains are supposed to represent – will necessarily entail both, abstract background knowledge and concrete facts about reality.

Example 2.5: The Domain *Society*

The domain *Society* $\in \Psi$ is to be defined as the set of all concepts referring to universals and particulars that explain or are part of a human society:

- extensional definition: *Society* := { FAMILY, MOTHER, Mary, FATHER, Barry, Peter, CHILD, ... }
- intensional definition: *Society* := { C $\in$Φ | C refers to *society* }

While the natural way to describe domains is intensional, knowledge representations in general and formal ontologies in particular, are defined as extensional enumerations of their different components, concepts in the case of Concept Theory.

Definition 2.8: Formal Ontology

Let D $\in \Psi$ be a domain. A corresponding *formal domain ontology* (or *formal ontology, ontology) O(D)* is a finite subset of D where

$\forall c_i, c_j \in O(D)$:
c_i represents a metaphysical existent $e \wedge c_j$ represents the same existent $e \Leftrightarrow c_i = c_j$,

i.e. within a formal ontology each concept refers to an existent *uniquely*, which implies that two concepts possessing different names will reference different existents.

This definition of formal ontologies implies that there may exist more than one formal ontology for a given domain which reflects an intuitive fact: there are different and potentially equally suited ways of modeling a formal description of one and the same domain. A simple ontology of the domain *Society* is given below.

Example 2.6: An Ontology of the Domain *Society*

A very simple ontology of the domain *Society* is

O_1(*Society*) := (FAMILY, PERSON, FATHER, MOTHER, CHILD).

O_1(*Society*) is represented by the corresponding denominator

SOCIETY := (5, (O_1(*Society*))).

Respective concept definitions are:

FAMILY := (3, (FATHER, MOTHER, CHILD))
PERSON := ( )
MOTHER := ( )
FATHER := ( )
CHILD := ( )

where PERSON, MOTHER, FATHER and CHILD are described as bare concepts.

Note that the concepts PERSON, MOTHER, FATHER and CHILD of Example 2.6 are actually referring to qualities (monadic relations) pertaining to individuals, e.g. *being a mother* can be instantiated by a woman, say, Mary Smith. Thus MOTHER could be defined more precisely as

MOTHER := (1, (WOMAN)).

The less specific definition of MOTHER within O_1(*Society*) as a bare concept does not give this information. This lack of detail is not (necessarily) a sign for an erroneous ontology design. Depending on the intended focus of an ontology, the universal *mother* may be described by a more or less detailed concept. The degrees of granularity and significance can be determined independently from each other when modeling a domain.

Technically, each ontology is also a domain of its own as any subset of a given domain is itself a domain. But this point of view ignores the aim of knowledge representation, namely providing an *adequate* formal description of some *meaningful* portion of reality (i.e. a domain). The quality and usability of an ontology therefore depends on how, regarding adequacy and completeness, it represents the target domain with respect to a predefined task. While it is true that virtually any set of concepts is trivially a knowledge representation on its own account, its actual value can only be judged by specifying the domain it has been designed to describe.

Regarding concepts from this perspective stresses their set theoretic properties. In this sense concepts form the basic constituents of larger bodies, sets of concepts like ontologies, domains and, ultimately, the Universe. On the other hand concepts are formal definitions of existents that are highly structured. Each concept may be regarded as an mereological object, a single unit organized by a tuple of component labels that mark its subconcepts. The following sections will concentrate on this latter view on concepts.

2.2.3 Kinds of Concepts

In this section we will present two important kinds of concepts, namely conceptual relations and particulars, by referring to their mereological properties. While the preceding sections have introduced concepts as formal definitions of existents and treated them as elements of domains, meaningful sets of concepts, we will now concentrate on their expressive power. Specifically, concepts should be able to describe existents of arbitrary kind. Based on arguments drawn from metaphysical realism we claim that conceptual relations and particulars own precisely the expressiveness needed to do so.

The ontology O_1(*Society*) shown in Example 2.6 is very limited not only due to the small number of concepts it encompasses but in expressiveness. It simply does not give much information about the participating concepts, e.g. the fact that mothers, fathers and children are persons is missing, although concepts MOTHER, FATHER, CHILD, and PERSON are actually part of the ontology. In order to increase expressiveness, facts of this kind, i.e. *relations* between concepts, should therefore be stressed within the ontology. Accentuating interrelations between concepts is an important concern as it means giving more and useful information on the respective domain which is the central task of an ontology.

Concepts are set in relation to each other by, yet again, concepts. It follows from the internal structure of concepts that two or more concepts are interrelated if they are contained in the same concept. In other words, a concept may be regarded as a relation on its components, represented by the corresponding tuple of concept labels. The only concept of O_1(*Society*) possessing subconcepts at this point is FAMILY. The components of FAMILY are MOTHER, FATHER, and CHILD, i.e. these three concepts are set into relation by their superordinate concept FAMILY.

This relational view on concepts (as opposed to a view where concepts are regarded as mere entities) can be comprehensibly illustrated by referring to the theoretical foundation as outlined by metaphysical Realism. Realism divides reality into universals and particulars, and Concept Theory claims that elements of both categories of existents can be referenced and defined by single constructs of the same kind, namely concepts. The two main subcategories of universals are qualities and relations, where qualities, in fact, are themselves monadic relations pertaining to only one existent. In Concept Theory relations are generally expressed through *conceptual relations*, concepts possessing subconcepts.

Definition 2.9: Relation

A *conceptual relation* (or *relation*) is a concept $R := (n, L) \in \Phi_U$ where

$n > 0 \wedge \exists \bar{l} \in L, \exists C \in \Phi_P: l(C)=\bar{l}$,

i.e. R does contain subconcepts and at least one of these subconcepts references a universal.

Per definition, not all subconcepts of a conceptual relation will reference a particular. This demand follows immediately from the realist point of view. Relations, according to realists, are universals that can be exemplified, or instantiated (by particulars), e.g. what is meant by the relation *family* can be instantiated by several particular groups of persons. Relations, as treated by Concept Theory, must reflect this. Thus they have to possess at least one subconcept that can be further exemplified, i.e. that is not itself a description of a metaphysical particular. Being referents to universals, conceptual relations consequently are elements of Φ_U, which is just the set of all concepts referring exclusively to universals.

Example 2.7: Relation

The concept FAMILY is a triadic relation. All of its subconcepts are themselves elements of Φ_U:

FAMILY := (3, (FATHER, MOTHER, CHILD)) $\in \Phi_U$

Let Barry $\in \Phi_P$ denote the individual Barry Smith. The relation

ANOTHER_FAMILY := (3, (Barry, MOTHER, CHILD)) $\in \Phi_U$

describes all families satisfying the restriction that the father of these families be Barry Smith – which in the special case of families can only be true for one family. Nevertheless the definition of ANOTHER_FAMILY still references a universal (that may theoretically be instantiated by only one metaphysical particular, the Smith family).

If a concept were to contain only subconcepts referencing particulars it would be fully determined. Accordingly, it could not be referencing a universal (a relation) anymore but an actual particular that instantiates a relation. Such a concept is called a *conceptual particular*.

Definition 2.10: Particular

A conceptual particular (or particular) is a concept $p := (n, L) \in \Phi_P$ where

$\forall\ C \in L: C \in \Phi_P$,

i.e. all subconcepts in L reference solely particulars.

Such particulars are concepts that hold volatile information about a given domain in a sense that they represent actual occurrences[27] that may be created, be subject to change and cease to exist. As an example, a specific family like the Smith family has been started at some

[27] Until their formal definition in Definition 2.14 we regard *occurrences* as – particular or universal – specializations of universals.

point in time, the members will change over the years, and the Smith family may also be ended, e.g. by divorce.

Example 2.8: Particular

Let Barry, Mary and John $\in \Phi_P$ denote the individuals Barry, Mary and John Smith. The concept

$$\text{Smith\_family} := (\, 3, (\, \text{Barry, Mary, John} \,)\,) \in \Phi_P$$

is a particular representing the very family of Barry, Mary and John Smith. All subconcepts are fully determined, i.e. themselves elements of Φ_P which cannot be further instantiated.

Syntactically, conceptual universals and particulars possess the same structure as they are both concepts. In order to distinguish between concepts of Φ_P and Φ_U on an ontology modeling level, as a syntactical convention, particular names in Concept Theory are indicated either by the prefix '**PARTICULAR:**', or by using lowercase letters – as opposed to universal names that only contain uppercase letters. Thus, 'Smith_family' designates a particular $\in \Phi_P$ while 'FAMILY' is a universal $\in \Phi_U$.

While relations state unchangeable facts about a domain, particulars express its present status at a given point in time. The analogy to the relationship between schema and data in database technology or classes and objects in object-oriented systems is obvious but also misleading. There is a fundamental difference in terms of abstraction and actuality between a database schema and the data (including the physical representation of the schema in database relations) stored in the respective database according to the schema. The same applies to classes that define the properties of objects and the objects themselves that exist and interact at runtime. There is no such functional difference between relations and particulars in Concept Theory. Both are structurally equal constructs: concepts referring to existents. The *objects* concepts are referencing may be different in their nature, i.e. be metaphysical universals or particulars, but *concepts* refer to all existents in the same manner, by specifying them individually and determining their internal composition.

Regarding concepts from this perspective stresses their relational properties. The internal structure of a concept defines the subconcepts it relates to each other. This view, combined with the set theoretic and mereological perspectives form the cognitive background of Concept Theory. Each concept is an individual unit and may be alternately regarded as

- an *element* of a larger, meaningful set of concepts (e.g. ontologies and domains)
- a *definition* of the existent (universal or particular) it references
- a *relation* on its subparts

Relational properties of concepts will be further analyzed in the following sections. The impact of these properties on ontologies consisting of formal concepts is central to Concept Theory.

2.2.4 Semantics of Concepts

On the basis of the set theoretic and mereological properties of concepts as examined in sections 2.2.2 and 2.2.3 we will now discuss their semantics in detail. The fundamental semantics of concepts are based on a special concept, **ISA**, a conceptual relation that represents specialization and generalization. The goal of this section is to motivate and demonstrate requirements for formal ontologies, which will finally lead to a formal definition of a *valid ontology* within Concept Theory. Among several other definitions central to Concept Theory, **ISA** accounts for the semantics of *occurrences* and instances (as special cases of occurrences). As the notion of occurrences is helpful for explaining some of the ideas in this section we will briefly introduce them here, before they are formally defined in section 2.2.4.2 in their appropriate theoretical context. Informally, occurrences are concepts that are examples of other concepts, e.g. MOTHER is an occurrence of PERSON, as mothers are examples of persons, and so is Mary. Note, as implied in this example, that occurrences may be elements of both, Φ_U and Φ_P.

Metaphysical Realism claims that universals are instantiated by particulars. This pertains to qualities as well as polyadic relations: a chair instantiates a certain shape and color, as well as a group of people instantiate being a family, etc. One of the characteristics of universals is that they can be instantiated by more than one particular at a time, e.g. from our experience we know that there are many chairs exhibiting the same shape, we will find numerous objects of the same color and there are also many different groups of people instantiating what it means to be a family. In other words, at any time it is possible that various occurrences of one and the same universal are exemplified.

That is to say that there exists a fundamental relation between each occurrence of a universal and the universal itself: the Smith family and the Simons family both are families, occurrences of the universal *family,* to be more exact. Moreover, the scope of this fundamental relation is not limited to hold only between universals and their related particulars but also among universals alone, e.g. a middle-class family is a family, too. Thus, the fundamental relation generally expresses *specialization and generalization*, the utmost specializations being the ones from universals to their respective occurrences, i.e. particulars. What makes this relation fundamental is that, without exception, *any* existent takes part in an occurrence of it. Therefore it comes as no surprise that metaphysicists regard this relation with special interest – as pointed out in section 2.1.1 the intention to answer the question "What is it?" for all existents has been a central motivation for designing ontologies in the first place.

2.2.4.1 Ontological Status of the Concept ISA

Specialization and generalization express the answer to the metaphysical question "What is it?" by positioning existents in relation to each other and in relation to ontological categories. In Concept Theory specialization and generalization are expressed through the concept **ISA**.

Definition 2.11: The Fundamental Relation ISA

The relation **ISA** is defined as the concept

$$\textbf{ISA} := (\,2, (\,\text{EXISTENT}, \text{EXISTENT}\,)\,) \in \Phi_U$$

---

implying the following semantics for the components of **ISA**:

- **ISA**[1] $\in \Phi$ is a concept that is generalized by **ISA**[2],
- **ISA**[2] $\in \Phi_U$ is a concept that is specialized by **ISA**[1].

Note: Occurrences of **ISA** are marked as such by the prefix **ISA:**, e.g.

ISA:isa_occurrence := (2, (APPLE, FRUIT)) $\in \Phi_P$

The prefix **ISA:** thus distinguishes occurrences of **ISA** syntactically from all other dyadic concepts.

The semantics of **ISA** are a declared part of Concept Theory, the relation is thus *ontologically prior* to all other concepts that may be defined within an actual formal ontology. Only with **ISA** being included in Concept Theory it is possible to capture specialization and generalization among existents semantically.

Example 2.9: Some Examples of the Relation ISA

Let MOTHER, Mary and PERSON be concepts of the ontology O_1(Society). We can now formulate that *Mary* is a *mother*, and that *mothers* are *persons*:

ISA:isa_mary := (2, (Mary, MOTHER))
ISA:isa_mother := (2, (MOTHER, PERSON))

The prefix **ISA:** identifies these two concepts as occurrences of **ISA**. Otherwise, the semantics of **ISA** could not be expressed as shown in the following example where Isa_mary is defined without the prefix, i.e. without explicitly stating that this concept represents a specialization.

Isa_mary:= (2, (Mary, MOTHER))
Note that at this point Isa_mary is not yet identified as an occurrence of **ISA**. It is still ontologically equal to all other dyadic concepts relating Mary and MOTHER. In order to mark it as an occurrence of **ISA** we could add a new concept:

EXAMPLE_OF_ISA := (2, (Isa_mary, **ISA**))

But, yet again, EXAMPLE_OF_ISA itself is just another dyadic concept providing no further semantics than that it relates Isa_mary and **ISA**. What this relation expresses semantically remains unresolved. In fact we can define an infinite number of concepts, each relating the former to **ISA**, but the underlying semantics that the respective concepts refer to examples of *specializations* could never be caught. Therefore the fundamental specializations from **ISA** to its occurrences need to be made explicit within the syntax definition of Concept Theory and this is exactly what is achieved by using the prefix **ISA:**. With the fundamental specializations being semantically tangible in this way all other specializations between existents can be modeled on this basis. In fact, **ISA:**isa_mary actually symbolizes two occurrences of specialization: (a) the particular specialization that marks the relation that exists between the particular *Mary* and the universal *mother* as an occurrence of the universal *specializati*on and (b) the particular specialization from *mother* to *Mary* itself.

This definition of the relation **ISA** and its occurrences (Definition 2.11) reflects the *self-referential* quality of specialization/generalization: a specific relation between, two existents, e.g. *Mary* and *mother,* is a specialization, and stating this fact is a specialization of the universal *specialization* itself, etc. Concept Theory is able to express this recursion to an arbitrary depth while the semantics of specialization/generalization are being preserved at any time. Otherwise a formal ontology could not coherently model relationships between existents as the concepts forming the ontology would be distinguishable, yet isolated in meaning, e.g. two occurrences of families could still be referenced by their corresponding concepts (that will both exhibit the same *structure*, i.e. number of components) but the fact that these concepts actually represent examples of the universal *family* would be lost. **ISA** provides the model with the semantic power that is needed to express such relationships.

The reason for the self-referential quality of specialization/generalization lies in the nature of universals, *specialization* being one of them. As universals can be instantiated by several particulars at a time, each particular takes part in its own private instantiation of a universal that is numerically different from all other instantiations of that same universal. Analogously, there must also be distinct conceptual particulars of **ISA**, each describing one of the numerically different instantiations. But then their common property, namely that they really are occurrences of **ISA** is not yet expressed. This has to be done by *self-reference of **ISA***, i.e. each specialization among arbitrary concepts implies a self-referential occurrence of **ISA** of the form

ISA:self-reference := (2, (**ISA:**isa_occurrence, **ISA**))

that describes the respective specialization among concepts also as specialization of the concept **ISA** itself.

2.2.4.2 Properties of the Concept ISA

ISA is the fundamental relation of Concept Theory. It represents specialization and generalization among existents and exhibits semantic properties that can be used for defining the structure of formal ontologies:

- *Self-referentiality*. The self-referential quality of **ISA** has been discussed in detail in section 2.2.4.1. Its integration into the syntactical definition of concepts is the semantic basis of Concept Theory. For any concept C, viewed from a relational perspective, **ISA** is identifying all concepts that are occurrences of C.
- *Reflexivity*. Trivially, any existent is its own specialization, which in Concept Theory is expressed by the rule that,

  $\forall$ C$\in\Phi$: $\exists$ an **ISA** occurrence **ISA:**occ :=(2, (C, C)) $\in\Phi$

  notwithstanding whether this occurrence of **ISA** is stated explicitly within an actual ontology or not.
- *Antisymmetry*. Specialization is an antisymmetric relation, i.e.
  $\forall$ C_i, C_j $\in\Phi$: if $\exists$ **ISA:**occ1:=(2, (C_i, C_j)) $\in\Phi \wedge \exists$ **ISA:**occ2:=(2, (C_j, C_i)) $\in\Phi \Rightarrow C_i = C_j$.
- *Transitivity*. The transitive property of **ISA** and its impact on the semantics of formal ontologies will be presented in section 2.2.4.2.1.
- *Stringency*. **ISA** is stringent in that it is necessarily related to all concepts of a formal ontology through one of its occurrences. The resulting graph structure and its implications will be discussed in section 2.2.4.2.2.

- *Continuity*. Within a formal ontology, the internal structure of concepts is being preserved by **ISA**, i.e. concepts inherit the structure of their parent concepts. This structural continuity will be further explained in section 2.2.4.2.3.

Based on these properties of the concept **ISA** we are in a position to develop a clearly defined notion of formal ontologies, and moreover, give semantic and syntactical conditions for valid ontologies.

2.2.4.2.1. Transitivity

Specialization is a transitive relation, i.e. from two **ISA** occurrences

ISA:occ1 := (2, (A, B)) and **ISA:**occ2 := (2, (B, C))

we can infer a third **ISA** occurrence

ISA:occ3 := (2, (A, C))

that is implied by the first two occurrences. Transitivity is an important semantic property of **ISA** as it allows for deducing new concepts, therefore explicitly manifesting implicit knowledge present within an ontology. The impact of transitivity on concepts is expressed by the principle of *reducibility*.

Definition 2.12: Reducibility

Let O be an ontology and S∈O and G∈O be two concepts of O. S is *reducible* to G in O if there exists a transitive sequence

ISA:isa1:=(2,(S,C_1)), **ISA:** isa2:=(2,(C_1,C_2)), …, **ISA:** isaN:=(2,(C_{N-1},G)) ∈O

of **ISA** occurrences from S to G.

Basically, reducibility expresses *categorization* within Concept Theory, i.e. if a concept S is reducible to another concept G, then S is *of kind* G, or *falls into the category* G. Note that any concept is trivially reducible to itself because of the reflexive quality of **ISA**, which is a semantically natural way of looking at concepts – any concept C is of kind C.

Example 2.10: Reducibility

Let O be an ontology and

ISA:occ1:= (2, (WOMAN, PERSON)), **ISA:**occ2 := (2, (Mary, WOMAN))

be **ISA** occurrences ∈O. From **ISA:**occ1 and **ISA:**occ2 we can infer

ISA:occ3:= (2, (Mary, PERSON))

i.e. Mary is reducible to PERSON.

Reducibility is a useful means for explaining the semantic implications of **ISA** seen from a relational point of view. As a transitive relation on other concepts **ISA** defines *structure* within ontologies, which leads to the constructive properties of **ISA** that are examined in the following section.

2.2.4.2.2. Stringency

Concept Theory acknowledges the fundamental significance of **ISA** in that it demands for any ontology in order to be valid to relate all of its non-**ISA** concepts by occurrences of **ISA**. This requirement is a direct consequence from the basic considerations presented in section 2.1.1 where we laid out that an ontology has to be able to answer the “What is it?” question for any existent it encompasses, which corresponds with the semantics of **ISA**. The underlying assumption is that any existent is of a certain kind and thus inhibits a definite and unambiguous position within an ontology.

The stringency of **ISA** immediately leads to far-reaching structural properties of ontologies. Occurrences of **ISA** are associated with all other concepts of an ontology, thus specifying a hierarchy of concepts, starting from the ones describing the most general universals to the ones referring to the most determined existents, i.e. particulars. In fact, the relation **ISA** defines a complete leveled graph with all other concepts being nodes that are connected by edges delineated by the set of occurrences of **ISA**.

Definition 2.13: Concept Graph, Ontology Graph

Let O be a formal ontology. A *concept graph* G_C:=(V_C, E_C) of a concept C:=(n, L) is a directed graph defined by

- the set of vertices $V_C \subseteq O$ where $\forall\ v \in V_C$: v is reducible to C in O, i.e. V_C contains C and all of its specializations, and
- the set of edges $E_C \subseteq V_C \times V_C$ where $\forall\ (v_1,v_2) \in E_C$: $\exists$ **ISA**:occ:=(2, (v_2,v_1))$\in$O, i.e. E_C is defined by occurrences of **ISA**, and edges are pointing from general to specialized concepts.

An *ontology graph* G_O is the concept graph of the fundamental, most general concept EXISTENT$\in$O, i.e. $G_O := G_{EXISTENT}$:=($V_{EXISTENT}$, $E_{EXISTENT}$).

The ontology graph G_O regards **ISA** as a relation on all other concepts. It is *leveled* in that it possesses a root vertex (usually EXISTENT) that has no incoming edges. The levels of G_O are delineated by **ISA** edges. As an ontology graph is the complete graph of all concepts of an ontology O, i.e.

$$O = V_{EXISTENT} \cup E_{EXISTENT},$$

there is a path from the root concept to any other concept (excluding **ISA** occurrences, the vertices of the graph). The distance from the root concept, i.e. the length of the longest path leading from the root concept to a vertex concept, then indicates the level of the particular vertex concept. Thus, an ontology graph level comprises all concepts of the same distance to the root concept. Obviously, as can be deduced from the semantics of **ISA**, there are no backward edges from child to parent vertices.

2.2.4.2.3. Continuity

For any concept there exists a subgraph of the ontology graph G_O featuring this concept as an ‘intermediate’ root concept. All subsequent concepts of this subgraph represent specializations, and thus *occurrences* of the intermediate root concept. There is a common property of these child concepts that is associated with their internal structure in comparison with the inner structure of the intermediate root concept. We paraphrase this common property as *structural continuity* which accounts for the general pattern that all occurrences 'inherit' the internal structure of the intermediate root concept, similar to inheritance in object-oriented design [66]. Accordingly, the intermediate root concept can be regarded as a class/type definition for its subsequent concepts, or in metaphysical terms, as a category that all child concepts are reducible to. As we will show, this property holds for simple as well as for multiple inheritance where a subsequent concept represents a specialization of several, more general concepts.

From the transitive property of **ISA** it follows that there exists a direct specialization from a general concept G to any of its child concepts S, i.e. there is an **ISA** occurrence

$$\textbf{ISA}{:}\text{occ} := (2, (S,G))$$

that contains S and G as subconcepts. This **ISA** occurrence, next to defining a specialization from G to S, establishes a *structural* relationship between these two concepts. The structural relationship is on the respective subconcepts of G and S, as defined by their label tuples, and allows for a formal definition of the syntactic properties of concept occurrences. Occurrences preserve the subconcept structure of their general concept. If an occurrence S of G does not possess any additional subconcepts it represents an instance of G.

Definition 2.14: Occurrence, Instance

Let $S:=(n, L_S)$ and $G:=(m, L_G)$ be two concepts of an ontology O. S is an *occurrence* of G in O if S is reducible to G.

S is a *valid occurrence* of G if

- S is reducible to G,
- $n \geq m$ for $S \in \Phi_U$ and $n=m$ for $S \in \Phi_P$,
- $\forall\ l_i \in L_G\ \exists\ l_j \in L_S$: the concept labeled by l_j is reducible to the concept labeled by l_i, according to a function *role*: $\mathbf{N} \rightarrow \mathbf{N}_0$ that maps indices of L_S uniquely to indices of L_G:
  $\forall\ 1 \leq j \leq n$: role(j):= k, for $0 \leq k \leq m$, where
  $\forall\ 1 \leq j_1, j_2 \leq n \wedge (\text{role}(j_1) \neq 0 \vee \text{role}(j_2) \neq 0)$: $\text{role}(j_1)=\text{role}(j_2) \Leftrightarrow j_1=j_2$, and
  $\forall\ 1 \leq i \leq m\ \exists\ 1 \leq j \leq n$: role(j):= i.

i.e. for any component of the more general concept G there exists a corresponding component of occurrence S that exclusively represents its specialization. The function role() is expressing this mapping of components. Accordingly, the components of G are called the *roles* of the subconcepts of S. The components of S that take over the roles defined in G are named *occupants*. As it is allowed that n>m, i.e. the number of components of S may be greater than the number of components of G, it is possible that, a component $l_j \in L_S$ is not an occupant of a role in G (although all components of G are occupied). In this case, role(j):=0 and l_j is said to be its own occupant.

S is an *instance* of G if

- S is a valid occurrence of G and
- n = m,

i.e. *all* subconcepts of S are occupants, covering all roles in G.

In valid occurrences, without exception, all components of the more general concept are generalizations of designated subconcepts of the more specialized concept. This concept may possess more concepts than its generalization but it is an essential requirement for it to be a valid occurrence that all subconcepts of the more general concept find their distinct specializations within its own subconcepts. Put in other words, the tuple of subconcepts of a valid occurrence comprises valid occurrences of all components of the general concept.

Note that the mapping between components of parent and child concepts, formally introduced with function role(), must be an explicit part of an ontology, as Concept Theory allows for multiple inheritance among concepts. The relation between roles and their occupants is expressed through the concept **ISA** using concept references, e.g.

ISA:occ:=(2, (S[1], G[2]))

For stating that the first component of concept S is an occupant of the second component of concept G.

Example 2.11: Occurrence, Instance

Let PERSON, CHILD and SON be concepts of O_2(*Society*), a second ontology of the domain *Society*, where

PERSON := (2, (NAME, GENDER)),
CHILD := (4, (NAME, GENDER, FATHER, MOTHER)),
SON := (4, (NAME, MALE, FATHER, MOTHER)),

and

ISA:isa1 := (2, (CHILD, PERSON)),
ISA:isa2 := (2, (SON, CHILD)),

assuming that concepts NAME, GENDER, MALE, FATHER, MOTHER are also defined within O_2(*Society*), along with appropriate **ISA** occurrences like

ISA:isa3 := (2, (MALE, GENDER))
ISA:isa4 := (2, (CHILD[1], PERSON[1])),
ISA:isa5 := (2, (CHILD[2], PERSON[2])), etc.

CHILD is a valid occurrence of PERSON as

- CHILD is reducible to PERSON because of **ISA:**isa1,
- the number of components of CHILD$\in\Phi_U$, 4, is greater than that of PERSON, 2,
- all components of PERSON, PERSON[1]=NAME and PERSON[2]=GENDER, have occupants in CHILD, CHILD[1]=NAME and CHILD[2]=GENDER,
- all occupants in CHILD of roles in PERSON are unique, i.e. no **ISA** occurrences besides **ISA:**isa4 and **ISA:**Iisa5 that are regulating the component mapping between CHILD and PERSON are defined in O_2(*Society*).

Analogously, SON is a valid occurrence of CHILD, again assuming that O_2(*Society*) contains a correct component mapping. SON is also an instance of CHILD as the number of subconcepts of the two concepts is identical.

Note that the notion of instances is different from the one in e.g. object-oriented systems. The object-oriented paradigm identifies instances with concrete objects existing at runtime, as opposed to classes, the definitions of objects. These objects, being fully determined existents, correspond with conceptual particulars in Concept Theory. Yet, instances in Concept Theory are not equaled with particulars of Φ_P. Definition 2.14 allows for concepts of Φ_U to be instances, as long as they represent specializations of their parent concepts. In fact, Concept Theory does not differentiate between specialization and instantiation as being two different relations but regards instantiation as a particular case of specialization, represented by **ISA**. Instances are identified syntactically as concepts solely possessing components that are reducible to the subconcepts of the more general concepts they are valid occurrences of.

It is a syntactic convention of Concept Theory that, for simple inheritance, the order of subconcepts of the general concept is preserved by its occurrences, i.e. for any occurrence S of a concept G the rule

S[i] is reducible to G[i]

holds for all subconcepts of G. This way, roles and their occupants can be related to each other by tuple positions. In case of multiple inheritance, this rule may only hold for one parent concept. The general way to express reducibility from occupants to their roles is by introducing occurrences of **ISA** that use concept references of the form

ISA:OCC:=(2, (S[j], G[i]))

where it is possible that $i \neq j$. The general postulate of Definition 2.1 that all roles of G are taken over by occupants in S is thus not affected by multiple inheritance.

Note that this definition also allows that one and the same component of a child concept may simultaneously act as an occupant for roles in different parent concepts – as long as the property that for each parent concept all roles have to be distinctly occupied is not violated. This means that, in case of two parent concepts G_1 and G_2 of a child concept S, a component S[j] of S can be an occupant of both, roles $G_1[i]$ and $G_2[k]$:

ISA:occ_1:=(2, (S[j], G_1[i]))
ISA:occ_2:=(2, (S[j], G_2[k])),

where indices i, j, and k are not necessarily (but possibly) identical. Therefore, two-way inheritance and more complex correlations among concepts can be elegantly expressed in Concept Theory.

Example 2.12: Two-way Inheritance

Let PERSON, MALE_PERSON, CHILD and SON be concepts of the ontology O_2(*Society*), where

PERSON := (2, (NAME, GENDER)),
MALE_PERSON := (2, (NAME, MALE)),
CHILD := (4, (NAME, GENDER, FATHER, MOTHER)),
SON := (4, (NAME, MALE, FATHER, MOTHER)),

and

ISA:ISA1 := (2, (MALE_PERSON, PERSON)),
ISA:ISA2 := (2, (CHILD, PERSON)),
ISA:ISA3 := (2, (SON, CHILD)),
ISA:ISA4 := (2, (SON, MALE_PERSON)).

We speak of two-way inheritance if a concept is the specialization of two concepts that are, themselves, reducible to the same parent concept. As determined by the four **ISA** occurrences this is the case with concept SON. SON possesses subconcepts that are occupants of roles in both parent concepts, e.g.

ISA:ISA5 := (2, (SON[2], CHILD[2])),
ISA:ISA6 := (2, (SON[2], MALE_PERSON[2])),

indicate that subconcept SON[2]=MALE is an occupant of both, CHILD[2]=GENDER and MALE_PERSON[2]= MALE.

Structural continuity has been introduced with the notion of occurrences by referring to the semantics of **ISA** and the basic categories of Concept Theory, relations and particulars. Naturally, conceptual particulars do not possess any further specializations and therefore, graphically, can only represent the leaves of an ontology graph. Furthermore, from a semantical point of view, conceptual particulars should be regarded as *instances*, not only as occurrences of a single concept representing a universal, which affects multiple inheritance to conceptual particulars. Metaphysically speaking, a particular instantiates universals (*properties*), yet not directly but through its membership to a certain *kind*, i.e. a single universal *relation* covering all these properties. Transferred to Concept Theory this concisely means that a conceptual particular should be an instance of solely one conceptual universal, inheriting its internal component structure. Accordingly, a requirement for valid ontologies is that they only allow for simple inheritance to conceptual particulars. This presents no limitation in expressiveness, as it can be easily proven that any ontology allowing for multiple inheritance to particulars can be converted into an ontology where each particular is only an instance of a single concept.

Proof 2.1

Let O be a formal ontology. From any two concepts $G:=(n, L_G)$ and $H:=(m, L_H) \in O$ one can construct a common valid occurrence $S:=(k, L_S) \in O$, either

(a) by simply combining the components of concepts G and H:

- $k := n+m$
- $L_S := (l_{G1}, l_{G2}, \ldots, l_{Gn}, l_{H1}, l_{H2}, \ldots, l_{Hm})$, or

(b) by merging semantically equal roles of G_1 and G_2 to their respective occupants in S so that:

- $\mathrm{Max}(n, m) \leq k \leq n+m$
- $\forall\, l_{Gi} \in L_G\ \forall\, l_{Hj} \in L_H\ \exists\, l_S \in L_S$: $l_S = l_{Gi} \vee l_S = l_{Hj} \vee l_S = l_{Gi} = l_{Hi}$.

In any case, the resulting concept S is a syntactically correct valid occurrence of G as:

- S is reducible to G, which is a premise of the proof, expressed by the **ISA** occurrence **ISA:**occ_1:=(2, (S,G)).
- $k \geq n$, for k is defined as n+m in case (a) and $k \geq \mathrm{Max}(n, m)$ in case (b).
- All components of G are addressed as roles by occupants in S. In case (a) this is done directly by copying the components of G. Considering the reflexive property of **ISA**, it follows for any subconcept of G that there is an identical subconcept in S that is trivially reducible to its counterpart in G. In case (b) this property follows directly from the rule how semantically equal roles are losslessly merged to subconcepts of S.

- Analogously, it follows from copying the subconcepts of G in case (a) and from the manner in which subconcepts of parent concepts are merged in case (b) that all specializations from roles to their occupants in S are distinct.

As concepts G and H can be regarded symmetrically in relation to S, the above considerations apply analogously to concept H as well, which proves that S is also a valid occurrence of H.

Consequently, for any child concept of multiple parent concepts we can introduce an intermediate parent concept that inherits from all initial parent concepts and acts further on as the sole parent of the child concept. Thus, any ontology containing multiple inheritances to particulars can be turned into one that only allows for simple particular inheritance by adding intermediate parent concepts in the explicated manner.

Definition 2.15: Valid Ontology, Saturated Ontology

Let O be a formal ontology. O is a *valid ontology* if

- the **ISA** occurrences of O define a connected, acyclic, directed ontology graph G_O,
- all concept occurrences in O are valid, and
- particulars only have one parent.

A valid ontology is *saturated* if all leaves of the ontology graph are particulars $\in \Phi_P$ and all inner nodes are concepts $\in \Phi_U$, i.e. within a saturated ontology all universals are exemplified by at least one particular.

A valid ontology, therefore, is a directed graph spanned by occurrences of **ISA** complying with the syntactical properties and semantics as outlined in Definition 2.15. From a graph theoretical point of view the specification of occurrences and instances in Definition 2.14 with reference to **ISA** identifies them as child concepts of more general ones within the ontology graph G_O. This definition also determines a general property of valid ontologies concerning their ontology graph structure. The number of concept components may never decrease from parent to child node but either increases or, in case of instances, stays the same. Hence, the degree of specialization that, per definition, increases along the **ISA** hierarchy, is coupled with the number of concept components, which reflects the intuitive fact that the most specialized existents also show the most properties. Accordingly, the most determined concepts of an ontology graph are its leaves, conceptual particulars, characterized by their (structurally identical) immediate parent concepts, the most specialized concepts referring to metaphysical universals.

As any concept may canonically be regarded as a category of its own – being a formal definition of an existent a concept remains an abstract, and thus general description, even if it references a metaphysical particular – and categorization is already included in the semantics of **ISA**, Concept Theory does not stipulate an extra definition of conceptual categories. All concepts capable of possessing further specializations, i.e. the concepts in Φ_U, represent categories for their occurrences, in particular for their instances.

Concept Theory is the formalism used within the EOS framework for defining valid ontologies with clear semantics as outlined in the course of this chapter. It has been designed with reference to metaphysical realism, which provides a strong theoretical basis for modeling formal ontologies. Furthermore, Concept Theory provides a thorough derivation of an appropriate, unicategorical syntax and accompanying semantics of valid ontologies, thus clarifying a suitable notion of formal ontologies. EOS ontologies are general because they are built from most basic constituents, formal concepts referring to existents. As such they are very well suited for capturing the significant aspects central to any domain of interest in an expressive way. Expressiveness in terms of epistemological sufficiency will be the central topic of Chapter 3 where the EOS framework is described in detail.

2.3 Graphical Notation for Concept Theory

As presented in the previous sections, EOS concepts can be defined using ordered tuples. While this syntax is formally concise, for human readers a graphical notation provides a more intuitive access to the subject matter conveyed by ontology concepts (cf. other graphical notations employed e.g. in software engineering such as UML [66], or in database design, as used for developing E/R models [15]). Therefore, this section introduces a simple but powerful graphical notation for EOS concepts. This notation has been designed with regard to its employment in a graphical user interface for ontology engineering. By offering a clear translation of concepts – as understood by Concept Theory – into a graphical metaphor, it is intended to support the work of domain experts when designing EOS ontologies, regardless of their personal mathematical background. The graphical metaphor we are employing for symbolizing a concept is that of a “bubble”, which is represented by an oval bearing the name of the respective concept. As a first example, Figure 2.3 shows the graphical equivalent to the universal EXISTENT:=( ) and two alternative ways of representing the particular Five whose value is “5”[28].

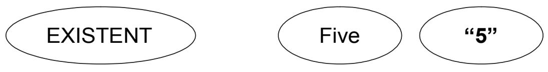


Figure 2.3: Graphical Representation of the Concept EXISTENT

EXISTENT and Five are defined as a bare concepts, i.e. they possess no components of their own. Other concepts that were described in this chapter, on the other hand, do have an internal subcomponent structure, e.g. FAMILY and MOTHER. Figure 2.4 depicts these two concepts and their components as nested bubbles reflecting the definitions FAMILY:=(3,(MOTHER, FATHER, CHILD)) and CHILD:=(4,(NAME, GENDER, FATHER, MOTHER)). As concept components are ordered in Concept Theory, the graphical notation must reflect this by aligning the component bubbles within a concept bubble in a predefined way. This is done by lining up component bubbles from left to right, e.g. the component MOTHER of the concept FAMILY in Figure 2.4 occupies the first component bubble on the left, i.e. it holds position FAMILY[1], while the FATHER and CHILD bubbles are placed at FAMILY[2] and FAMILY[3], respectively. In case a concept possesses a large number of components, these may be arranged in several consecutive lines.

[28] Note the alternative way of displaying particulars using their values is not as precise as using their exact concept names. Concepts, by definition, are unique references to existents while their values may possibly be the same, e.g. Five and Age_of_Peter can have the same value, “5”, but are denoting semantically different facts about the world. An EOS user interface can therefore offer an extra “value view” on particulars, but will always operate by default on the specific concept names.

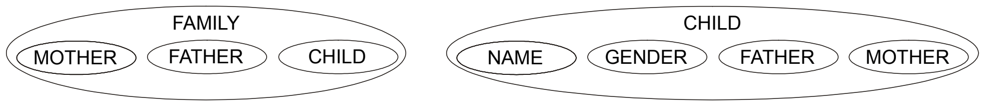


Figure 2.4: Graphical Representation of the Concepts FAMILY and CHILD

Note that representing concepts and their components by nested bubbles may be used for graphically displaying different representation granularities. This means that one and the same concept can be represented in different levels of detail in terms of its subcomponent structure. Figure 2.5 exemplifies a "drill-down" into the component structure of the concept FAMILY. An EOS user interface may offer this drill-down capacity to the user in order to display an arbitrary detailed view on (parts of) an ontology.

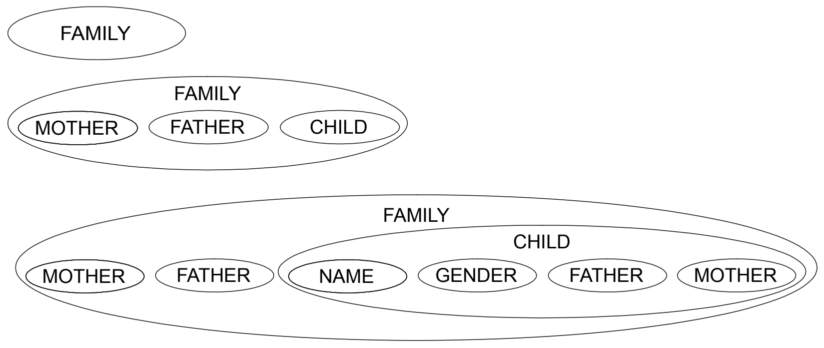


Figure 2.5: Three Representations of the Concept FAMILY in different Levels of Detail

For depicting an arbitrary number of components of the same kind, we introduce the "*" operator. Applied to the CHILD component of FAMILY this indicates that a *family* may possess not only strictly one, but also zero or several *children*. A correspondingly expanded FAMILY concept is shown in Figure 2.6.

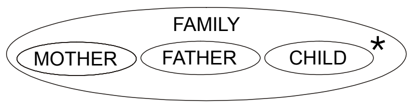


Figure 2.6: An alternative Definition of the Concept FAMILY

Next to single concepts, the graphical EOS notation can also be used for depicting the overall structure of EOS ontologies, i.e. the ontology graph as introduced in Section 2.2.4.2. An ontology graph is a representation of the concepts of an EOS ontology where all **ISA** occurrences are displayed as directed edges between the remaining concepts.

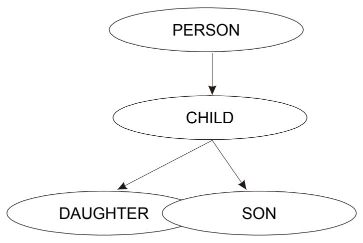


Figure 2.7: Graphical Representation of the Specialization Hierarchy

A fraction of an ontology describing the domain *Society* is presented in Figure 2.7. **ISA** occurrences are depicted as arrows pointing from the more general concept to its specializations. For example **ISA:**cp:=(2,(CHILD,PERSON)) is being represented by the arrow going from the PERSON to the CHILD bubble. In the same way, the edges between the CHILD node and its specializations correspond to **ISA:**dc:=(2,(DAUGHTER,CHILD)) and **ISA:**sc:=(2,(SON,CHILD)). Thus, using this graphical notation based on the bubble metaphor, an ontology engineer can define all concepts of an EOS ontology and arrange them into an ontology graph according to the specialization hierarchy of a given domain. Note that Figure 2.7 only portrays the specialization hierarchy among concepts and does not show any component mappings (cf. Section 2.2.4.2). If necessary, these can be also included in the graphical representation. As Figure 2.8 indicates, component mappings are also depicted using directed arrows.

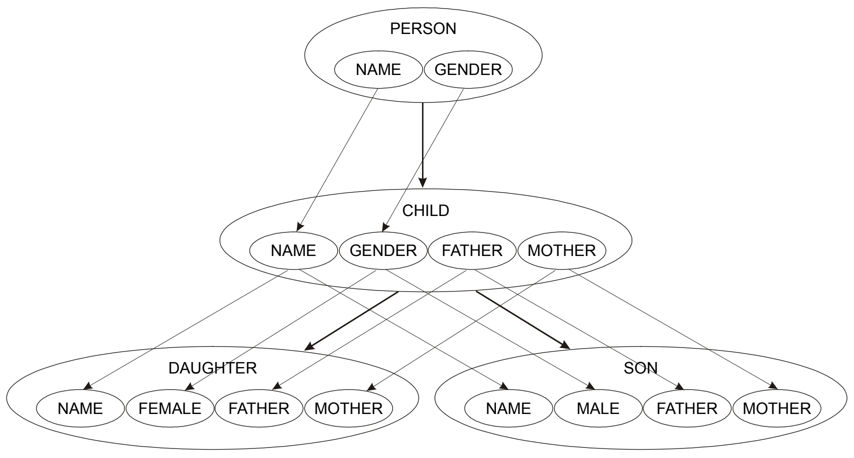


Figure 2.8: The Specialization Hierarchy Including Concept Mappings

The **ISA** occurrences that are describing the respective component mappings are defined by corresponding concept references, e.g. **ISA:**GENDER_MAPPING:=(2,(DAUGHTER[2], CHILD[2])) denotes the mapping between the GENDER component of CHILD and the FEMALE component of DAUGHTER. In the graphical EOS notation, such mappings can either be depicted by arrows as shown in Figure 2.8, or directly as independent concept bubbles containing reference components. Figure 2.9 demonstrates how concept references are represented within component bubbles by referring to **ISA:**GENDER_MAPPING that is defined as just mentioned.

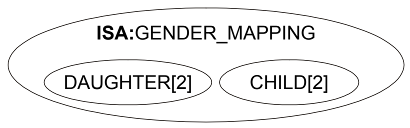


Figure 2.9: Graphical Representation of Concept References

In certain cases (e.g. when modeling epistemological concepts, see Section 3.2) concepts may contain complete ontological substructures as their components, i.e. these components refer to concepts and their corresponding **ISA** occurrences. Then, an alternate presentation form of the encompassing concept is to depict its components as subgraphs in the same way an ontology graph is drawn. This is exemplified in Figure 2.10 for the concept CHILD_INFO that contains all concepts of Figure 2.7, including the respective **ISA** occurrences[29].

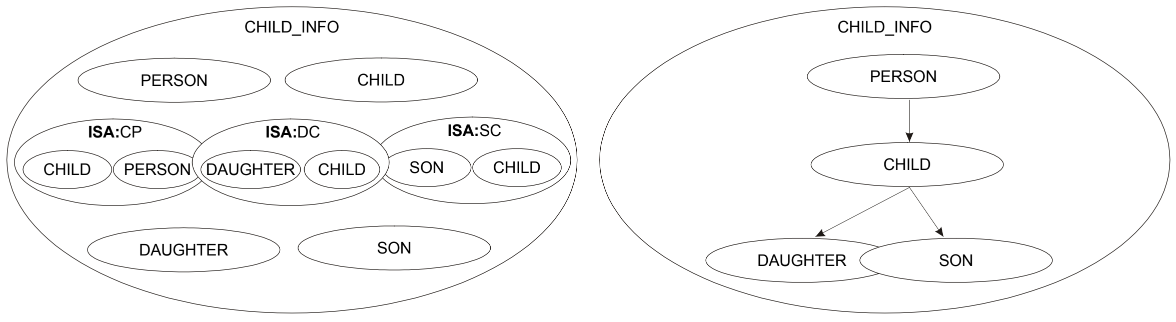


Figure 2.10: Two Alternative Representations of the Concept CHILD_INFO

This completes the basic syntax of the graphical notation for EOS ontologies. Whenever appropriate we will use this notation throughout this thesis for illustrating our examples. An extended example referring to points central to ontological design that were raised in this chapter will be presented in the following section.

2.4 Extended Example

This concluding section will present an extended example of a formal ontology. As with the previous examples we will refer to an ontology of the domain *Society*. Any ontology requires a most general, or graphically speaking, a fundamental root concept that all other concepts are reducible to. This concept, here as in the preceding sections, is EXISTENT, in concordance with the ontological perspective of metaphysical realism. Following metaphysical realism, the basic structure of the formal ontology O_3(*Society*) is as depicted in Example 2.13a.

[29] Note that the graph representation of concept components does not portray the correct component order. However, if this way of representing the component structure of a concept is chosen, the emphasis will anyhow lie in the ontological view on the specialization hierarchy. An EOS user interface may offer functionality to switch between these two views.

Example 2.13a: A Simple Ontology of Metaphysical Realism as Part of O_3(*Society*)

EXISTENT := ( )
UNIVERSAL := ( )
PARTICULAR := ( )
QUALITY := ( )
RELATION := ( )
ABSTRACT_PARTICULAR := ( )
CONCRETE_PARTICULAR := ( )
MATERIAL_PARTICULAR := ( )
IMMATERIAL_PARTICULAR := ( )

ISA:isa1 := (2, (UNIVERSAL, EXISTENT))
ISA:isa2 := (2, (PARTICULAR, EXISTENT))
ISA:isa3 := (2, (QUALITY, UNIVERSAL))
ISA:isa4 := (2, (RELATION, UNIVERSAL))
ISA:isa5 := (2, (ABSTRACT_PARTICULAR, PARTICULAR))
ISA:isa6 := (2, (CONCRETE _PARTICULAR, PARTICULAR))
ISA:isa7 := (2, (MATERIAL _PARTICULAR, PARTICULAR))
ISA:isa8 := (2, (IMMATERIAL_PARTICULAR, PARTICULAR))

The corresponding ontology graph G_O of the formal ontology of Example 2.13a is shown in Figure 2.11. Concepts are graphically represented as ovals that form the nodes of the ontology graph. The vertices of the graph are simple arrows pointing from general concepts to their specializations as determined by the **ISA** occurrences of Example 2.13a. Note that the names of child concepts of PARTICULAR are abbreviated, e.g. we use 'ABSTRACT_P.' instead of 'ABSTRACT_PARTICULAR' for keeping the representation of the graph compact.

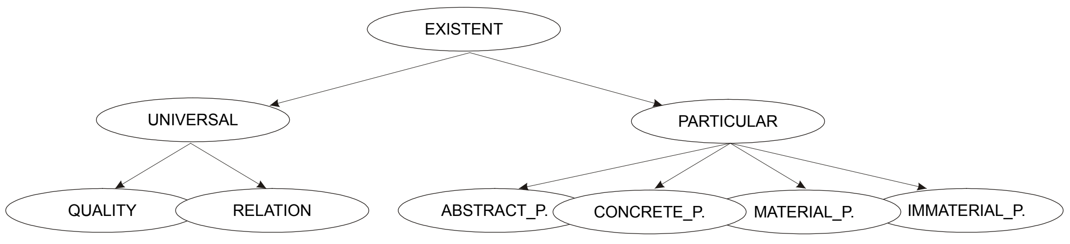


*Figure 2.11: Basic Ontology Graph of O_3(*Society*)*

With the basic outline, all consecutively defined concepts of O_3(*Society*) can be specified and categorized accordingly. This way, the domain *Society* is modeled with respect to metaphysical realism as described in section 2.1.1. Note that this is not at all a requirement of Concept Theory. O_3(*Society*) need not necessarily contain any of the concepts mentioned so far, as long as it is syntactically a valid formal ontology. The semantics of Concept Theory have been motivated by referring to metaphysical realism, yet what a particular ontology is expressing may very well disregard or even contradict this philosophical theory. Accordingly, we could also have chosen, e.g. SOCIETY as the fundamental (root) concept of O_3(*Society*) and developed the ontology from there on.

Example 2.13b: Some Universals of the Domain *Society* represented in O_3(*Society*)

COLOR := ( )
BLUE := ( )
ISA:isa9 := (2, (COLOR, QUALITY))
ISA:isa10 := (2, (BLUE, COLOR))

FAMILY := (3, (MOTHER, FATHER, CHILD))
ISA:isa11 := (2, (FAMILY, RELATION))

PERSON := (2, (NAME, GENDER))
ISA:isa12 := (2, (PERSON, RELATION))

FEMALE_PERSON := (2, (NAME, FEMALE))
MALE_PERSON := (2, (NAME, MALE))
CHILD := (4, (NAME, GENDER, FATHER, MOTHER))
ISA:isa13 := (2, (FEMALE_PERSON, PERSON))
ISA:isa14 := (2, (MALE _PERSON, PERSON))
ISA:isa15 := (2, (CHILD, PERSON))

MOTHER := (3, (NAME, FEMALE, CHILD))
FATHER := (3, (NAME, MALE, CHILD))
DAUGHTER := (4, (NAME, FEMALE, FATHER, MOTHER))
SON := (4, (NAME, MALE, FATHER, MOTHER))
ISA:isa16 := (2, (MOTHER, FEMALE_PERSON))
ISA:isa17 := (2, (FATHER, MALE_PERSON))
ISA:isa18 := (2, (DAUGHTER, FEMALE_PERSON))
ISA:isa19 := (2, (DAUGHTER, CHILD))
ISA:isa20 := (2, (SON, MALE_PERSON))
ISA:isa21 := (2, (SON, CHILD))

Note: Some components of concepts, e.g. GENDER, and according **ISA** occurrences for depicting roles and occupants, e.g. **ISA:**isa21a:=(2,(SON[2],CHILD[2])), are not listed, here, because this would lengthen our example immoderately. A complete enumeration of O_3(*Society*) must certainly contain these concepts.

The universals mentioned in Example 2.13b illustrate mainly family relations of people. Figure 2.12 depicts the correspondingly expanded ontology graph of O_3(*Society*). Concept components have been omitted, here, for better readability.

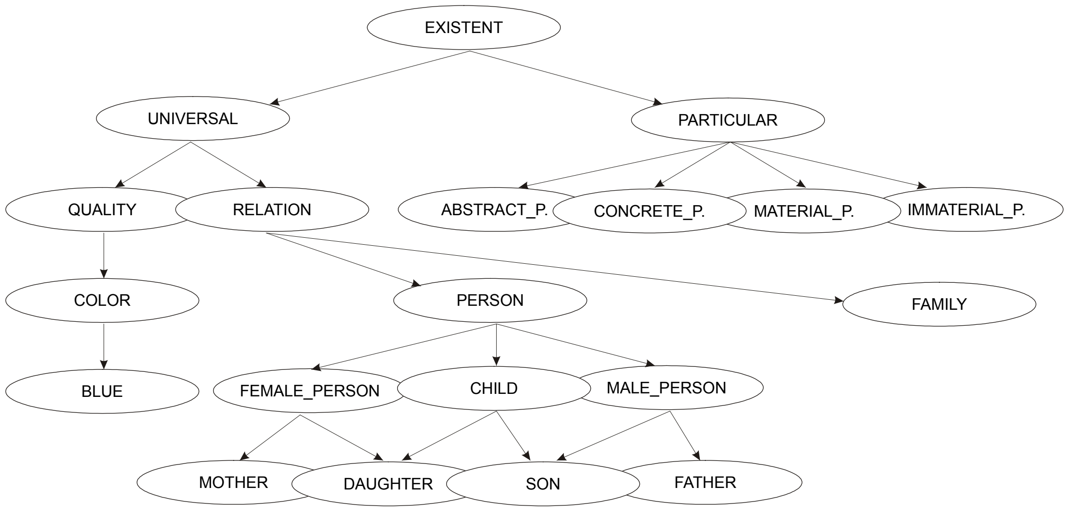


*Figure 2.12: Ontology Graph of O_3(*Society*), extended by Universals*

Finally, Example 2.13c presents some particulars of O_3(*Society*) that are occurrences of SON and FAMILY. Note that metaphysical realism describes humans and events as concrete, material particulars. This is incorporated into O_3(*Society*) by using the conceptual universals PARTICULAR_SON and PARTICULAR_FAMILY.

Example 2.13c: Some Particulars of the Domain *Society* represented in O_3(*Society*)

Before defining John and Peter as *sons*, or Smith_family as a *family*, we introduce two intermediate universals representing their metaphysical status:

PARTICULAR_SON := (4, (NAME, MALE, FATHER, MOTHER))
ISA:isa22 := (2, (PARTICULAR_SON, SON))
ISA:isa23 := (2, (PARTICULAR_SON, CONRETE_PARTICULAR))
ISA:isa24 := (2, (PARTICULAR_SON, MATERIAL_PARTICULAR))

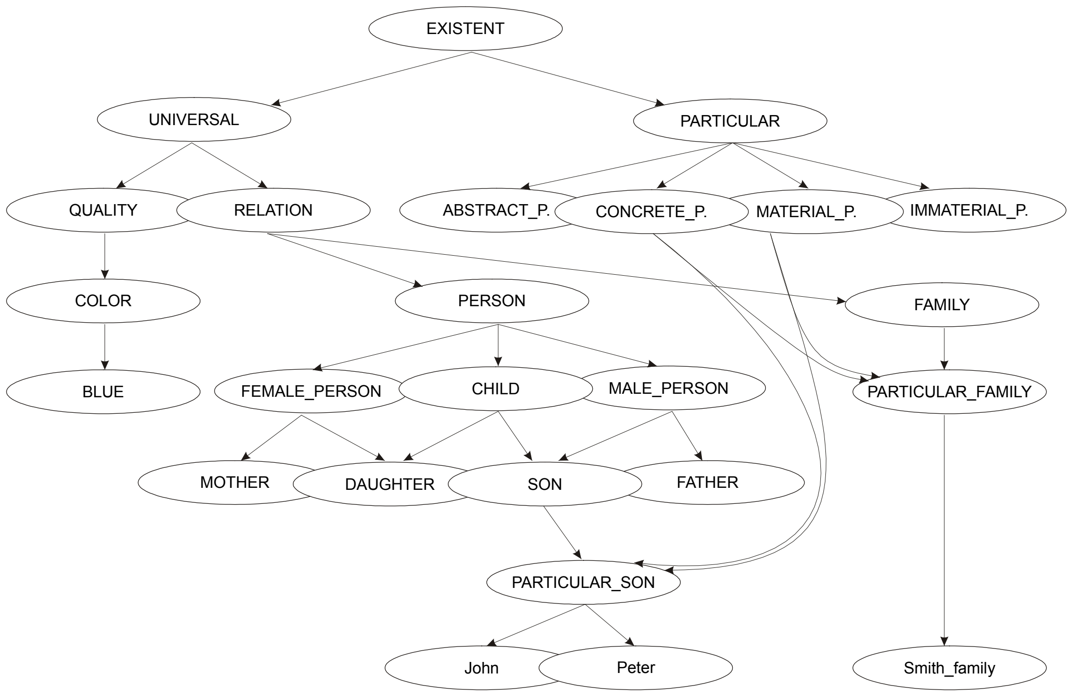
PARTICULAR_FAMILY := (3, (MOTHER, FATHER, CHILD))
ISA:isa25 := (2, (PARTICULAR_ FAMILY, FAMILY))
ISA:isa26 := (2, (PARTICULAR_ FAMILY, CONRETE_PARTICULAR))
ISA:isa27 := (2, (PARTICULAR_ FAMILY, MATERIAL_PARTICULAR))

The concept definitions of particulars John, Peter and Smith_family thus are:

John := (4, (John_name, M, Barry, Mary))
Peter := (4, (Peter_name, M, Scott, Beth))
Smith_family := (3, (Mary, Barry, John))
ISA:isa28 := (2, (John, PARTICULAR_SON))
ISA:isa29 := (2, (Peter, PARTICULAR_SON))
ISA:isa30 := (2, (Smith_family, PARTICULAR_FAMILY))

Note: These definitions of conceptual particulars exemplify the important difference between the semantics of the domain model represented by O_3(*Society*) and the semantics of Concept Theory itself. John is reducible to the concept PARTICULAR but this does not yet account for it to be a *particular* in terms of Concept Theory. From a conceptual point of view, PARTICULAR is simply a bare concept of equal status to other bare concepts like RELATION. Only the different writing of 'John' (using lowercase letters) marks this concept as a particular according to Concept Theory.

Figure 2.13 shows the complete ontology graph of O_3(*Society*) to the extent it has been presented in the previous examples.



---

*Figure 2.13: Ontology Graph of O_3(*Society*), extended by Universals and Particulars*

This concludes the discussion of the formal foundation of EOS, Concept Theory, a novel formalism for designing valid formal ontologies, knowledge representations based on the semantics of concepts. Concepts are formal constructs for defining and referring to any existents, abstract or concrete objects of reality. We discussed the syntactical and semantic features of Concept Theory by examining different sets and kinds of concepts. Special emphasis has been put on the semantics and properties of the fundamental concept **ISA** that represents specialization and generalization in Concept Theory. Subsequent chapters will use the ontological basis of Concept Theory for defining the epistemological characteristics of the EOS framework and the EOS system.

Chapter 3 EOS Epistemology

This chapter will present the application environment of EOS ontologies. As a motivation we will first discuss a general architecture for knowledge processing systems and subsequently develop the EOS framework for ontology-based knowledge processing that is formally grounded in Concept Theory as introduced in Chapter 2. This discourse is used for identifying the different tasks of computerized knowledge processing, and for motivating a consistent approach covering the questions immanent in these tasks. While the previous chapter developed a sound formalism for *defining* formal ontologies, we will now concentrate on the entailments of *utilizing* the knowledge ontologies are representing. This also marks an important shift regarding the understanding of knowledge. Ontologically, knowledge about existents can be categorized and formalized, i.e. molded into a particular formal representation. Examining knowledge epistemologically, as we will do in the following sections, however means to explore practical ways of employing it. Specifically, our interest lies in explaining the implications of this epistemological perspective in terms of knowledge processing systems where 'active' epistemological knowledge is used by a computer system in order to perform tasks on the basis of 'static' ontological knowledge.

3.1 Knowledge Processing

Knowledge processing can be understood in two separate ways. One is to stress the *purpose* of a knowledge processing system to work on input data, such as text, audio or video, that is expected to carry information, or knowledge, of some kind (see also Section 1.2.1). In this sense, knowledge processing entails dealing with knowledge as facts encoded in input data, and is thus foremost occupied with decoding and extracting this knowledge. Commonly, this knowledge is expected to pertain to a domain that can be determined in advance, which implies that the system works better if it possesses a formal understanding of the domain, i.e. ontological knowledge. Knowledge processing in this respect can apparently be interpreted as 'processing (knowledge in) data on the basis of ontological knowledge'. The second way of approaching knowledge processing is to concentrate on the *behavior* of a knowledge processing system. Processing data of any kind (including ontological knowledge) requires certain strategies, application semantics, that define the manner in which data has to be treated in order to produce some desired result. The particular behavior of such a system is generally determined by a set of processing instructions, the system's epistemological knowledge, that is exercised on the data. Knowledge processing in this second sense may therefore be paraphrased as 'processing data using epistemological knowledge'. We put emphasis on this distinction between ontological and epistemological knowledge, which may not be ignored when discussing knowledge processing.

Note that these two perspectives on knowledge processing are by no means contradicting each other. On the contrary, it is only feasible to regard a knowledge processing system as one that is 'processing data on the basis of ontological knowledge using epistemological knowledge'. In fact, this definition mirrors our understanding of knowledge processing that is recognizing both perspectives on knowledge equally. Specifically, our approach promotes epistemological knowledge processing on the basis of EOS ontologies. The motivations and implications of this decision will be the central theme of the following sections. With this program in mind we will subsequently approach and clarify the complex undertaking of knowledge processing from a theoretical point of view. Section 3.1.1 presents a short outline of a theoretical perspective on knowledge processing, which will be used for concretizing the impact of an epistemological treatment of knowledge. These results are then motivating our assessment of the general framework for knowledge processing systems as presented in Section 3.1.2 and the discussion of the EOS framework for knowledge processing in Section 3.1.3.

3.1.1 Philosophical Perspective

In this section we will briefly sketch some basic positions of Epistemology, the philosophical theory of knowledge, and draw conclusions for the task at hand, computerized knowledge processing using formal ontologies. Epistemology tries to answer the questions

- 'What is knowledge?' and, subsequently,
- 'How can knowledge be acquired?'

(cf. [70]). We will concentrate on epistemological arguments brought forward in reply to these questions as they are valid for any in-depth consideration on dealing with knowledge practically. Arguments on determining the nature of knowledge help structuring the task of knowledge processing, while a general discussion on ways of acquiring knowledge gives suggestions on how to implement this task.

Note that this epistemological perspective on knowledge differs from the ontological understanding of knowledge in Chapter 2. As already pointed out in Section 1.2.1 we can, and must, regard knowledge ontologically *and* epistemologically. The ontological perspective of Chapter 2 has treated knowledge as facts about reality that may be analyzed and categorized according to metaphysical theories, or according to a knowledge representation formalism such as Concept Theory. Hence a formal ontology is a formalized body of knowledge (pertaining to a domain), i.e. ontologies *represent* knowledge, they are concrete products of modeling processes that involve human expertise in the respective domains. How, on the other hand, knowledge may be generally *utilized* is not within the scope of an ontological viewpoint. However, this pertains to the epistemological perspective on knowledge, which is the subject of the current chapter. Using knowledge appropriately presupposes a concrete notion of knowledge and a basic understanding of the modalities of acquiring it – and these are precisely the central topics Epistemology is dedicated to.

In search of an adequate response to the first question about the nature of knowledge (*'What is knowledge?'*) Plato proposed to specify it as 'justified true belief' (see also section 2.1.1), a definition that is still important because of its theoretical significance. According to Plato, all three parts of this definition express necessary conditions for knowledge. Even though some modern philosophical conceptions of knowledge deny the necessity of one or two of these parts, their proponents still have to argument along the lines of this definition and explain the validity of their own approach with reference to 'justified true belief':

- *Belief.* A belief may be any idea or thought that can be expressed in a statement, i.e. beliefs can be articulated verbally or in written form, be it in a natural or formal language. Consequently, knowledge representations, in particular formal ontologies, when regarded epistemologically are collections of beliefs. That is to say that any concept of Concept Theory, actually states a belief, e.g. the concept **ISA:**AF:=(2,(APPLE, FRUIT)) conveys the belief that an *apple* is a *fruit*, and the concept FAMILY:=(3,(MOTHER, FATHER, CHILD)) implies that *families* consist of *mothers*, *fathers* and *children*.
- *Truth*. Beliefs may be incorrect, e.g. **ISA:**AM:=(2,(APPLE, MAMMAL)) would imply that an *apple* is a *mammal*, which is a false statement. In fact, **ISA:**AM is a syntactically correct concept, yet for it to count as epistemological knowledge its semantics have to be accurate as well[30]. Philosophers, here, are demanding objective truth of beliefs that are anchored in an ontological reality.
- *Justification*. A true belief is only seen as knowledge if the believer can give an account that justifies it correctly, i.e. in terms of objective reality. Various theories on justification permissible in this sense have evolved and lead to different modes of justifying beliefs:
  - *Foundationalism* holds that there are some beliefs that are fundamentally true and therefore *independent* of their relationship to other beliefs. These other beliefs, then, are justified *because of* their relationship to other beliefs.
  - *Coherentism* denies the existence of fundamental beliefs, i.e. it recognizes only one kind of beliefs. Justification is therefore regarded as an act of proving or refuting a belief by referring to a body of beliefs of equal epistemological status.

  Foundationalism and coherentism are *internalist* theories of justification, i.e. they relate to the inner awareness of a thinker. Important to internalist theories are the conditions that have to be fulfilled in order to guarantee valid justification for the thinker. Justification itself is then a deductive process based on the thinker's conviction, be it foundational or coherent. In Concept Theory, an ontology is a set of concepts that can be regarded analogously to true beliefs. Next to the simple domain concepts as presented in Chapter 2, Concept Theory also knows epistemological concepts which will be the central theme of Section 3.2. On the basis of these fundamental epistemological concepts other concepts can be tested, queried or generated. Therefore, Concept Theory is foundational concerning the internalist perspective.

  On the other hand, *externalist* theories of justification concentrate on the thinker's evidence, e.g. produced by perception. Sense-data may be false, as optical illusions or misinterpretations of distant voices illustrate, and externalist theories are arguing with respect to this phenomenon:

  - *Reliabilism* concentrates on the manner how a belief comes to existence. The idea is to ground justification in the *reliability* of belief-forming processes, i.e. a belief is regarded as justified if it has been produced by a reliable procedure. Concerning sense-data a reliable procedure for determining the color of an object would be, for example, to take a look at the object in bright daylight as this is the natural condition for colors to appear correctly (assuming that the observer also has a normal eyesight).

[30] Note that **ISA:**AM:=(2,(APPLE, MAMMAL)), according to the semantics of Concept Theory, is only a (valid) concept if it refers to a corresponding metaphysical existent – because concepts, per definition, actually are references to existents. In our outline of metaphysical realism we have not restricted reality to consist of *true* existents alone, in fact, practically all realist theories allow for 'false' existents, universals that are ideas which are not, or cannot be instantiated by particulars, e.g. *unicorns* or *square circles* are mere intellectual pastimes, yet still universals.

Externalist theories thus interpret justification as a reliable way of producing beliefs while internalist theories analyze adequate preconditions for deducing the truth or falsity of beliefs.

As the preceding paragraphs have indicated, the second epistemological question on how to acquire knowledge (*'How can knowledge be acquired?'*) is closely connected to justifying beliefs. We can identify two fundamental ways of acquiring knowledge, namely by deducing, or proving, it from already existing knowledge, or by extracting it from fallible sources in a reliable way. The latter option is directed towards an objective perspective on reality while the former is grounded in syllogistic arguments [10], or more generally, in formal logic.

These considerations have exemplified a basic notion of knowledge, along with crucial distinctions for treating, and therefore processing, knowledge. The facts, or beliefs, that constitute knowledge must satisfy a high quality standard (truth) that can be obtained in several ways. On the one hand, knowledge can be proven and inferred, which requires a deductive basis as internalist theories are stressing. On the other hand, externalist theories show that appropriate ways of judging reality need to be observed. Applying these observations to a technical view on computerized knowledge processing we can formulate some fundamental conclusions:

- *Internal View*. Any knowledge processing task exhibits two different kinds of knowledge, a basic set of true facts (ontological knowledge) and a set of rules how to utilize these facts (epistemological knowledge). Thus, ontological knowledge represents the essential basis of epistemological processes (cf. internalist theories). Ontological knowledge may be an integral part of a knowledge processing system, i.e. belonging to its algorithmic implementation, or given to the system as an input. In the latter case, knowledge representations, in particular formal ontologies, can be used to provide the system with the ontological knowledge they are formalizations of. Note that in terms of the internal view, a knowledge processing system is necessarily attributing truth to all facts of ontological knowledge, as any epistemological activity grounds in an ontological basis – inference needs facts (or beliefs) to draw from. For knowledge processing systems this means that all concepts of a formal ontology are regarded as being true (per definition), which directly follows from the internalist perspective on concepts. Concepts *are* the ontological basis, i.e. justified *true* belief.
- *External View*. The main focus of knowledge processing systems is not restricted to internal deductive processes but focuses on the utilization of knowledge for a particular purpose, e.g. intelligent information retrieval [4] or document management [21]. Thus the system accepts data from external sources that has to be processed according to its internal ontological and epistemological knowledge. The basic assumption for knowledge processing in this context is that the input data carries knowledge that can be extracted, stored and analyzed by reliable procedures. The definition of such reliable procedures is naturally easier for data that may be interpreted as structured information, which allows for exploiting its syntactical features. Still the 'knowledge' extracted in this way has a lower status than the ontological and epistemological basis of the system, and from that point of view it is at this stage 'insecure knowledge' – epistemologically speaking, insecure knowledge comprises beliefs that have not yet been justified internalistically but only externalistically, through suitable procedures. Depending on the reliability of these procedures, insecure knowledge may fully qualify for justified knowledge or not. We may certainly assume that there is a reliable procedure for passing EOS ontologies to an EOS system, i.e. EOS ontologies are encoded in a way known to the system. Yet, this assumption cannot be maintained for arbitrary input data, e.g. original Web documents. Parsing these documents for instances of implicitly contained ontology concepts yields results that may be fallible,

depending on whether the data has been interpreted correctly or not. However, this insecure knowledge still leaves the option for proving or falsifying it according to the internal, and thus secure, knowledge of the system.

- *Epistemological Limits*. Knowledge processing is dependent on ontological knowledge, the basis of epistemological processes, and has therefore no possibility of any semantic evaluation of ontological knowledge. In other words, truth (or semantical soundness) of a formal ontology is a premise that cannot be proven, only its *syntactical* correctness. Hence, the semantics inherent in the syntactical structure of formal ontologies is an important criteria for their practical applicability. For that reason, Concept Theory has been designed with a clear semantic foundation based on that of the fundamental concept **ISA**. This conceptual relation is being used as a constructive element for defining the internal graph structure of EOS ontologies. The properties of **ISA** account for the syntactical definition of instances and, ultimately, that of a valid ontology. But these semantics incorporated into Concept Theory certainly cannot guarantee truth. It is a matter of modeling and validation techniques to ensure semantic correctness of formal ontologies.

  Knowledge processing relies on epistemological knowledge, the semantics of epistemological processes. Again, these logical rules cannot be proven valid in terms of truth pertaining to an objective reality. Even the basic requirement that they form a body of rules free from any semantical contradictions is outside the scope of computerized knowledge processing in its strict sense, as these systems are merely applying, not validating them. Defining these rules is, like modeling domain knowledge, an intellectual process of human experts (or based on a formal procedure, which in return has to be designed by humans). What we can ultimately expect from knowledge processing is thus delineated by the ontological and epistemological knowledge it has at its disposition, while the validity of both kinds of knowledge is a necessary presumption. It is within these limits where knowledge processing has to be examined.

The epistemological perspective on knowledge has led to some theoretical insights that can be applied to knowledge processing in general. Regarding the semantics of knowledge processing, these always depend on two kinds of knowledge, ontological and epistemological facts. On this basis computer systems can extract and process knowledge from external sources. Before integrating externally received information into the internal body of knowledge, its validity has to be secured. This can be done either by reliable methods of input interpretation or by deducing its correctness from ontological knowledge. Information that cannot be validated in either way may be regarded as ‘insecure knowledge’ that has a different status than the secure ontological and epistemological basis underlying knowledge processing. Unless it can be justified, e.g. on the basis of newly acquired knowledge, it may not be used actively during knowledge processing, i.e. deduction grounded in insecure knowledge may produce unwanted and false conclusions.

We will refer to this theoretical outline during our discussion on knowledge processing. In particular, the preceding observations allows for identifying the different areas of epistemological processes:

- *Knowledge Acquisition*. Knowledge can be acquired by a knowledge processing system from external sources through reliable procedures. In this sense, knowledge acquisition can be equated with extracting knowledge from input data.
- *Knowledge Generation*. Once knowledge has been acquired from external sources it can be integrated into the internal ontological and epistemological knowledge representation of the system. Deductive processes can then be utilized to increase this ontological basis.

Hence, knowledge generation means applying epistemological rules in order to produce new ontological knowledge.

- *Knowledge Retrieval*. The goal of knowledge processing is to provide external parties, human users or application systems, with information. For that reason, knowledge processing systems must provide facilities for querying the internally stored knowledge. Knowledge retrieval is therefore occupied with processing queries.

Knowledge acquisition is directed towards the external view on knowledge processing systems. Knowledge generation and retrieval both touch on the internal view. Defining formal descriptions for the realization of these three areas of epistemological processes is the main focus of this chapter. In fact, all of Section 3.2 may be read as a coverage of these topics central to knowledge processing as such. Beforehand, the following sections will locate knowledge acquisition, generation and retrieval within knowledge processing systems seen from the information science perspective. Section 3.1.2 will motivate a general architecture for knowledge processing systems, and section 3.1.3 will present the EOS framework, which is a refinement of the general architecture.

3.1.2 Knowledge Processing in Information Science

Knowledge processing in computer systems comprises all tasks and methods concerned with employing domain knowledge and application semantics for enabling a desired system performance, e.g. in the fields of intelligent knowledge management, information search, electronic and web commerce, to name but a few. In this section we will present a commonly agreed upon design for knowledge processing systems and point out its shortcomings [76]. We will later use this discussion in Section 3.1.3 for motivating a novel and undiluted, epistemological perspective on knowledge processing and present how it may be implemented by the EOS framework.

In order to motivate our considerations we will sketch an exemplary knowledge processing system that may be used for intelligent information retrieval from heterogeneous information sources such as the Web, interpreted as a vast knowledge base. This choice enables us to demonstrate ontological as well as epistemological aspects of knowledge processing in an illustrative way. Within the Web environment we find an area where all characteristics of knowledge processing can be immediately exemplified. Recurring to the definition of a knowledge processing system as one that is processing data that carries semantics on the basis of ontological knowledge using epistemological knowledge, we can identify

- data carrying semantics: Web documents (of a certain domain)
- ontological knowledge: a domain model for the intellectual contents of these documents
- epistemological knowledge: processing strategies, e.g. for querying Web documents.

Representative systems falling into this category of knowledge processing systems are [30] and [53]. The internal domain model can be influenced or completely defined by a suitable knowledge representation, in advanced systems typically a formal ontology. This ontological knowledge describes some domain of interest and is used to extract and evaluate domain specific information from external sources, e.g. Web pages. The information acquired in this way is subsequently available as part of the ontological knowledge of the system to all internal processes. It can therefore be stored as such persistently, e.g. in a knowledge base attached to the system, and may from this point on serve as a repository for providing external applications with data, or for answering ad-hoc user queries. Computer systems that use an explicit modeling of domain knowledge and consequently accept this knowledge representation as an input, generally exhibit a basic layout as depicted in Figure 3.1. From a functional point of view such a knowledge processing system will accept input data and process it according to the internal knowledge representation. As indicated, the system returns structured information that may, on its own part, serve again as a new input to the system.

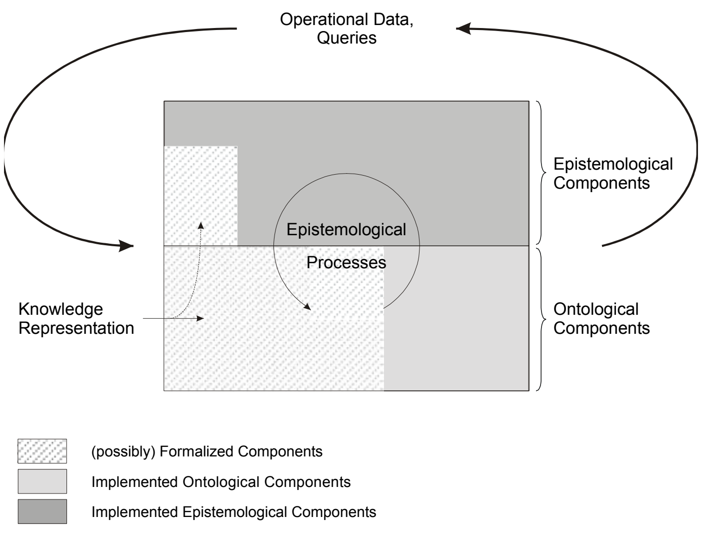

Operational Data,
Queries
Epistemological Components
Epistemological
Processes
Knowledge Representation
Ontological Components
(possibly) Formalized Components
Implemented Ontological Components
Implemented Epistemological Components

Figure 3.1: A General Outline of Knowledge Processing Systems

We will discuss the general system architecture presented in Figure 3.1 with reference to the theoretical results of Section 3.1.1. The internal view on knowledge processing systems is dominated by the distinction between ontological and epistemological knowledge that influence the behavior of such systems. In that respect we will examine the internal components of knowledge processing systems. The external view, on the other hand concentrates on communicating with external systems or humans, i.e. questions arising here are concerning input and output data. The different system components can be explicated as follows:

- *Internal System Composition.* From a theoretical point of view, knowledge processing systems are internally divided into *ontological* and *epistemological components* that support *epistemological processes*.

  Ontological Components. For any knowledge processing task the system needs some basic information, domain knowledge, in order to perform it. Domain knowledge can be an integral part of the implementation of the system, or, to a certain extent, given to it as a formal knowledge representation. Other ontological components are knowledge about data formats, their syntactical features and semantics, etc. We put emphasis on the fact, that it is certainly not possible to provide all ontological facts as an input to the system, as any computer system needs at least some algorithmical ontological basis for interpreting the data it accepts. The extent to which a knowledge processing system relies on an externally provided knowledge representation may vary from system to system. Figure 3.1 shows the difference between system immanent ontological knowledge and externally supplied information by a difference in color. The share of system immanent knowledge is shaded gray.

  Epistemological Components. The ontological basis is being used according to guidelines, axioms or rules, which can be domain or task dependent (e.g. semantic implications of domain entities, instructions for handling data formats), as well as independent of the application area (e.g. rules for logical deduction). These system components form the epistemological foundation of a knowledge processing system and are usually integrated into the implementation of knowledge processing systems. As graphically indicated in Figure 3.1, some advanced knowledge processing systems (cf. [33]) are also accepting simple epistemological facts on domain knowledge in the form of logical axioms as part of knowledge representations. The application of these axioms to knowledge processing tasks, i.e. the main share of epistemological knowledge, is then again part of the algorithmical realization of these systems.

Epistemological Processes. The reasoning capacities of the system utilize the ontological and epistemological knowledge of the system and produce structured information from the previously received input data. Inference semantics and all associated mechanisms for processing a knowledge representation instance are integral parts of the system. Naturally, epistemological processes are used within the system in order to extract and deduce knowledge. This knowledge can then be formalized and issued as a return value for queries, and possibly reenter (presumably modified) the system again. On the other hand, inferred knowledge may also be integrated into the ontological basis and this way become an explicit part of the system itself.

- *Input Data*. We can identify several kinds of input data serving different purposes, *operational data* that has to be processed by the system, a *knowledge representation* that may be integrated into the system itself, and *queries* against the system.

Operational Data. Any type of documents (e.g. structured or unstructured text files, graphics, audio, video, etc.) containing information that corresponds to the internally held ontological knowledge is regarded as operational data. For performance reasons it may be stored and indexed separately, e.g. inside a knowledge base attached to the system. Note that, here, we make no assumption on whether the semantic content of operational data is eventually made part of the ontological basis of the system, or not. Both variants are possible, depending on the main focus of the system.

Knowledge Representation. Essential to the overall performance of a knowledge processing system is its ontological knowledge. Parts thereof, in particular domain knowledge, can be formalized into a knowledge representation that models the application area of the system. The knowledge representation is held persistent within the system and serves as a basis for advanced tasks such as epistemological processes. As this framework is intended to present an outline of knowledge processing systems applicable *in general*, we do not stipulate a specific type or format of knowledge representations. After all, as pointed out in Section 2.1.2, the majority of formal ontologies used in current systems are not meeting the quality standards we are endorsing with EOS ontologies. In many cases, the knowledge representations used do not contain any epistemological semantics at all (which is indicated by the dotted arrow in Figure 3.1). Moreover, knowledge representations are an optional input, as the ontological basis of a system may very well be hard-coded into the system.

Queries. User interaction with the system, as well as automated processes, may trigger queries against the knowledge processing system. Queries can address the system's ontological knowledge itself, or be directed towards the structural or semantic content of operational data. Thus, queries may be document-related, e.g. '*return all addresses mentioned in document D*', '*return all London addresses mentioned in documents*', or domain-related, e.g. '*return entities that can have addresses*'. The inferential complexity of queries depends on the way knowledge is held internally within the system. If the ontological basis is held minimally, most queries will result in a series of inference operations (cf. deductive database systems). On the other hand, it is possible to materialize deduced knowledge as part of the ontological basis of the system. In this case, queries will consist merely of directly scanning this knowledge, as in conventional database systems.

- *Output Data*. The system returns structured information computed from operational data and/or the internal ontological basis. Return values from operational data can comprise e.g. specific information held within documents, or meta-information such as classifications and indices. Usually, knowledge processing systems use identical input and output formats, which makes it possible to pass an output produced by the system back into it.

As depicted by this general layout, the formalization of epistemological knowledge is almost entirely hard-coded into the algorithmical implementation of present knowledge processing systems. In fact, to the best of our knowledge, there is no knowledge processing system that goes beyond formalizing logical axioms on ontological knowledge, while a formal access to the actual *modes of application* of epistemological knowledge has not been tackled in any approach to knowledge processing so far [76]. Consequently, knowledge processing systems generally act as black boxes when it comes to the heart of their enterprise, namely processing knowledge. The semantics of knowledge processing, therefore, remain not only hidden but cannot be influenced at all. This leads to several drawbacks:

- Knowledge processing systems must be designed for very specific tasks and domains as the corresponding semantics may differ to a great extent, e.g. natural language processing and deduction on chemical data require very different application semantics. Note that this is neither a matter of format nor of the modeling paradigm applied. Of course two different domains can be modeled using the same ontology language, which ensures a uniform syntax and modeling guidelines. Yet, semantical implications of e.g. words and sentences are very different from the meaning of molecules – and this has to be observed when designing knowledge processing systems.
- For similar reasons exchanging knowledge representations among different systems poses serious problems. Again, foreign knowledge representations must, apart from modeling paradigms, comply with native application semantics of a given knowledge processing system in order to render it capable of processing it adequately. Even knowledge representations exhibiting a native syntactical makeup may be interpreted incorrectly as there is no direct coupling between the objects of the knowledge representation and its semantic impact. This accentuates a serious shortcoming of the utilization knowledge representations in common approaches. Conventional knowledge representations are only considering ontological facts, leaving epistemological semantics about the employment of these facts aside. This way, one and the same knowledge representation may very well be interpreted very differently from two distinct systems. Ironically, the lack of specifying epistemological knowledge within knowledge representations can therefore lead to a disagreement (among different systems) concerning the actual semantics of this representation – while the initial intent to introduce knowledge representations was to create *shared* understandings.
- Moreover, common systems are highly inflexible regarding conceptual changes of the knowledge representation used, i.e. only a restricted class (in terms of expressiveness, not limited to representation formats) of knowledge representations can be processed by a particular system. Increasing the expressive power of a knowledge representation that would require a modified system behavior must therefore result in a costly system redesign. The reason for this lies in the general flaw that common knowledge processing systems provide no access to their active epistemological components.
- Similarly, as epistemological processes rest inaccessibly hidden within these systems, the ontological knowledge held within a knowledge representation can only be used in a sin-

gle, very specific way, depending on the fixed system design. Respective changes tackling the system behavior, again, require extending the system functionality, which means adding or altering its existing implementation.

In summary, the lack of an adequate formalism that allows for modeling both, ontological and epistemological aspects of a domain, leads to knowledge representations that cannot provide enough semantics for their uniform employment. Knowledge processing systems must consequently suffer from this representational shortcoming which results in the common drawbacks as just mentioned. This is where the EOS approach offers a new direction. The EOS framework as presented in Section 3.1.3 combines ontological and epistemological components, and promotes according knowledge representations, EOS ontologies, that are apt to express both. The necessary extensions to Concept Theory for expressing epistemological facts in the form of EOS concepts will then be the subject of Section 3.2.

3.1.3 EOS Framework

Based on the preceding considerations we will now introduce the EOS framework for knowledge processing systems as shown in Figure 3.2. The EOS framework refines and extends the general layout of knowledge processing systems (cf. Section 3.1.2) by introducing different semantic layers to the internal knowledge representation.

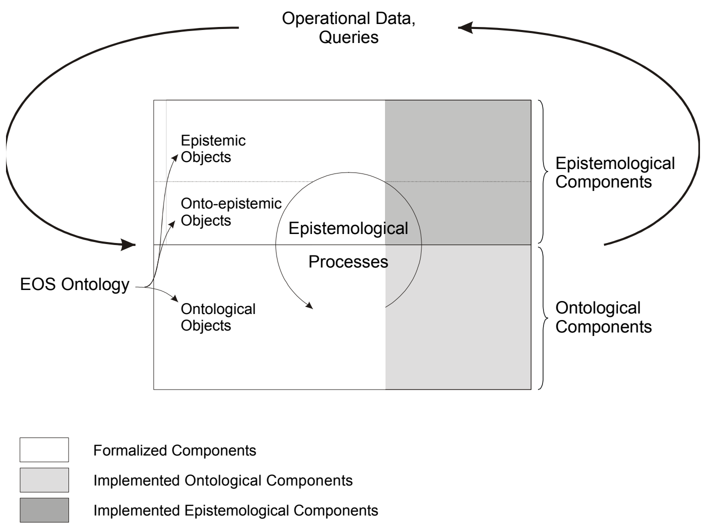

Operational Data, Queries
Epistemic Objects
Onto-epistemic Objects
Epistemological Processes
EOS Ontology
Ontological Objects
Epistemological Components
Ontological Components
Formalized Components
Implemented Ontological Components
Implemented Epistemological Components

Figure 3.2: The EOS Framework

The EOS approach regards a knowledge processing system as a highly parameterized software tool. An EOS system will consequently accept a comprehensive knowledge representation, in particular an EOS ontology, that provides it with the information necessary to manage its knowledge processing tasks. As indicated in Section 3.1.1, this involves ontological, as well as epistemological knowledge, where the latter has unfortunately been neglected by common knowledge representation methods. The EOS framework, on the other hand, allows for differentiating between several kinds of objects of ontological and epistemological knowledge that may be part of an EOS ontology. As portrayed in Figure 3.2, these kinds are elementary *ontological objects*, *onto-epistemic objects* that provide meta-information about other objects, and purely *epistemic objects* that refer to the employment of other objects of knowledge during knowledge processing. According to Concept Theory, all of these objects can be represented uniformly by concepts. In particular, this means that EOS ontologies consist of the static domain model, as well as application semantics, which is a novel approach to designing knowledge processing. In this way, epistemological knowledge can be modeled on the same terms as ontological knowledge, which leads to a seamless and complete formalization of a domain where the gap between application semantics and the formal domain model has been bridged. Moreover, epistemological processes that work on EOS ontologies are on the one hand regulated by epistemological knowledge, and on the other hand, they can also be executed on it, i.e. epistemic objects can define a system behavior concerning themselves. This underlines the powerful possibilities of Concept Theory as a modeling formalism. Its expressiveness grounds in the different kinds of objects of knowledge it comprises:

- *Ontological Objects of Knowledge*. Regarding a domain ontologically is to specify and categorize its existents. How this can be achieved was the central topic of Chapter 2 and led to the notion of EOS ontologies. We have shown that EOS ontologies are apt to describe existents using formal concepts. The semantics of an EOS ontology are expressed by its graph structure that is derived from the fundamental ontological relation *specialization*, represented by the concept **ISA**. Therefore, ontological domain knowledge in the EOS approach is being organized into a directed graph that mirrors the specialization hierarchy within the domain. All elementary nodes of this graph, i.e. all concepts mirroring this hierarchical domain structure, are ontological objects of knowledge. This is the ontological basis of the computer system on top of which processing semantics can be defined.
- *Onto-epistemic Objects of Knowledge*. The ontological basis of an EOS system must be interpreted in a way that complies with the correlations among the existents of the real-world domain that is represented by an EOS ontology. So far, the only semantics that have been incorporated into Concept Theory are that of the concept **ISA**. Technically, this means that an EOS system must treat occurrences of **ISA** differently from other concepts, i.e. the system knows the semantics of **ISA**. Concepts that are merely representing ontological objects of knowledge, on the other hand, possess as such no other semantics for an EOS system than that they are formalizations of existents, expressing some relation among their components. The semantical implications of conceptual relations cannot be conveyed exclusively on the basis of **ISA**. In order to exemplify this, we are referring to the previous examples that centered around modeling family relations. A specialization of the concept FAMILY could be GENETIC_FAMILY:=(3,(MOTHER,FATHER,CHILD)) that possesses the same internal component structure as its parent concept. Yet, the semantics of GENETIC_FAMILY are that there exist additional relations on its components, namely that *mother* and *father* are both *real parents* of their *child*. Thus, an according EOS ontology will also include concepts like IS_REAL_MOTHER:=(2,(MOTHER,CHILD)). What such an EOS ontology then has to be able to express is that all occurrences of GENETIC_FAMILY satisfy the condition that the *mother* of this *'genetic family'* is also the *real mother* of the *child*. Any such rule on concepts and the conditions it defines are onto-epistemic objects of knowledge. Rules and conditions themselves can, of course, be expressed by concepts (as any such correlation among existents is, metaphysically speaking, itself an existent). Nevertheless, the semantical status of the concepts **RULE** and **CONDITION** is different from that of purely ontological concepts, such as GENETIC_FAMILY and IS_REAL_MOTHER. Occurrences of these concepts must be interpreted by the system as specific rules and conditions, i.e. they express statements about other concepts, and carry as such epistemological knowledge directed towards a semantic specification of the concepts they are attributed to. In this sense, onto-epistemic objects complete the ontological domain model by providing domain-specific details to the ontological objects of knowledge. A concise introduction to the notion of conditions and rules is given in Section 3.2.2.

- *Epistemic Objects of Knowledge*. Ontological and onto-epistemic objects of knowledge describe the static domain model, i.e. they name existents and sketch their semantics in terms of conditions. The task of knowledge processing is to utilize this knowledge in order to enable a desired system performance. This comprises deducing new knowledge as well as answering queries, and mastering related processes. It requires epistemological knowledge to do so, i.e. instructions on what kind of behavior will yield valid computations within a certain domain. Processing instructions of this sort which are referring to ontological and onto-epistemic objects of a domain model, are epistemic objects of knowledge. In Concept Theory, concepts that are representing epistemic objects are called *laws*, occurrences of the concept **LAW**. Laws specify the fundamental patterns for knowledge retrieval, generation and acquisition. As such they are formalizations of problem solving methods (PSMs) (cf. [36], [58]) in knowledge processing systems. An example for a law on ontological concepts would be that from two particular instances is_real_Mother1:=(2,(MARY,John)) and is_real_Mother2:=(2,(Sue,MARY)) of the concept IS_REAL_MOTHER the knowledge processing system can generate, a new particular is_Grandmother1:=(2,(Sue,John)), an instance of the concept IS_GRANDMOTHER. Laws represent the epistemological domain model and define how the ontological domain model should be used during knowledge processing. Obviously, as with rules, occurrences of the concept **LAW** are treated differently by an EOS system, compared to ontological concepts. This means that the semantics generally needed for interpreting occurrences of the concept **LAW** have to be known to an EOS system in order to be able to execute the epistemological content of such a law. Laws and their application to knowledge processing will be covered in Section 3.2.3.

In order to motivate this distinction between different kinds of objects of knowledge we are referring, as an analogy, to a specific class of knowledge processing systems, namely relational database systems. Ontological objects of knowledge comprise universals and particulars of a domain of interest. The universals of a domain can be compared with the relational database schema that defines relations and thus domain entities, whereas particulars would represent the tuples, i.e. concrete instances of the domain entities. Onto-epistemic objects in this context would be, for example, integrity constraints on these relations. Epistemic objects of knowledge, on the other hand, pertain to the actual behavior of the database system, i.e. they would be located on the application programming side that uses the services of the database management system, e.g. for answering user queries in a desired way.

As described above, the EOS approach defines a detailed perspective on knowledge processing that allows for identifying and specifying different kinds of knowledge that render a computer system capable of managing the tasks involved. Our claim is that a great share of this knowledge can be formalized into an EOS ontology using ontological, onto-epistemic and epistemic objects of knowledge. With this distinction at hand the modalities of knowledge processing become visible, particularly advanced possibilities of using knowledge representations in areas that have been traditionally regarded as pertaining to the algorithmical realization of knowledge processing tasks. Conventional approaches are often limited to addressing ontological objects of knowledge alone. There are only a few systems that allow for modeling onto-epistemic objects of knowledge and none at all that uses formalized epistemic objects of knowledge [76]. Yet, these latter actually depict the core epistemological semantics that define the essential tasks of a knowledge processing system, namely the modes of knowledge acquisition, generation and retrieval within a certain domain of interest. Missing to formalize these semantics for using them as an input to the system therefore means not just a loss of generality but also hard-coding *domain knowledge* into the system implementation, although this is exactly what knowledge representations should help to avoid.

What sets EOS systems apart from conventional knowledge processing systems is thus a richer inventory for formalizing knowledge on the basis of Concept Theory, along with adequate mechanisms for interpreting this knowledge, which requires a thorough formal definition of the semantics expressed by the respective concepts. Compared to other approaches, the share of formalized knowledge that is passed to an EOS system has been extended significantly, in terms of scope and integrity. The scope of an EOS ontology encompasses both, ontological and epistemological knowledge. Its integrity concerning a domain of interest is reflected in the assertion that it covers the (ontological as well as epistemological) domain model more completely than other approaches. Unlike these approaches, an EOS system consequently does not possess any hard-coded domain knowledge at all, it must only be able to interpret concepts. It therefore expects a complete ontological domain model (ontological objects of knowledge), and accompanying application semantics (onto-epistemic and epistemic objects of knowledge).

In summary, epistemological knowledge comprises onto-epistemic and epistemic objects that define metadata about the static domain model of an ontology, foremost semantic processing rules. Again, this epistemological knowledge has to be understood and executed by software components but the invaluable benefit it provides when explicitly specified is a homogeneous formal description of the semantic and syntactic implications of such processes. Epistemic objects may be regarded as task templates describing these processes. Thus, they are expressing how to perform knowledge processing tasks for certain kinds of ontological objects using the semantics of onto-epistemic objects. For elucidating the different roles of these objects of knowledge we will present another example where new ontological objects are generated from already existing ones on the basis of laws and rules. We can assume particular instances of BROTHER_OF(2,(BROTHER,PERSON)) and CHILD_OF(2,(CHILD, PERSON)) satisfying the condition that the occupants of BROTHER_OF[1] and CHILD_OF[2] are identical, i.e. the occupant concept is a *brother* of a *person* and at the same time *father* of a *child*. From this we can deduce an instance of AUNT_OF, if BROTHER_OF[2] is a female *person*. In this example, BROTHER_OF, CHILD_OF and AUNT_OF all are ontological objects. The epistemic object would be a law, the concept GENERATE_AUNT_OF_INSTANCE that defines that the system may produce an instance of AUNT_OF according to AUNT_OF(2,(BROTHER_OF[2],CHILD_OF[1])). An onto-epistemic object in this context would be the rule setting the condition that BROTHER_OF[2] must be female.

As indicated by the preceding considerations, the scope of and the distinction between different objects of knowledge that make part of an EOS ontology implies far-reaching properties of an actual EOS system that is able to process these objects. The EOS framework addresses the underlying semantics of knowledge processing and, by doing so, pinpoints the share of knowledge that can and should rather be explicitly formalized and passed to an EOS system than be part of its implementation. This leads to a number of important features of the EOS framework:

- The EOS framework acknowledges a detailed view on knowledge processing along two dimensions. On the one hand, it tackles the distinction between ontological and epistemological knowledge. On the other hand it balances implemented versus formalized components. Regarding explicitly formalized knowledge, i.e. knowledge representations, the EOS framework differentiates between ontological, onto-epistemic and epistemic objects of knowledge that do not only model static domain knowledge but also its application in epistemological processes. Thus, it describes knowledge processing holistically, but independent from specific application areas or programming paradigms. It therefore offers a very detailed architecture for evaluating and comparing the capabilities of different knowledge processing systems.

- The EOS framework draws from the advantages of knowledge representation that allow for more flexible computer systems. By altering or exchanging the ontological basis, i.e. by feeding new ontologies to it, the system can be taught to process operational data of different format and domains. Unlike conventional systems, the EOS framework expressly allows for modeling epistemological knowledge in the form of epistemic and onto-epistemic objects. Hence, an EOS systems can also be tuned regarding the application side, i.e. concerning the utilization of static knowledge. As the EOS approach allows for defining dynamic uses of a static domain model, one and the same EOS system may be deployed for very different tasks on the same exact static ontological knowledge. This can be easily achieved by replacing an actual set of formalized epistemic objects (and, possibly, onto-epistemic objects) by another, while keeping the original ontological objects of knowledge. Therefore, EOS systems are also flexible concerning their behavior.
- Formal ontologies are explicit domain models that are human readable and that can also be communicated across different computer systems. Yet, current approaches to ontologies are concentrating on the *static* domain model. From a modeling perspective, the evident advantage of being able to explicitly model application semantics is that the *employment* of a static domain model becomes visible, and thus discussible. Concept Theory, as a modeling paradigm for ontologies, offers an abstract and declarative access to the semantics of a domain. A human domain expert can use Concept Theory for describing both, the static domain model and its semantics[31] – within the bounds of knowledge acquisition, generation and retrieval. In this way, a scientific community can communicate the objects of their domain plus their understanding of these objects. This understanding may differ from group to group within the community, and these semantics should be explicitly expressible within an ontology. Concept Theory provides the semantic primitives (epistemic objects) to do so.
- As a practical consequence, exchanging EOS ontologies across different knowledge processing systems is greatly facilitated. The common problem when transferring knowledge representations to a foreign system is that the static domain model must be interpreted correctly, i.e. according to the application semantics of the native system (that follows a specific understanding of the domain model). EOS ontologies include epistemic objects that render the epistemological impact of the static domain model explicit while remaining part of the ontologies (as their metadata). This way the shortcomings of introducing new ontologies into different environments can be overcome. In conclusion, processing ontologies and passing them over to other EOS systems poses no problems because the application semantics of the domain model objects is being supplied along with the ontology.

With the basic implications of the EOS framework in mind, the following sections are aimed at motivating an epistemological perspective on Concept Theory. In order to do so, we are referring to the principal areas of knowledge processing which have been introduced in Section 3.1.1, namely *knowledge acquisition*, *knowledge generation* and *knowledge retrieval*. We are taking a look at these fields with respect to our notion of formal concepts, which allows us to formulate requirements for modeling epistemological knowledge using concepts.

[31] A survey on different (to some extent machine-supported) methodologies for creating ontologies can be found in [51] and [57].

3.1.3.1 Ontological Outset for Knowledge Processing in EOS

The first step of translating application semantics into EOS concepts has already been taken by defining the three main areas of knowledge processing, knowledge acquisition, knowledge generation and knowledge retrieval. Our next step involves examining the semantics needed in order to master these tasks. Accordingly, the subsequent sections will discuss what (epistemological) knowledge the system needs in order to master them. Concretely, this means specifying new (epistemological) concepts that must be integrated into Concept Theory. These epistemological concepts will then be the basis of the actual behavior of an EOS system, i.e. such a knowledge processing system will interpret these concepts differently than mere ontological concepts. Ontological concepts draw all of their semantics from their position within the ontology graph and their internal structure. Epistemological concepts, on the other hand, have a predefined meaning that is known to the EOS system in the same sense that it knows the concept **ISA**. It is part of the algorithmical implementation of an EOS system to know how to process occurrences of **ISA**, while the definition of these occurrences is a matter of modeling a domain. The same applies to epistemological concepts. Their semantics must also be familiar to the system, which allows for a specific interpretation of their respective occurrences. Thus, in the following it is our interest to determine these semantics that will be used later on (in Section 3.2) for extending Concept Theory accordingly.

Beforehand, we will first briefly stress the semantics already at hand, namely those of the ontological basis as provided by EOS ontologies. Concepts, on the one hand, define relations as structured entities possessing components, and on the other hand, they are subject to relations when used as subconcepts by other concepts. This dualism has been used to establish the internal structure of EOS ontologies which are based on the semantics of the relation **ISA**. These are the only ontological semantics Concept Theory stipulates – besides the explicit accentuation of instances of **ISA** (using the concept name prefix **ISA:**) and the syntactical distinction between particulars and universals (denoted by the prefix **PARTICULAR:** for particulars, or, respectively, by using lowercase names for particulars). The semantic properties of **ISA** define an EOS ontology structurally as an ontology graph whose composition mirrors these properties (see Section 2.2.4). For example, from its position within the ontology graph one can immediately delineate the ontological kinds a concept belongs to, and, moreover, instances of concepts exhibit the same internal component structure as their parent concepts. Yet, the semantics of any concept other than **ISA**, in particular conceptual relations, are not determined ontologically, which complies with the understanding of ontologies as presented in 0.

What an ontological perspective has to accomplish is to specify the kinds, or categories, something, a concept, belongs to – and this is exactly what the semantics of specialization and generalization, i.e. of **ISA**, are supplying. Ontological knowledge is thus static in nature, and directed towards identifying and categorizing the existents of a domain. Conceptual particulars may vary over time but the overall ontological structure of a domain, i.e. the anatomy of its ontology graph, will naturally stay the same. This graph structure, along with its semantical implications, forms the basis for all dynamic processes that work on it in order to generate and query this ontological knowledge.

The epistemological perspective on knowledge processing requires the basic semantics for expressing guidelines for the behavior of respective computer systems concerning knowledge acquisition, generation and retrieval. As already motivated in our discussion of the EOS framework, the EOS approach promotes a solution that provides semantic constructs we call *conditions*, *rules* and *laws*. The exact connotations of these constructs and how they are integrated into Concept Theory will be the main focus of Section 3.2. Here, we will restrict ourselves to presenting the general intent of conditions, rules and laws.

- Conditions are the conceptual equivalent to truth statements, e.g. there are conditions that express logical conjunctions (A **AND** B), or disjunctions (B **OR** C), as well as combinations thereof (A **AND** (B **OR** C)).
- Rules are used to attribute conditions to concepts, e.g. for stating that certain conditions are valid for specific components of a concept.
- Laws, finally, are used for defining the way in which rules should be applied, e.g. for generating new concepts or during query processing.

Within Concept Theory, conditions, rules and laws are represented by the concepts **CONDITION**, **RULE** and **LAW**. The semantics of these concepts must be known to an actual EOS system, just like it must be able to interpret the concept **ISA**. Before these concepts are formally defined in Section 3.2, we will first characterize the different fields of their employment in order to get a better notion of what semantics they must be able to express. The following sections can thus be read as an outline of the different application areas of conditions, rules and laws.

3.1.3.2 Knowledge Acquisition

Knowledge acquisition is concerned with understanding external information sources. Corresponding tasks a knowledge processing system has to manage are extracting knowledge from input documents in various formats, e.g. XML text files, and integrating this knowledge into the ontological basis of the system, i.e. into its internal ontology. Consequently, knowledge acquisition involves many practical aspects concerning data formats and related extraction techniques. Questions arising in the context of extracting knowledge will therefore be covered in Chapter 4 when we are discussing issues concerning the realization of an actual EOS system. Here, we will focus on describing the general process of integrating newly acquired knowledge into an EOS ontology. In terms of Concept Theory this raises the question how a new concept can be integrated correctly into an EOS ontology.

At this point we differentiate between two kinds of acquisition procedures. Reading in an EOS ontology itself can be regarded as a procedure that is absolutely reliable in a sense that the encoding format of ontologies is predefined and known to the system. Therefore, integrating an EOS ontology into a knowledge processing system consists of directly parsing native syntax, extracting the respective concepts and storing them internally for later access. The acquisition procedure, here, can interpret incoming data reliably as concepts belonging to a uniform ontology. The second class of acquisition procedures involves foreign data formats that encode knowledge differently, e.g. in natural language. This means that these acquisition procedures must first translate the data content into concepts that can then be integrated into the ontological basis of the system. Relevant techniques, here, involve natural language processing and information wrapping methods that help identifying and extracting concepts from foreign sources. Note also the difference in scope of these procedures: reading in an ontology can be directed towards both, universals and particulars of a domain, while extracting concepts from foreign sources will necessarily be only oriented to particulars that can be identified *on the basis* of the already known (universal) concepts.

Incorporating a foreign concept into an existing EOS ontology is not at all an arbitrary process. On the contrary, we can expect that the extraction procedure is reliable in a sense that this new concept can be identified as an instance of some specific conceptual universal. For example, the result of an extraction procedure may be the concept **PARTICU-**

LAR:NEW_ADDRESS that is, according to the semantics of the reliable procedure, an occurrence of the ontology concept ADDRESS. Note that **PARTICULAR:**NEW_ADDRESS is a conceptual particular, which is not a specialty of this example. In fact, all concepts extracted from foreign documents will necessarily be particulars, while input describing EOS ontologies (containing universals) are regarded as native documents. In Section 3.1.2 we have presented the different kinds of input data for knowledge processing systems, in particular documents and knowledge representations, in our approach EOS ontologies that comprise ontological, onto-epistemic and epistemic objects of knowledge[32]. An EOS ontology contains the domain model as such, and therefore all conceptual universals that are necessary to describe the respective domain of interest, such as e.g. the universal ADDRESS. We can assume that there is a reliable procedure that reads in an EOS ontology directly and generates an internal representation of the concepts involved. Our focus in knowledge acquisition lies not here, but in integrating new particular concepts that enter the system after this initialization phase.

Documents from other sources carry concrete information belonging to actual entities of that domain, represented by conceptual particulars like **PARTICULAR:**NEW_ADDRESS. When integrating such a foreign concept into an EOS ontology, the outset of the knowledge processing system is that this concept is a particular, and the task of the system is to determine the most specialized parent concept of the particular within the set of all occurrences of the specific parent concept the extraction procedure has proposed. For **PARTICULAR:**NEW_ADDRESS this means finding its most specialized parent that is also an occurrence of ADDRESS. At a first glance it seems to suggest itself to immediately chose the already detected parent concept ADDRESS. Yet, the purpose of an ontology is to determine – most accurately – all existents, universals and particulars, of a domain. The more specialized a particular can be described the more knowledge about it is held within an ontology. It does make a difference in terms of expressiveness whether **PARTICULAR:**NEW_ADDRESS is an occurrence of ADDRESS, or one of its occurrences, e.g. GERMAN_ADDRESS or ENGLISH_ADDRESS. Consequently, it must be the aim of all integration processes to find the ontologically most differentiated position for any new concept.

Integrating this new concept into an existing EOS ontology means finding its correct position within the corresponding ontology graph. The assumption, here, is that the domain model represented by the ontology is sound in a sense that each domain particular can be non-ambiguously assigned as an instance of a specific universal. If this were not the case, the domain model would not reflect the domain correctly and require modifications. For example, any particular possesses an immediate parent, i.e. a most specialized universal. If there were a domain particular that could be attributed to several of these most specialized universals, it is an instance of all of them. Yet, this implies that the particular integrates the semantics of all of its parent concepts, which also entails the existence of another, intermediate universal that should be a child concept of these universals and the single parent concept of the particular (see also Section 2.2.4.2.3 and Proof 2.1 for a discussion on single immediate parent concepts of ontology particulars).

In fact, the new particular will be a leaf of the concept graph determined by the acquisition procedure. In our example, this procedure found a particular **PARTICULAR:**NEW_ADDRESS that is expected to be an occurrence of ADDRESS, i.e. a leaf of the concept graph G_{ADDRESS}. This can be immediately derived from the graph structure that represents the specialization

[32] Integrating these ontological concepts into the system is, of course, a straight forward procedure. The syntax of an ontology file is previously known to the system and contains a complete EOS ontology, i.e. the position of each of these concepts within the ontology is already fully determined.

hierarchy. Specifically, we can expect that **PARTICULAR:**NEW_ADDRESS is a child concept of one of the most specialized universal occurrences of ADDRESS, i.e. of a conceptual universal whose only specializations are particulars. Additionally, the internal component structure of such possible parent concepts can be verified, which means checking whether the particular is a structural instance of the universals in question. All necessary tests in that respect can be run on the ontology and concept structures alone.

Yet, this syntactical testing may yield several candidates that possess the required internal structure but convey different semantics, e.g. HOME_ADDRESS and OFFICE_ADDRESS. This shows that structural properties alone may not suffice for a correct interpretation of a new particular in relation to its prospective parent concepts. What is needed is a way of expressing semantic facts about these concepts. Such semantic facts can be expressed by conditions, conceptual relations which must be valid for all occurrences of specific concepts, e.g. that a HOME_ADDRESS pertains to the actual residence of persons. Rules that define conditions for concepts can then be used by the system to determine the correct parent concept of a newly acquired particular. In this way, the potential parent concepts of **PARTICULAR:**NEW_ADDRESS could be accordingly distinguished, which allows for a correct attribution of the particular to its unique parent. The algorithmical solution for integrating a new concept into the ontology graph according to these rules can be described by an according law, the concept **ACQUIRE**.

3.1.3.3 Knowledge Generation

Knowledge generation comprises all processes that result in new concepts computed from already existing ones. It can therefore be straightforwardly paraphrased as deriving concepts on the basis of an ontology. Here, we can differentiate between processes that produce new conceptual particulars and those that infer new conceptual universals. Inferring new universals results in a more detailed domain model, i.e. the ontological knowledge is being augmented by new conceptual relations. Producing new particulars means materializing the current status of actuality a system possesses about a certain domain. Hence, particulars represent the immediate facts about reality a knowledge processing system has at its disposition.

Conditions in terms of knowledge generation will concern the internal component structure of ontological concepts, as well as their properties as relations, such as transitivity or symmetry. Rules, then, attribute these conditions to specific concepts which can be used by laws that govern the generation of new concepts. The according algorithmical application of these rules is represented by the concept **GENERATE**.

3.1.3.4 Knowledge Retrieval

Knowledge retrieval in knowledge processing systems is concerned with computing return values for queries against the ontological basis. Queries will thus concern information about domain universals or particulars, as well as a combination thereof. Generally, we can differentiate between *structural queries* that pertain to the internal makeup of an ontology, and *semantical queries* that are directed towards the intellectual content an ontology represents. Typical structural queries would be ‘return all (mediate) occurrences of a concept C whose distance from C is a path of length 3’ or ‘return all concepts that have three components’. Semantical queries will be asking for (kinds of) concepts satisfying certain conditions, e.g. ‘return all (particular) occurrences of PERSON that work at a university X’, ‘return the number of (particular) occurrences of PERSON are working at university X’, or ‘return if there is a department of computer science at university X’. We will mainly concentrate on the latter type of queries, as these semantical queries are the main interest of human users who depend on the services of knowledge processing systems. Human information needs will generally be met by the semantic content of the ontological basis of the system, not by its concrete structural aspects.

Knowledge retrieval in knowledge processing systems is typically directed towards three kinds of semantical queries. Most commonly, queries will be asking for information on existents, usually particulars, of a domain. Considering EOS ontologies this means that such queries are retrieving sets of concepts of a certain type, possibly narrowed down further by satisfying specific conditions, e.g. all particular instances of PERSON that are called 'John'. Secondly, numerical values computed from such result sets can be of interest. Regarding the previous example, a possible numerical value would be the total number of concepts that satisfy the above query, i.e. how many particular instances of PERSON actually can be found that are called 'John'. Thirdly, it is useful to find out whether certain conditions are true for domain existents, i.e. whether they are met by concepts of the ontology, e.g. if there is a known particular instance of PERSON that is called 'John'. Consequently, queries against formal ontologies will typically expect one of three kinds of return values, sets of concepts, numerical values, or truth values. At this point we are leaving aside any further computations that are outside the immediate ontological scope of knowledge processing, like conversions (e.g. from one currency to another, between units of measurement, temperatures, etc.), determining the average of numerical values, or performing any other higher level calculations. All of these computations can be regarded as secondary processing that can also be handled by applications that build upon the primary services provided by a knowledge processing system.

We will thus restrict our focus on the properties of the above mentioned three essential kinds of semantical queries immediately touching on the ontological basis of a knowledge processing system:

- *Sets of Concepts*. Queries related to sets of concepts can be immediately translated into the makeup of ontologies as graph representations. Whenever specializations, i.e. occurrences or instances, of concepts are searched for the system is, graphically speaking, processing their subtrees. Generalizations, on the other hand, are marked by the paths leading from these concepts to the root concept of the ontology graph. Within result sets we can differentiate between conceptual universals and particulars, if called for by a query. Note that this kind of queries is, from a semantical point of view, certainly not limited to specialization and generalization. Sets of concepts can also consist of components of concepts, i.e. of concepts that are set in relation by any other concept, or family of concepts. Such a query could be searching, e.g. for all conceptual particulars that are part of a family, i.e. for all particular components of occurrences of the concept FAMILY.
- *Numerical Values*. Instead of returning a set of concepts it may already be sufficient to specify the cardinality of this set. The manner in which such queries are processed is essentially the same, before the result set is transformed into a numerical value. We will therefore not treat this kind of query separately on a theoretical basis but regard all important aspects and problems already covered by queries that are directed towards sets of concepts.
- *Truth Values*. A knowledge processing system must be able to test conditions on concepts and return the according truth values, *true* or *false*. As introduced in section 3.1.1, concepts can be equated with epistemological beliefs, and during knowledge processing it must be possible to proof or falsify the statement this belief, or concept, represents against the background of the ontological basis.

Next to different kinds of return values[33], queries show another dimension that has not been relevant in knowledge acquisition and generation, namely external users. Processing queries is a form of interaction of an EOS system with the outside world, i.e. applications or human users. In any case it might be feasible to model individual and groups of users in order to provide personalized views and access restrictions. Conditions and rules on queries will therefore concern both, target concepts and user properties. The algorithmical realization of queries itself is constituted by a class of laws, the concept **QUERY**.

3.1.3.5 Epistemological Outset for Knowledge Processing in EOS

Knowledge acquisition, knowledge generation and knowledge retrieval as motivated in the previous sections depict a classification model for epistemological processes within an EOS system. Such processes operate on ontological concepts with respect to their semantical properties. Concept properties are described by *conditions* and *rules*. Conditions are representing truth statements that may be attributed to ontology concepts by according rules. These rules are, at their part, associated with *laws*. Laws, generally speaking, specify procedures that are utilizing rules in order to manage epistemological processes. As such, they are representing the algorithmical realization of these processes, offering parameterizations for them. The following sections will discuss how conditions, rules and laws can be incorporated into the formal body of Concept Theory, i.e. how epistemological knowledge can be formalized and integrated into an ontological domain model.

3.2 Epistemology in Concept Theory

This section will present how epistemological facts can be modeled in Concept Theory. In Section 2.2 we have learned the derivation of concepts from metaphysical realism, their essential syntax and their semantics, which are delineated by the semantics of specialization. To this point, Concept Theory knows only ontological semantics, represented by **ISA**, and a notion of parthood, as components are parts of concepts. These semantics are syntactically mirrored in the structural makeup of EOS ontologies. An EOS ontology is thus a directed graph marking the specialization structure of a domain of interest. While this structure represents a concrete formalization and categorization of this domain, its ontological expressiveness cannot exclusively account for all questions arising when processing knowledge. Knowledge processing covers several different areas of epistemological processes that have already been introduced in Section 3.1.1: knowledge acquisition, generation and retrieval.

Epistemological semantics are statements *about* ontological facts or, in terms of Concept Theory, statements about concepts. Examples of such statements would be those about the meaning of concepts like '*the SHAPE of a BALL is ROUND*', and others about properties of concepts, such as '*TALLER_THAN is a transitive concept*'. These examples show the different perspectives of knowledge representation and processing. Knowledge representation is concerned with formalizing facts *about a domain*, which leads to the notion of formal ontologies. This ontological knowledge can then be utilized, e.g. for knowledge generation or retrieval, which requires an understanding that allows for a correct interpretation of the ontological facts. All according statements *about ontological knowledge* is what we call epistemological knowledge.

[33] We are here concentrating on semantical queries that are addressing ontology concepts, e.g. *'return all occurrences of concept C that satisfy condition X'*. It would also be possible to query an ontology structurally, e.g. *'return all concepts whose **ISA** distance to concept C lies within a path of length 2'*. The return values for these kinds of queries can also be expressed using sets of concepts, numerical and truth values.

3.2.1 On Modeling Epistemological Rules

Conditions in Concept Theory express statements that must be satisfied (true) within the context of the concepts they are attributed to. Accordingly, conditions are statements *about* concepts, and may be true for one concept and false for another. In this sense, they may be used to define rules or expressions for concepts, e.g. the rule that all *persons* called 'John' are *males*. This example illustrates a basic ontological property of such rules – the scope of rules is not limited to single concepts. A rule that applies to the concept PERSON will also be valid for all occurrences of the concept. For example a CHILD is an occurrence of PERSON (cf. Example 2.1c), and all instances of CHILD called 'John' (i.e. where the NAME component is occupied by JOHN_NAME) are certainly *males* as well. As pointed out in Section 2.2.4.2.1, each concept can be regarded as an *ontological kind* of all of its occurrences because of the semantical properties of specialization. Accordingly, rules pertaining to a concept will at the same time touch on all representatives of this ontological kind, i.e. these rules must also be valid for all occurrences of this concept within a formal ontology.

Rules are onto-epistemic objects and form part of the epistemological domain model that should be integrated seamlessly into the ontological domain model, represented in our approach by EOS ontologies. Thus it is of importance to examine how rules may be incorporated into Concept Theory. Obviously, in metaphysical terms, rules are universals and as such describable by conceptual relations, i.e. in Concept Theory rules are simply concepts. However, the exact structure of conceptual rules and their employment, as well as their exact semantics within the theoretical body of Concept Theory has yet to be developed. In order to illustrate the syntactical and semantical aspects of rules concerning Concept Theory, we will elaborate an example rule and discuss its implications. We argue that ontological semantics alone cannot account for rules, i.e. the semantics of rules must be regarded from an epistemological perspective. Based on these considerations we will then give a precise definition of conceptual rules and describe their epistemological semantics.

Our example describes rules on the concept LANDLORD, another occurrence of PERSON. The statement '*a landlord owns a building*' is a rule for what it means to be a *landlord*, besides being a person, which implies having a name, etc. In a first approach to modeling this rule using concepts, the fact that a landlord owns a building could naturally be expressed by a concept like OWNS_BUILDING:=(2,(LANDLORD,BUILDING)). However, the semantics of OWNS_BUILDING are not what '*a landlord owns a building*' is actually intended to denote. OWNS_BUILDING itself merely states the kinds of components that are valid for its own occurrences such as e.g. OWNS_BUILDING_OCC:=(2,(BARRY,SMITH_MANSION)). The actual semantics of OWNS_BUILDING can best be explicated by referring to its ontological context. It is very probable that an ontology comprising OWNS_BUILDING also contains a parent concept of the form OWNS:=(2,(OWNER,PROPERY)) that models ownership. OWNS_BUILDING, as an instance of OWNS, can therefore be only interpreted as a special case of ownership, and not as a fact about landlords. While the statement '*a landlord owns a building*' may be formally described as

$$\forall \text{ x, x is a landlord: } \exists \text{ building b: x owns b}$$

the concept OWNS_BUILDING thus really conveys

$$\exists \text{ x, x is an ownership: x.owner is a landlord} \wedge \text{x.property is a building.}$$

What OWNS_BUILDING obviously fails to express is a direct relationship with the concept LANDLORD. Note that this cannot be redeemed by using another concept operating with concept references, e.g. denoting *identity* with concept LANDLORD_IDENTITY:=(2,(LANDLORD, OWNS_BUILDING[1])). The semantics of this concept, again, are describing a special case of identity, and not LANDLORD itself. The impact of this becomes clear when regarding instances of the two concepts LANDLORD and LANDLORD_IDENTITY. Again, there is no way of assuring that there exists an appropriate instance of LANDLORD_IDENTITY for each instance of LANDLORD. In fact, one can even specify syntactically correct instances of LANDLORD_IDENTITY that explicitly, and falsely, state the identity of two non-identical occurrences of LANDLORD. Note that this is not a matter of modeling a domain correctly, i.e. all concepts of an ontology are supposedly sensible, but of ontological expressiveness. The semantics of EOS ontologies so far are solely based on specialization and parthood, represented by **ISA** and the status of concept components. These alone simply cannot account for conditions whose scope is intended to cover all occurrences of a concept intensionally (besides the semantics of specialization themselves, of course). When modeling the statement '*a landlord owns a building*' in an *ontological* way (i.e. based on specialization) one may consequently only refer to LANDLORD itself, or concepts related to it along the **ISA** hierarchy, in particular its parent concepts and components.

There are two explicit ways to incorporate such a rule ontologically into an EOS ontology, namely isolated by (a) introducing a new component BUILDING to LANDLORD or, respectively, by (b) defining an appropriate parent concept BUILDING_OWNER of LANDLORD. These two possibilities are depicted in Figure 3.3. Concept components in this figure are represented by ovals enclosed by concept ovals.

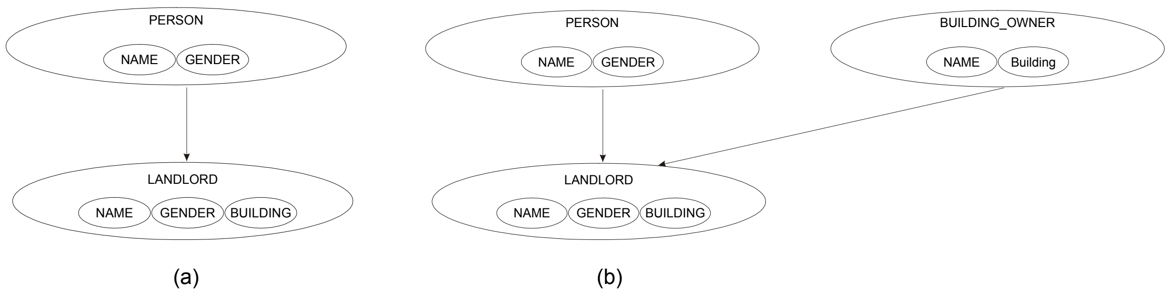


Figure 3.3: Ontological Explanation of the Concept LANDLORD

Yet, further exemplifications of LANDLORD may pose difficulties when modeling them ontologically, i.e. by specialization or structure only. The statement '*a landlord owns a building or a piece of land*',

$$\forall \text{ x, x is a landlord: } \exists \text{ building b: x owns b} \vee \exists \text{ land l: x owns l,}$$

cannot be easily translated directly into an ontology, as the disjunction *or* in this context is not supported by ontological structures (nor are more complex statements such as '*a landlord owns a building or a piece of land and rents it to a tenant*'). One of the problems arising here is that ontological specializations are conjunctive, and are therefore not apt for expressing disjunctive statements. For this reason Concept Theory treats inheritance of concept components as Figure 3.4 indicates – a child concept per definition occupies *all* components of all of its parent concepts (see also Definition 2.14). The implication of the graph structure of Figure 3.4 thus can be paraphrased as '*a landlord owns a building* and *a piece of land*',

$$\forall \text{ x, x is a landlord: } \exists \text{ building b: x owns b} \wedge \exists \text{ land l: x owns l,}$$

(derived from the ontology semantics that LANDLORD is a specialization of the three concepts LAND_OWNER, PERSON and BUILDING_OWNER, defining a relation on its four subconcepts NAME, GENDER, LAND and BUILDING). The disjunction *or* has thus been turned into the conjunction *and*, which is not the intended content of the original statement.

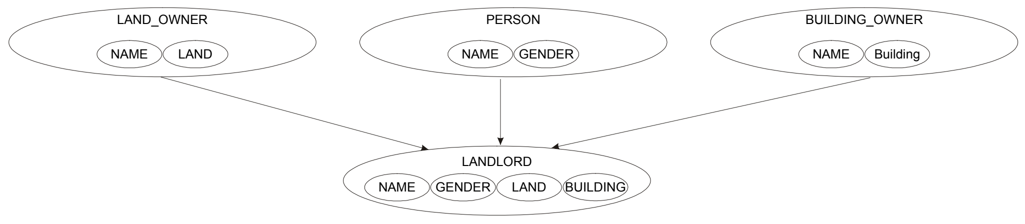


Figure 3.4: Conjunctive Property of Specializations

A correct, but inconvenient way to convey the meaning of '*a landlord owns a building* or *a piece of land*' ontologically, would be to specify an additional concept LAND-LORD_PROPERTY that is a parent of both, LAND and BUILDING. This concept can then be used as a component of LANDLORD (cf. Figure 3.5). The concept LANDLORD_PROPERTY acts as a generalization of LAND and BUILDING, i.e. it may be specialized *either* by LAND *or* BUILDING, which models a disjunctive ontological structure.

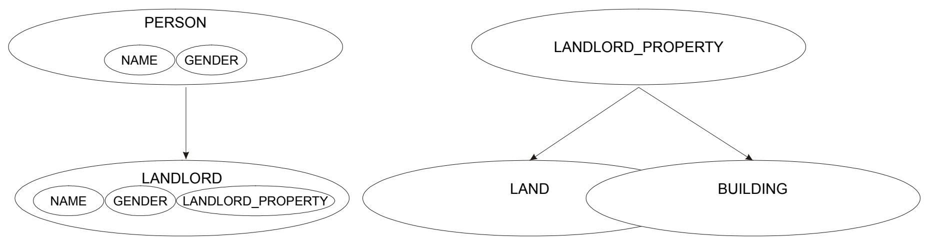


Figure 3.5: Refined Ontological Explanation of the Concept LANDLORD

Still, we cannot describe arbitrary rules for a concept C ontologically by introducing according components to C. This is for example the case if these components are themselves interrelated through another concept R, but within the context of C. As an example we will discuss the already mentioned extended statement '*a landlord owns a building or a piece of land and rents it to a tenant*',

$$\forall \text{ x, x is a landlord: } \exists \text{ t, t is a tenant:}$$
$$(\exists \text{ building b: x owns b} \wedge \text{t pays for b}) \vee (\exists \text{ land l: x owns l} \wedge \text{t pays for l}).$$

Clearly, there exists a relationship between a landlord and his property (the landlord owns it) as well as between a landlord and a tenant (the person the landlord rents the property to). On the other hand, there is a relationship between the tenant and the landlord's property (the tenant pays for using it). The implication, here, is that the property owned by the landlord is the very same property the tenant is paying the landlord for. Analogously to the initial consideration to use a concept OWNS_BUILDING for expressing ownership, one could define the concept PROPERTY_DEAL:=(3,(LANDLORD,TENANT,PROPERTY)) for depicting this relationship. Yet again, PROPERTY_DEAL does not, as it should, directly attribute to the definition of LANDLORD, nor does it ensure that LANDLORD and TENANT relate to the same PROPERTY. This means that one could construct a valid instance PROPERTY_DEAL_INST:= (3,(L_P1,T_P2,P1)) of PROPERTY_DEAL, implying that the property P1 really is owned by the landowner L_P1 but the tenant T_P2 is actually using another property P2. Thus, what cannot be expressed ontologically, here, is identity (or any other relation) among components of concepts. Analogously, even defining LANDLORD:=(4,(NAME,GENDER,PROPERTY, TENANT)) as a concept comprising TENANT as a component (besides being not feasible from a modeling point of view) still cannot account for the semantic relationship between TENANT and PROPERTY.

As a last option one might want to solve this referential problem by using the technique shown previously in Figure 3.5, i.e. by nesting concepts. It is possible to define LANDLORD with reference to the concept TENANT that now encapsulates LANDLORD_PROPERTY as indicated in Figure 3.6. But specified in this way LANDLORD is only indirectly related to LANDLORD_PROPERTY through TENANT, which contradicts any intuitive notion of defining concepts like LANDLORD.

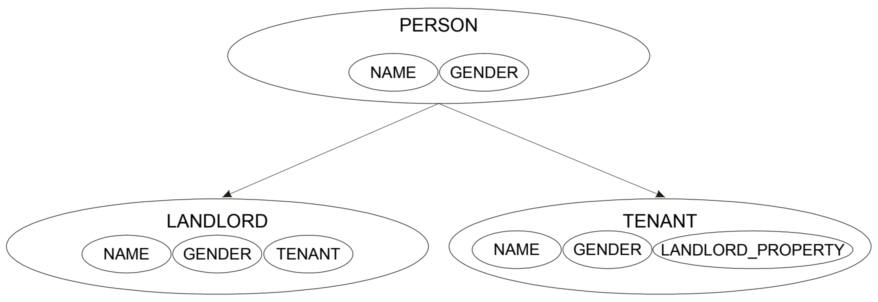


Figure 3.6: Nested Ontological Explanation of the Concept LANDLORD

In conclusion, the structural properties of specialization, which are representing the internal structure of an ontology, do not account for interrelationships between concept components. Put more general, this means that conditions and rules on concepts cannot be expressed ontologically but belong to the epistemological perspective on a domain. How this epistemological perspective can be modeled within Concept Theory will thus be the central topic of the following sections.

3.2.2 Conditions and Rules

Epistemological knowledge can be subdivided into onto-epistemic and epistemic objects of knowledge (cf. Section 3.1.3). The latter are represented by conceptual laws while the former are expressed by conceptual conditions and rules, where conditions state properties that are related to ontological concepts via rules pertaining to these concepts. In this sense, conditions of a concept C are formalizations of propositions (or parts thereof) that are true for C. In general, any statement expecting a truth value with respect to concepts can be regarded as a condition. When modeling an ontology, conditions can be attributed to arbitrary concepts by according rules. Thus they are related to specific concepts at designated positions within an EOS ontology. From the structural properties of ontologies, in particular continuity (cf. Section 2.2.4.2.3), it then immediately follows that a condition defined for a concept C also affects all specializations of C.

In order to test conditions expressing logical statements within a given EOS ontology, the notion of truth values for concepts must be defined. This is the central question of Section 3.2.2.1. Subsequently, conceptual rules and applying identity conditions are formally introduced in Section 3.2.2.2. Testing rules for given concepts is then discussed in Section 3.2.2.3. Section 3.2.2.4 shows how complex truth statements can be formulated using logical conditions, and Section 3.2.2.5 demonstrates the applicability of rules for modeling general relation properties.

3.2.2.1 Truth in EOS Ontologies

Conditions interpret concepts as truth values, and are embodiments of statements about concepts and their occurrences. Truth properties for concepts can be defined by an according function I_O that interprets each concept as either logically *true* or *false*.

Definition 3.1: The Interpretation Function I_O

Let **O** be an EOS ontology. The interpretation function I_O: $\Phi \rightarrow$ BOOL for ontology **O** is defined as follows:

- $I_O(C) =$ *true* $\forall\ C \in \mathbf{O}$
- $I_O(C) =$ *false* $\forall\ C \notin \mathbf{O}$

The interpretation function I_O yields the Boolean value *true* for all concepts of **O**, i.e. all ontology concepts can be interpreted as true statements about the domain they are modeling.

This understanding of concepts as true statements, or beliefs, is in accordance with the general perspective on knowledge as laid out in Section 3.1.1, where true beliefs have been identified as the basis for knowledge processing. In this sense, ontology concepts form the fundamental set of beliefs for epistemological processes. Thus, the domain model itself is accepted by an EOS system as a true perspective on the domain of interest. Any new concept NEW_CONCEPT (e.g. provided by external sources) that enters the system can be tested on this basis. Thus, a concept will either get rejected if it does not comply with the ontology, or it is integrated into the ontology once it has been judged compatible with the other ontology concepts.

This testing procedure builds upon conceptual rules that are specifying conditions for given concepts. Therefore, rules act as integrity constraints on concepts and are, as such, a means for determining the compatibility of NEW_CONCEPT with the respective ontology: if a rule applies to it, this rule must accordingly be tested on NEW_CONCEPT. Only if it satisfies all rules pertaining to it, NEW_CONCEPT can be made part of the ontology. The application of (general) rules to new (specific) concepts is delivered from the specialization hierarchy of the ontology, i.e. a rule defined for a concept C will also be valid for its occurrences. Thus, for NEW_CONCEPT all rules apply that refer to the concepts it is reducible to. Generally, rules are defined for ontology concepts, e.g. the rule *'each mother has a child'* refers to the concept MOTHER and specifies that HAS_CHILD:=(2,(MOTHER,CHILD)) must be true, i.e. a condition, for it. A rule valid for an ontology concept such as MOTHER is also binding for all occurrences of it. This means that the rule *'each mother has a child'* is not restricted to MOTHER alone, but will still be valid analogously for its occurrences e.g. SINGLE_MOTHER and WORKING_MOTHER, as well as for their particulars. If NEW_CONCEPT can be determined as e.g. a particular SINGLE_MOTHER, and is therefore also an occurrence of MOTHER, then the rule *'NEW_CONCEPT has a child'* (and all other rules for *mothers* as well) must hold for NEW_CONCEPT, too. Thus, if this and all other rules for MOTHER can be validated, NEW_CONCEPT will be regarded as a *true* concept and integrated into the ontology.

3.2.2.2 Rules and Identity Conditions

Conditions and rules, themselves, are relations, and are thus describable by Concept Theory. Conceptual rules interpret their internal components as conditions. Such a rule is expected to be in itself true – syntactically as a concept of the ontology, and also semantically concerning the truth statements they are representing.

Definition 3.2: The Concept RULE

Rules are represented by the fundamental relation **RULE**, which is defined as the concept

$$\textbf{RULE} := (\ 2, (\ \text{EXISTENT}, \text{EXISTENT}\ )\ ) \in \Phi_U$$

where

- **RULE**[1] is a condition specifying the concepts the rule is valid for and
- **RULE**[2] is the condition that applies for these concepts.

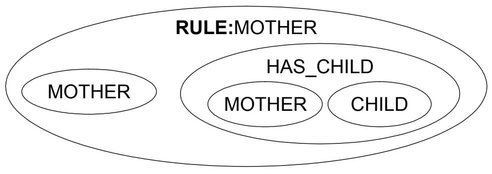
An occurrence of **RULE** states that whenever the condition of **RULE**[1] is true within an ontology, **RULE**[2] must be true as well.

Note: As rules are semantically distinct from other concepts, their occurrences are syntactically marked by according prefixes, i.e. **RULE:**.

The simple exemplary rule *'each mother has a child'* of the previous section is then formalized as shown in Example 3.1.

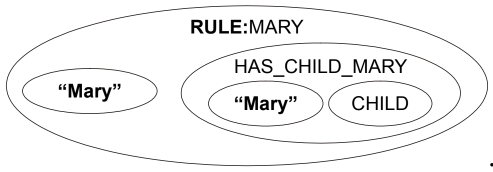
Example 3.1: A Rule for the Concept MOTHER

Let O be a formal ontology, let MOTHER, CHILD, HAS_CHILD:=(2,(MOTHER, CHILD)) and their components be ∈ O. The rule '*each mother has a child'* can be expressed by the conceptual rule



where MOTHER is the concept the rule **RULE:**MOTHER refers to, and HAS_CHILD specifies the condition that has to be met by all occurrences of MOTHER.

The rule **RULE:**MOTHER is binding for all occurrences of MOTHER. Yet, its definition alone, as depicted in Example 3.1, does not account for the semantic implications between its components MOTHER and HAS_CHILD. In particular, **RULE:**MOTHER does not express, the relationship between MOTHER mentioned in **RULE:**MOTHER[1] and MOTHER of **RULE:**MOTHER[2][1], i.e. as the first component of the condition HAS_CHILD. This relationship, which is valid in the context of **RULE:**PERSON, depicts that **RULE:**MOTHER[1] and **RULE:**MOTHER[2][1] refer to the same concept, which is trivially true for MOTHER but must also be assured for all occurrences of this concept. For example, for a particular **PARTICULAR:**MARY with value "Mary" the according rule must correctly be



This example has shown that in order to transport the correct semantics of rules, a correlation between their components (possibly to an arbitrary depth) must be specified. Defining such correlations in Concept Theory is done using *identity conditions*.

Definition 3.3: The Condition IDENTITY

The condition **IDENTITY**, is defined as the concept

IDENTITY := (2, (EXISTENT, EXISTENT)) $\in \Phi_U$,

where **IDENTITY**[1] and **IDENTITY**[2] delineate concept references pertaining to concept components of occurrences of **RULE** and **LAW** (see Section 3.2.3) that must be occupied by the same concept within an ontology. It follows that **IDENTITY**[1] and **IDENTITY**[2] must be of the same conceptual category, i.e. there must exist a specialization between them:

let **O** be an ontology, then

$\forall$ identity conditions **IDENTITY:**COND $\in$**O**:

$\exists$ **ISA:**COND1_2:=(2,(**IDENTITY:**COND[1], **IDENTITY:**COND[2])) $\in$**O** $\vee$
$\exists$ **ISA:**COND2_1:=(2,(**IDENTITY:**COND[2], **IDENTITY:**COND[1])) $\in$**O**.

Note that occurrences of **IDENTITY** can be syntactically identified by the prefix **IDENTITY:**. Furthermore, **IDENTITY** is a symmetric concept, i.e. within any ontology **O**, an identity condition **IDENTITY:**AB:=(2,(REF_A, REF_B)) implies a second, symmetric condition **IDENTITY:**BA:=(2,(REF_B, REF_A)). Without loss of generality one can assume that all mutually symmetric identity conditions are materialized in **O**. This assumption allows for defining a general integrity constraint for identity conditions in Concept Theory. Let **RULE:**R be any conceptual rule in **O**, and let $\mathbf{IDCON}_{\mathbf{RULE:}R}$ be the set of identity conditions that apply to **RULE:**R. All identity conditions of $\mathbf{IDCON}_{\mathbf{RULE:}R}$ must obey the following integrity constraint within ontology **O**:

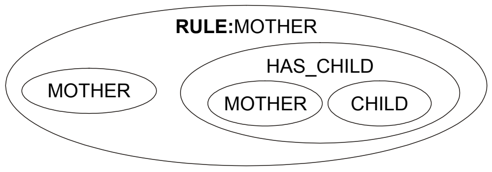
$\forall$ identity conditions **IDENTITY:**A, **IDENTITY:**B $\in\mathbf{IDCON}_{\mathbf{RULE:}R}$:
IDENTITY:A[1]=**IDENTITY:**B[1] $\Leftrightarrow$
$\exists$ **IDENTITY:**C:=(2,(**IDENTITY:**A[2], **IDENTITY:**B[2])) $\in\mathbf{IDCON}_{\mathrm{RULE:R}}$,

i.e. whenever two identity conditions relate one rule (sub)component with two distinct other (sub)components, these in turn must also be associated with an according identity condition. This assures that no rule (sub)component can be interpreted inconsistently when testing this rule on ontology concepts (see Section 3.2.2.3).

Identity conditions delineate the semantical structure of rules by specifying which concepts are bound to each other. Example 3.2 shows how an identity condition can be used for specifying the correlation between **RULE:**PERSON[1] and **RULE:**PERSON[2][1] of Example 3.1.

Example 3.2: An Identity Condition for RULE:MOTHER

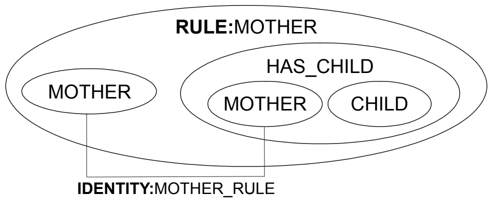
The rule '*each mother has a child'*, which is expressed by the conceptual rule



implies that **RULE:**MOTHER[1] and **RULE:**MOTHER[2][1] must be identical for all applications of this rule. The corresponding identity condition is:

IDENTITY:MOTHER_RULE := (2, (**RULE:**MOTHER[1], **RULE:**MOTHER[2][1])).

where MOTHER is the concept the rule **RULE:**MOTHER refers to, and HAS_CHILD specifies the condition that has to be met by all occurrences of MOTHER. For depicting identities graphically, the EOS notation is extended by edges used for connecting concepts that are being identified. Where emphasis on their individual function is necessary, these edges may be labeled with their concept names. The resulting graphical representation of **RULE:**MOTHER, including **IDENTITY:**MOTHER_RULE is then:



Rules, in combination with identity conditions, determine semantical aspects of ontology concepts. As such they are trivially *true* for the very concepts they have been defined for. Moreover, a rule defined for a concept C will also apply to all specializations of C, which demands a uniform procedure for testing rules on these occurrences. This procedure that examines whether a rule is applicable and *true* for a given concept will be discussed in the following section.

3.2.2.3 Testing Rules

Conditions in Concept Theory represent statements about concepts, and as such truth values. Either a condition is met by a concept C, in this case the condition is *true* for C, or this is not the case, which renders the condition *false* for C. They are defined explicitly for specific concepts, and are valid for these ontological kinds as wholes, i.e. for all of their occurrences. This property makes conditions very useful for epistemological processes, e.g. when answering queries against EOS ontologies, as they introduce new semantics on top of purely ontological semantics, i.e. semantics that are based on specialization alone (cf. Section 2.2.4). Rules bind conditions to concepts and determine integrity constraints for these concepts, as well as for their specializations, within a given ontology. This requires testing a (general) rule on these specializations, a procedure that proves or confutes a rule for them within the bounds of an EOS ontology. Here, the difference between syntactical and semantical truth of rules becomes obvious:

- *Syntactical Truth*. Any rule that is part of an EOS ontology O is syntactically true within O, as it is an element of O. Syntactical truth of rules (and all other concepts) is defined by the interpretation function I_O (cf. Section 3.2.2.1) that determines the truth of concepts with reference to a given ontology O.

- *Semantical Truth*. Rules represent true statement about ontology concepts, i.e. they are true by virtue of their semantical content. Semantical truth of rules is defined by another interpretation function, $I_{O,M}$ that decides on the truth of rules with respect to a specific *minting M* within a given ontology O.

Definition 3.4: Concept Minting

A *concept minting* of a concept $C \in \Phi_U$ is defined as the concept $M_C \in \Phi$ where

M_C is a valid occurrence of C.

Note that the concept C may possess very complex substructures, i.e. components that themselves have their own components, etc., up to an arbitrary but finite depth (smaller than the cardinality |O| of the ontology O of which C is an element of). The set of all of these nested components encapsulated by C is called ϕ_C.

M_C is a *valid concept minting* of a concept $C \in \Phi_U$, if

- M_C is a valid occurrence of C, and
- $\forall$ identity conditions $i \in \Phi \wedge (i[1] \in \phi_C \wedge i[2] \in \phi_C)$: i holds for M_C,

i.e. all identity conditions that are defined for C are also satisfied by its minting M_C.

Note: The set of all mintings for a concept C thus describes the set of all concepts that are structurally compliant with C. This set is not identical with all semantically correct occurrences of C, as some mintings may fail to satisfy rules defined for C.

The interpretation function $I_{O,M}$ which defines whether a rule is true for a given concept can now be specified.

Definition 3.5: The Interpretation Function $I_{O,M}$

Let **O** be an EOS ontology. The interpretation function $I_{O,M}$: $\Phi \rightarrow$ BOOL for a rule **RULE:**R $\in$**O** is defined as follows:

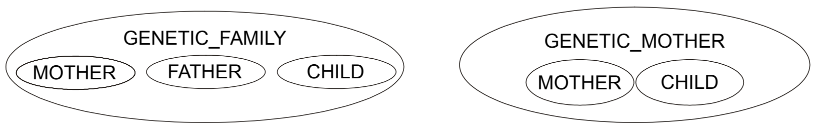
- $I_{O,M}$(**RULE:**R) = *true* $\Leftrightarrow$
  M is a valid minting of **RULE:**R[1]
  $\wedge \exists$ C$\in$**O**: C is a valid minting of **RULE:**R[2]
  $\wedge \forall$ identity conditions $i \in$**O** $\wedge (i[1] \in \phi_{\textbf{RULE:}R[1]} \wedge i[2] \in \phi_{\textbf{RULE:}R[2]})$:
  i holds for **RULE:**R
- $I_{O,M}$(**RULE:**R) = *false* else.

The interpretation function $I_{O,M}$ yields the Boolean value *true* only for valid mintings of rules in **O**, i.e. $I_{O,M}$ tests whether a given concept satisfies a rule or not.

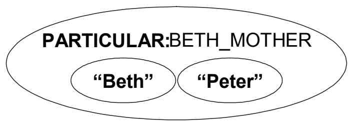
How rules can be tested according to the semantics of the interpretation function $I_{O,M}$ is shown in Example 3.3.

Example 3.3: Testing Rules

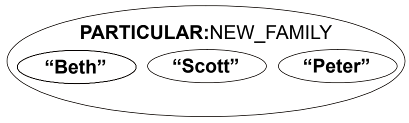
Let GENETIC_FAMILY and GENETIC_MOTHER be two concepts of an ontology **O**:



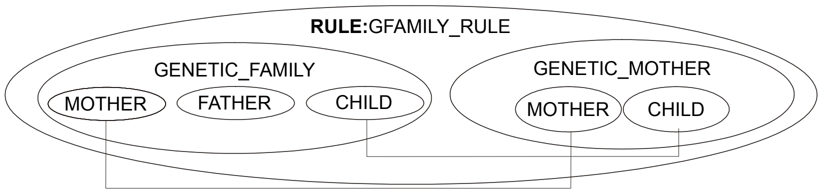
where GENETIC_FAMILY is an instance of FAMILY and GENETIC_MOTHER implies that all conceptual particulars of this relation touch on children and their real mothers. Let one such particular of **O** be:



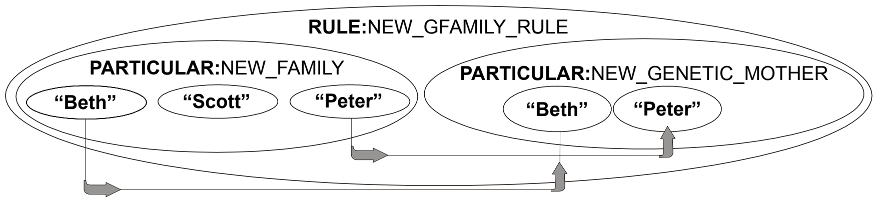
If a new concept has to be integrated into **O**, e.g. this concept has been extracted from an external source, it has to be decided, what ontological position it should receive. Let



be a newly extracted concept that has been identified as an instance of FAMILY. For determining **PARTICULAR:**NEW_FAMILY as precisely as possible, it is important to know, if it pertains to a specific class of families (of universal occurrences of FAMILY), e.g. if it is an instance of GENETIC_FAMILY. Let this be the case according to its ontological structure, i.e. all of its components are reducible to their respective roles in GENETIC_FAMILY. Yet, epistemologically, a rule has to be fulfilled for all instances of GENETIC_FAMILY, namely that the concept occupying the MOTHER role is the real mother of the occupant of the CHILD role. This rule is expressed by the concepts



The new concept **PARTICULAR:**NEW_FAMILY can now be tested against this rule. In order to do this the system creates new instances of these conditions that pertain to **PARTICULAR:**NEW_FAMILY. The newly created rule is of the form


RULE:NEW_GFAMILY_RULE
PARTICULAR:NEW_FAMILY
"Beth"
"Scott"
"Peter"
PARTICULAR:NEW_GENETIC_MOTHER
"Beth"
"Peter"

where **PARTICULAR:**NEW_FAMILY occupies the first component of **RULE:**NEW_GFAMILY_RULE and the new concept **PARTICULAR:**NEW_GENETIC_MOTHER the second. Thus, **PARTICULAR:**NEW_FAMILY is the minting of GENETIC_FAMILY of **RULE:**GFAMILY_RULE. The minting **PARTICULAR:**NEW_GENETIC_MOTHER has been constructed from its role GENETIC_MOTHER on the basis of the according identity conditions (which is highlighted by the arrows along the identity edges in the graphical representation of **RULE:**NEW_GFAMILY_RULE).

For determining if concept **PARTICULAR:**NEW_FAMILY is an epistemologically valid instance of GENETIC_FAMILY, the system has to test whether the rule **RULE:**NEW_G-FAMILY_RULE applies to it, i.e. if this rule is *true*. This were the case if all of its components are *true*. Per hypothesis this is the case for **PARTICULAR:**NEW_FAMILY but it still has to be tested for **PARTICULAR:**NEW_GENETIC_MOTHER. The task, therefore, is to find an instance of GENETIC_MOTHER within **O** that possesses the components of **PARTICULAR:**NEW_GENETIC_MOTHER. If this test is positive, the rule **RULE:**G-FAMILY_RULE applies to **PARTICULAR:**NEW_FAMILY. As indicated above, this is the case because **PARTICULAR:**BETH_MOTHER is part of **O**.

In summary, conditions and rules thus represent general truth statements about concepts with respect to a given ontology. For all particulars and universal occurrences of an ontology concept, the rules concerning it will also hold for them. This is an important property of concepts regarding epistemological processes. Knowledge acquisition procedures can make use of such rules when determining the ontology graph position of new concepts, rules must be met when generating new concepts, and query processing is facilitated as semantical properties of concepts can be determined at an early level (relatively close to the root concept) of the ontology graph. A concept is regarded as being true within an ontology if it is an actual element of the ontology graph, i.e. trivially, any ontology concept is in itself *true*. For any non-ontology concept (e.g. extracted from foreign sources), rules can be tested and found satisfied or not, which corresponds to assigning the truth values *true* and *false* to these concepts on the basis of a certain rule.

3.2.2.4 Rules and Logical Conditions

Ontologically (regarding only the specialization hierarchy of an ontology), relations among concept components can only be expressed in their general form, i.e. outside the scope of a specific concept (cf. the motivation for modeling rules in Section 3.2.1). Identity conditions, on the other hand, allow for defining component correlations that can be used in other conditions and rules (as well as for laws). Specifically, this allows for concept mintings on rules. In this way, rules can be tested for a given concept. So far, rules can only contain simple conditions, i.e. ontological domain concepts. For expressing more complex truth statements, logical conditions that combine several conditions by conjunction, disjunction or negation must be defined in Concept Theory. Logical Conditions for rules and laws in Concept Theory are represented by the concepts **AND**, **OR** and **NOT**.

Definition 3.6: The Conditions AND, OR and NOT

Concept Theory knows the semantics of several logical conditions:

- **AND** := (2, (EXISTENT, EXISTENT)) $\in \Phi_U$
- **OR** := (2, (EXISTENT, EXISTENT)) $\in \Phi_U$
- **NOT** := (1, (EXISTENT)) $\in \Phi_U$

with occurrences of **AND**, **OR** and **NOT** being delineated by the prefixes **AND:**, **OR:** and **NOT:**, carrying the semantics:

- $\forall$ occurrences **AND:**OCC of **AND**:
  I_O(**AND:**OCC) = *true* $\Leftrightarrow$ I_O(**AND:**OCC[1]) = *true* $\wedge$ I_O(**AND:**OCC[2]) = *true*

- $\forall$ occurrences **OR:**OCC of **OR**:

I_O(**OR:**OCC) $\Leftrightarrow$ I_O(**OR:**OCC[1]) = *true* $\vee$ I_O(**OR:**OCC[2]) = *true*

- $\forall$ occurrences **NOT:**OCC of **NOT**:
  I_O(**NOT:**OCC) = *true* $\Leftrightarrow$ I_O(**NOT:**OCC[1]) = *false*

where I_O is the interpretation function for concepts C of an ontology **O**.

Notes:

- As occurrences of conditions are syntactically emphasized using respective prefixes, they need not be explicitly modeled as such in EOS ontologies, i.e. it suffices to define, e.g. a condition **AND:**OCC without specifying **ISA:**ANDOCC:=(2,(**AND:**OCC, **AND**)). Implicitly, any EOS ontology certainly possesses all condition concepts and according **ISA** occurrences.
- Without loss of generality one can assume occurrences of **AND** with arbitrary arity, e.g. conjunctions of several concepts with nested **AND** occurrences, such as

  AND:NESTED_AND := (2,(EXISTENT, **AND:**1)),
  AND:1 := (2,(EXISTENT, **AND:**2)),
  AND:2 := (2,(EXISTENT, **AND:**3)), …
  AND:n-1 := (2,(EXISTENT, **AND:**N)),

  can be equivalently expressed using a single n-ary **AND** occurrence:

  AND:N-ARY_AND := (n, (EXISTENT, …, EXISTENT)),

  which is due to the associative property of conjunctions. For example the **AND** occurrence **AND:**ABC:=(3,(A,B,C)) conveys a meaning that is semantically equal to both, **AND:**A_BC:=(2,(A,**AND:**BC)) and **AND:**A_BC:=(2,(**AND:**AB,C)), for the matching conjunctions **AND:**AB:=(2,(A,B)) and **AND:**BC:=(2,(B,C)). As conjunctions are also commutative, **AND:**ABC:=(3,(A,B,C)) is also equal to, e.g. **AND:**BAC:=(3,(B,A,C)) or **AND:**CBA:=(3,(C,B,A)). Thus, the concept **AND:**ABC can be seen as a representative of all of its commutative and associative variations.

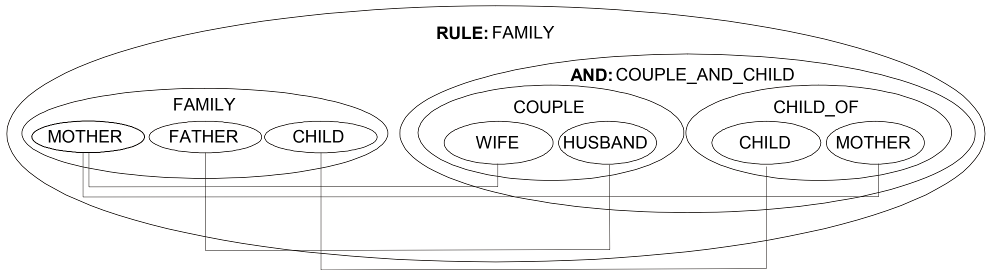
  The same applies analogously for occurrences of the condition **OR**.

Note that it is essential to observe the semantics of these conditions compared to those of other concepts. Condition occurrences are attributed epistemological semantics of their own, i.e. their meaning is not limited to their ontological status. Ontologically, they are nothing but elements of an EOS ontology, i.e. specializations of the root concept EXISTENT. Epistemologically, they are expressing truth statements. Unlike other concepts, conditions therefore have an ontological as well as an epistemological scope and can thus be used for formulating predicates. Logical conditions, such as occurrences of **AND** and **OR**, are conceptual relations that can be nested in order to express more complex statements.

From the basic outset of Concept Theory, concepts act as relations on their components and specializations of these components, respectively. This means that components can be replaced by their specializations while the outer relation remains valid. The formal basis for this has been presented in Definition 3.4 that describes the semantics of concept minting. Valid concept mintings for conceptual relations are their syntactically correct occurrences. For example, the concept FAMILY:=(3,(MOTHER, FATHER, CHILD)) implies a valid occurrence of the form MARY'S_FAMILY:=(3,(Mary, FATHER, CHILD)) if there exists an according **ISA:**MARY:=(2,(Mary, MOTHER)). Yet, usually, replacing concept components with their specializations will not be an arbitrary process, e.g. INCORRECT_FAMILY:=(3,(Mary, Peter, John)) may very well possess the correct syntactical structure for *families*, yet be a semantically incorrect occurrence of FAMILY if *Mary* and *Peter* are not a married couple, or *John* is not their child (supposing that the semantics of FAMILY are defined in that way). This can be translated into rules such as '*within a family, mother and father must be a married couple*' and '*within a family, the child must be the real child of the mother*'. Logical conditions, in this context, allow for combining such statements into a single rule. In order to illustrate this we will model the rule *'within a family, mother and father must be a married couple and the child is the real child of the mother '* as depicted in Example 3.4.

Example 3.4: A Simple Rule for the Concept FAMILY

Let O be a formal ontology, let FAMILY:=(3,(MOTHER, FATHER, CHILD)), COUPLE:=(2,(WIFE, HUSBAND)), CHILD_OF:=(2,(CHILD, MOTHER)) and their components be ∈ O. The rule '*within a family, mother and father must be a married couple and the child is the real child of the mother*' can be expressed by the concept:

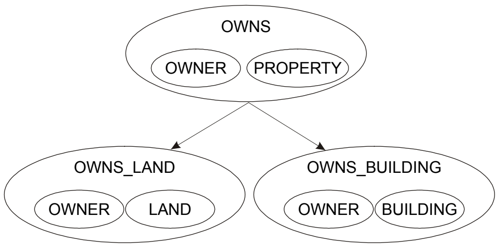


In this way, arbitrary complex truth statements can be defined by means of logical conditions. This also leaves an option to formalize the rule for *landlords* of our motivating example in Section 3.2.1. The characteristics of the concept LANDLORD can therefore be defined epistemologically using the appropriate conditions for delineating the rule '*a landlord owns a building* or *a piece of land and rents it to a tenant*' for it. An according excerpt of an EOS ontology is given in Example 3.5 below.

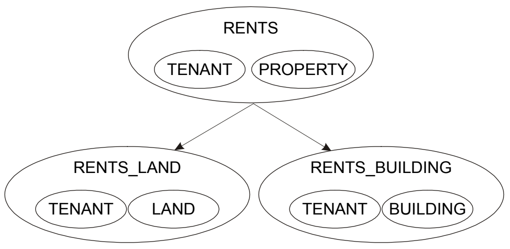
Example 3.5: A Rule for the Concept LANDLORD

The statement '*a landlord owns a building* or *a piece of land and rents it to a tenant*', formally expressed as *'$\forall x$, x is a landlord: $\exists t$, t is a tenant: ($\exists$ building b: x owns b $\wedge$ t pays for b) $\vee$ ($\exists$ land l: x owns l $\wedge$ t pays for l)'* is represented by the concept **RULE:**LANDLORD, which is defined as follows (assuming that concepts LANDLORD, TENANT, OWNER, PROPERTY, LAND, and BUILDING have been previously defined).

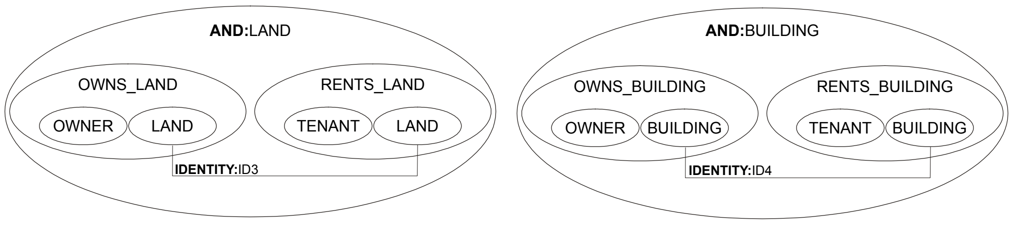
Let OWNS be a concept expressing ownership, let OWNS_LAND and OWNS_BUILDING be two instances of OWNS:



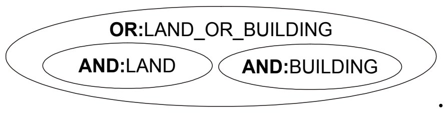
Let RENTS be an according concept denoting the activity of renting property, with instances RENTS_LAND and RENTS_BUILDING:



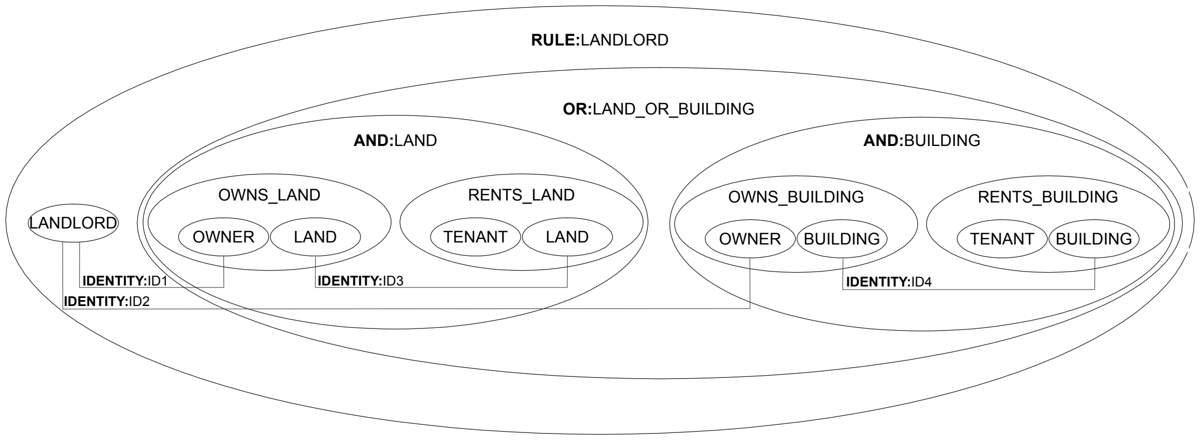
The two conjunctions of the statement can then be specified as



while the disjunction of the statement is defined as



Accordingly, the complete definition of the rule for the concept LANDLORD, including all identity conditions involved, is[34]:



3.2.2.5 Rules and Relation Properties

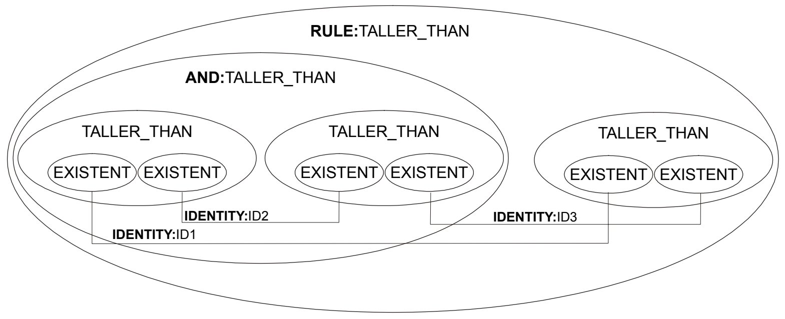
Rules specify true statements about ontology concepts, i.e. they can be used to define the properties of these concepts with reference to a given ontology. The previous sections have presented examples of rules, e.g. for the concepts PERSON (Example 2.1), FAMILY (Example 3.4) and LANDLORD (Example 3.5). Generally, a rule **RULE:**R associates concepts of the condition **RULE:**R[1] with the concepts of condition **RULE:**R[2], e.g. LANDLORD, the first component of **RULE:**LANDLORD in Example 3.5 has been associated with the concepts OWNS_LAND, RENTS_LAND, OWNS_BUILDING, RENTS_BUILDING of the condition **RULE:**LANDLORD[2]. Semantically, this means that LANDLORD is correlated with these concepts in the way that **RULE:**LANDLORD and its according identity conditions determine.

[34]**RULE:**LANDLORD is equivalent to the predicates Landlord(owner,land) ← OwnsLand(owner,land), RentsLand(tenant,land). Landlord(owner,building) ← OwnsBuilding(owner,building), RentsBuilding(tenant,building).

In cases where the (non-conditional) concepts of **RULE:**R[1] and **RULE:**R[2] are identical, i.e. both conditions of **RULE:**R refer exclusively to one and the same concept C, this rule describes a *relation property* of C. Examples for simple relation properties are symmetry, or transitivity as shown in Example 3.6.

Example 3.6: Transitivity of TALLER_THAN

Let TALLER_THAN:=(2,(EXISTENT,EXISTENT)) be a concept of an ontology O with the semantics that TALLER_THAN[1] is an existent that is higher in stature than the existent TALLER_THAN[2]. The transitivity of TALLER_THAN can be expressed by the conceptual rule



The exposition of rules and conditions as presented in this and the preceding sections has motivated how semantical aspects besides specialization can be modeled for ontology concepts. Rules in Concept Theory are a powerful means for describing simple and complex properties of ontology concepts. As just explicated, rules are apt to define relational properties that are restricted to one ontological kind (i.e. a conceptual relation), as well as concept properties on a general level where a concept is associated with other ontology concepts. Utilizing such properties concerns the application of rules and associated conditions in the context of epistemological processes. The semantics of these processes follow the guidelines of specific conceptual laws, which will be discussed in the following sections.

3.2.3 Laws

Laws are representing methods for managing epistemological processes, in particular knowledge acquisition, knowledge generation and knowledge retrieval.

Definition 3.7: The Concept LAW

The fundamental concept **LAW** is defined as

$$\textbf{LAW} := (\ ) \in \Phi_U$$

and constitutes the generic category of epistemological objects in Concept Theory. Occurrences of **LAW** embody algorithmical solutions to specific epistemological processes.

The semantics of laws must be known to an EOS system in order to interpret them correctly. In this way, occurrences of laws can be used to model the activity of the system. In the following sections we will discuss the different (universal) occurrences of the fundamental concept **LAW** that Concept Theory provides for modeling the system behavior. The crucial questions, here, are:

- *What is the basic set of laws?*

  Laws are a methodological way of approaching application semantics of an actual knowledge processing system. Hence, they represent algorithmical components of the system in a declarative way. We can think of the algorithmical implementation of a knowledge processing system as a set of methods that consist of more basal operations. Such basal operations can concern e.g. finding occurrences or generalizations of an ontological concept within an EOS ontology, while methods using these operations would be to process certain rules or queries. In this context, it is of decisive importance for the epistemological system model to determine the level of granularity in which implemented system components can be addressed by laws, i.e. whether laws are representing single operations or more complex methods – and if so, then which methods.

  The EOS approach promotes a high level of abstraction concerning the representation of epistemological knowledge. Therefore, laws are specifying methods, not basal operations. In fact, Concept Theory supports exactly three kinds of laws (i.e. three universal occurrences of the concept **LAW**), namely **ACQUIRE**, **GENERATE** and **QUERY** (cf. Sections 3.1.3.2, 3.1.3.3 and 3.1.3.4). From a practical point of view, this decision takes into account that it should not be a matter of knowledge representation to determine the system behavior at an algorithmical level. If this were the case, the correct *syntactical* usage of the semantics of concepts, in particular those of conditions and rules, in specific epistemological contexts would reside on the ontology modeler's side. Yet, a domain expert should in any case be freed from having to model correct algorithms on the basis of a particular ontology structure – which is an error prone and costly process. On the other hand, modeling epistemological facts should be held simple and easily applicable to an ontological domain model, and thus be seamlessly integrated into the body of a formal ontology. For these reasons, EOS allows for a mere *semantical* definition of complete epistemological processes, i.e. using conceptual laws with predefined semantics. Immediately, the question arises, what kinds of laws an EOS system must support in order to provide the services of a full-fledged knowledge processing system. The arguments for the different kinds of epistemological processes and their implications have already been presented in detail in Section 3.1. Accordingly, Concept Theory operates with a basic set of laws consisting of **ACQUIRE**, **GENERATE** and **QUERY**, that represent the identified areas of epistemological processes knowledge acquisition, generation and retrieval.

- *What are the semantics of laws?*

  Next to the mere fact that laws are representing methods for managing epistemological processes, the way how this can be achieved is decisive to Concept Theory. We may think of laws as parameterizable entities, similar to method declarations in programming languages. While the abstract parent concept **LAW** of all laws does not possess a specific internal structure, the concise definitions of its occurrences **ACQUIRE**, **GENERATE** and **QUERY**, as we will show in the subsequent sections, each comprise their characteristic component tuples. Continuing the analogy between methods in programming languages and laws, these components define the parameters that are used as an input to the epistemological processes the respective laws are representing. Modeling laws for a domain, then, simply means specifying instances of **ACQUIRE**, **GENERATE** and **QUERY**, using domain concepts as occupants of the roles these kinds of laws are defining. These instances of laws are, analogously, representing specific method calls.

  As **ACQUIRE**, **GENERATE** and **QUERY** are corresponding with epistemological processes, they also embody the paradigm these processes are following. According to Concept Theory, there are two main categories of concepts, namely the set of all conceptual universals Φ_U, and the set of all conceptual particulars Φ_P. The decisive difference between possible algorithmical realizations of these processes lies in the decision, how conceptual particulars are treated. Conceptual particulars are representing concrete facts of a domain, e.g. a particular address **PARTICULAR:**BOB'S_ADDRESS as opposed to definitions of different kinds of addresses, which are conceptual universals such as GERMAN_ADDRESS or ENGLISH_ADDRESS. The two possible perspectives concerning the treatment of particulars can be paraphrased as the *closed world* and the *open world paradigm*:

- *Closed World Paradigm*[35]. If an EOS system will accept solely particulars that structurally comply with its internal domain model, it conforms to the closed world paradigm. In other words, for each particular the system integrates into its ontology, there must exist a suitable universal parent concept that exactly describes its structure, i.e. the particular is a valid occurrence of the universal. Particulars of an unknown structure are rejected by the system.
- *Open World Paradigm*. If an EOS system will also accept structurally unfamiliar particulars, it follows the open world paradigm. This implies that the system must adaptively create new universals for categorizing these particulars.

The interpretation of conceptual laws has to follow one of the following paradigms. For the remainder of this chapter we will concentrate on the closed world paradigm as this view portrays the general position of ontology engineering, where a domain is modeled in advance and domain facts are interpreted according to the domain model, i.e. the respective ontology. We will come back to the possibilities of EOS systems concerning the open world paradigm in our discussion of a wider application area of the EOS approach in Section 5.2.

Based on these preliminary considerations on laws the following sections will define the different kinds of laws and demonstrate the interplay between laws and the algorithmical implementation of an EOS system. Section 3.2.3.1 presents the generic algorithm for EOS systems that reacts on external input. In section 3.2.3.2 we will present the law **ACQUIRE** and its impact on knowledge acquisition. Section 3.2.3.3 introduces the law **GENERATE** for knowledge generation and Section 3.2.3.4 explains how knowledge retrieval can be modeled using the law **QUERY**.

3.2.3.1 Generic Algorithm for EOS Systems

The overall algorithm of an EOS knowledge processing system recurs to the system architecture of the EOS framework (cf. Section 3.1.3).

[35] The closed world paradigm of EOS should not be confused with the 'closed world assumption' (CWA) in logic programming languages (e.g. PROLOG), theorem provers and databases. CWA entails that any proposition P that cannot definitely proven to be true is assumed false (cf. Harcourt: www.harcourt.com/dictionary). The implication of the EOS closed world paradigm, on the other hand, refers to the syntactical properties of conceptual particulars: only particulars that can be structurally associated with existing ontology universals may be integrated into the ontology.

```
while( true ) do
{
   accept input concept c;
   if( c is an occurrence of ACQUIRE )
   {
     // invoke the knowledge acquisition procedure acquire():
     acquire(c);            // see Section 3.2.3.2
   }
   elsif( c is an occurrence of QUERY )
   {
     // invoke the knowledge retrieval procedure query():
     query(c);              // see Section 3.2.3.4
   }
}
```

Figure 3.7: The Generic Algorithm for EOS Systems

As shown in Figure 3.7, an EOS system waits for external input data in the form of concepts as defined by Concept Theory. Generally, input data may consist of concepts extracted from operational data (e.g. Web pages), an EOS ontology, or query concepts that are passed to the system via an according interface (see also Section 3.1.2). New ontology concepts introduced to the system are processed by the knowledge acquisition procedure that we will discuss in the subsequent Section 3.2.3.2. Queries against an EOS system are managed by the knowledge retrieval procedure as presented in Section 3.2.3.4. These two procedures handle the system's direct interaction with external processes. Internally, both, knowledge acquisition and knowledge retrieval, are closely connected to knowledge generation, the creation of new concepts on the basis of according **GENERATE** laws. The respective knowledge generation procedure that governs such operations will be laid out in detail in Section 3.2.3.3.

3.2.3.2 Knowledge Acquisition and the Concept ACQUIRE

Knowledge acquisition from foreign sources presupposes the existence of two individual procedures:

- *Knowledge Extraction Procedure*. This procedure provides an EOS system with new concepts. Strictly speaking, the extraction procedure is not system immanent. It can consist of one or several distinct applications and user interfaces that collect data and transform this information into concepts. In this sense, the extraction procedure realizes the *reliable procedure* that provides *externalist* justification (cf. Section 3.1.1). In conceptual form, the new concepts are handed over to the system. The knowledge extraction procedure proposes a potential category for the new concept within the original ontology, i.e. a possible parent concept that has to be subsequently evaluated by the acquisition procedure of the system (cf. Section 3.1.3.2).
- *Knowledge Acquisition Procedure*. This procedure is system immanent and integrates new concepts into the internal ontology of an EOS system, provided these concepts comply syntactically and semantically with the ontology. Syntactically, the concept structure must be according to the internal structure of the parent concepts as proposed by the extraction procedure. Semantically, all rules and associated conditions of the parent concepts must be met by the new concept. If this is the case, the knowledge acquisition procedure will integrate the new concept into the ontology. Thus, the acquisition procedure realizes the *in-*

ternalist justification of new concepts on the basis of *foundational beliefs*, i.e. the ontological basis of the system.

As pointed out, any new concept C is predetermined in a sense that the reliable extraction procedure suggests a probable category for C. In terms of Concept Theory, this category is a concept of the original ontology, and C is reducible to it. The outset for system immanent knowledge acquisition processes is thus an **ISA** occurrence of the form

ISA:NEW := (2, (C, PARENT_CONCEPT)),

alongside an according mapping between the components of C and PARENT_CONCEPT. This mapping, at its part, is expressed by **ISA** occurrences using concept references (cf. Section 2.2.4.2), e.g.

ISA:MAPPING1 := (2, (C[1], PARENT_CONCEPT[1])),

for specifying the first component of C as an occupant of the first component of PARENT_CONCEPT. The complete set of such mapping concepts may then be combined conjunctively to a condition, here shown for the case of three components, resulting in the three specializations **ISA:**MAPPING1, **ISA:**MAPPING2 and **ISA:**MAPPING3:

AND:TWO_COMPONENTS := (2, (**ISA:**MAPPING1, **ISA:**MAPPING2))
AND:COMPLETE_MAPPING := (2, (**AND:**TWO_COMPONENTS, **ISA:**MAPPING3)).

Note that it is possible that C possesses more components than its PARENT_CONCEPT. If C is a universal, this simply means that it has a richer internal structure than PARENT_CONCEPT. In case C is a particular (i.e. actually **PARTICULAR:**C), the parent concept proposed by the extraction procedure, then, is too general and the acquisition procedure must find a more specialized universal occurrence of PARENT_CONCEPT. The system immanent acquisition procedure is represented by the concept **ACQUIRE**.

Definition 3.8: The Epistemological Concept ACQUIRE

The law **ACQUIRE** is defined as the concept

ACQUIRE:= (2, (EXISTENT, EXISTENT)) $\in \Phi_U$

used for introducing a new concept into an EOS ontology.

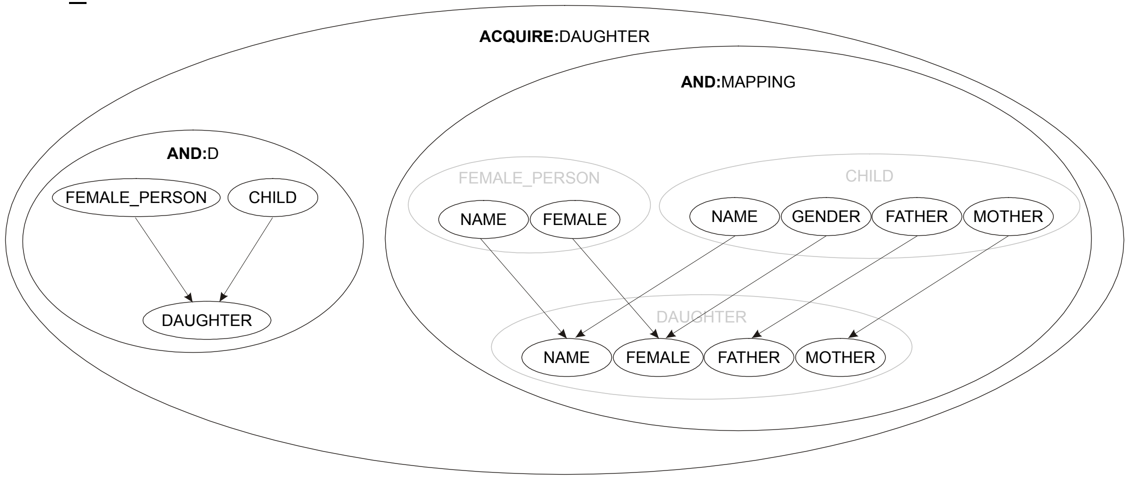
- **ACQUIRE**[1] specifies either a single occurrence of **ISA** (simple inheritance), or an (n-ary) **AND** condition combining several **ISA** occurrences (multiple inheritance). The new concept is the first component of all **ISA** occurrences of **ACQUIRE**[1].
- **ACQUIRE**[2] is either an (n-ary) **AND** condition determining the component mapping for **ACQUIRE**[1], a single occurrence of **ISA** (only one component mapping between the new concept and one of its parents), or left unspecified (no component mapping at all).

Note: Occurrences of **ACQUIRE** that do not specify any component mappings (e.g. because all parent concepts of the new concept are bare concepts) leave **ACQUIRE**[2] unchanged as EXISTENT.

Occurrences of the law **ACQUIRE** are generated by the knowledge extraction procedure and passed to the EOS system. Naturally, the extraction procedure must be aware of the structure of concepts as defined by Concept Theory, and – if available – the system's ontology for identifying specific concepts within source documents. In order to do this, it must be able to perform a mapping between the (raw) data it is processing and the conceptual vocabulary of the ontology. Different ways of achieving this mapping in a concrete implementation of an extraction procedure will be exposited in our discussion of the realization of EOS systems in Chapter 4 (see Section 4.1.3). For explaining the notion and employment of laws, it suffices to presume the existence of such a procedure that provides an EOS system with new concepts, i.e. conceptual universals and particulars described by occurrences of **ACQUIRE**. An illustration of such an **ACQUIRE** law is shown in Example 3.7.

Example 3.7: The Law ACQUIRE:DAUGHTER

Let **O** be a formal ontology, let NAME, FEMALE, GENDER, FATHER, MOTHER,
FEMALE_PERSON := (2, (NAME, FEMALE)), and
CHILD := (4, (NAME, GENDER, FATHER, MOTHER))
be ∈**O**. The law specifying the universal DAUGHTER as an occurrence of FEMALE_PERSON and CHILD is defined as



The law **ACQUIRE:**DAUGHTER thus contains the concept definition of DAUGHTER, its parent concepts and the according component mappings, which comprises all the information needed for integrating the concept DAUGHTER into ontology **O**.

In general, conceptual universals are read in from input documents containing EOS ontologies in a native format, while particulars are mostly extracted from foreign sources. The input ontology in a native format provides complete information about the internal ontology structure, i.e. these concepts can be immediately integrated into the ontology representation of an EOS system as their definition is complete and free from contradictions (per hypothesis, cf. Section 3.1.3.2). On the other hand, the acquisition of new universals and particulars from foreign sources may be more unspecific, i.e. a parent concept as proposed by the extraction procedure may be too general. The aim of the system immanent acquisition procedure is to find the most specialized position for any new concept. Graphically, this means pushing the new concept down along the edges of the ontology graph until its immediate parent concepts are determined. While the ontology graph position of new universals may be somewhere along the **ISA** hierarchy of their parents, particulars must necessarily be integrated as leaves. This integrity constraint of EOS ontologies demands that whenever the extraction procedure has chosen a parent concept that is not at the lowest universal level of the ontology graph, the acquisition procedure must consequently find such a most specialized universal parent concept. Therefore, the system's knowledge acquisition procedure that processes occurrences of **ACQUIRE** must find the most specialized universal parent concept of **PARTICULAR:**C within the ontology graph. As already pointed out, this is especially of importance if **PARTICULAR:**C possesses more components than its proposed parent concept. Then, it is a matter of the acquisition process to determine the correct roles of these components within a yet to be identified (universal) occurrence of the parent concept within the ontology. In case the acquisition procedure cannot resolve such an appropriate parent, the concept will be rejected if the system adheres to the closed world paradigm.

At the heart of the algorithm for the acquisition procedure are the two methods `acquire()` and `determinePosition()`, which are shown in Figure 3.8. Their short description is as follows. The base types used in these (and all subsequent) methods are *`Concept`*, *`ConceptVector`*, *`Integer`*, *`Boolean`*, and *`Void`* with their obvious semantics:

- `acquire(`*`Concept`*` a)`:

  processes `a`, an instance of the law **ACQUIRE**. The components of `a` are being analyzed and handed over to the recursive method `determinePosition()` if the new concept is a valid minting of the proposed parent concepts.

- `determinePosition(`*`Concept`*` c, `*`ConceptVector`*` p, `*`ConceptVector`*` m)`:

  determines the ontology position of a new concept `c`. The concepts in `p` have been determined by the extraction procedure as possible parent concepts of `c`, and `m` comprises the component mappings between `c` and its possible parents. The method `determinePosition()` tests whether there are more specialized concepts within the ontology that are occurrences of the concepts in `p` and can act as parent concepts of `c`. In fact, the task of `determinePosition()` is to find the most specialized parent concepts of `c`, i.e. a structurally compliant lowest level universal that solely possesses particulars as child concepts (cf. Section 2.2.4.2.3 on the existence of single parents for each particular). Only if such a universal can be found, c will be integrated into the ontology (closed world paradigm).

Supporting methods in order of appearance in `acquire()` and `determinePosition()` are:

- `IsValidMinting(`*`Concept`*` c, `*`ConceptVector`*` v, `*`ConceptVector`*` m)`:

  tests whether concept `c` is a valid minting of all concepts in `v`, where `m` specifies the according mappings between `c` and the proposed parent concepts in `v`.

  Return Value: *`Boolean`*.

- `getCommonSpecializations(`*`ConceptVector`*` v)`:

  returns all concepts that form the set of the most general common occurrences of all concepts in `v`.

  Return Value: *`ConceptVector`*.

- `testRules(`*`Concept`*` c, `*`Concept`*` p)`:

  tests rules defined for concept `p` on concept `c`.

  Return Value: *`Boolean`*.

- `propagateMappings(`*`Concept`*` c, `*`ConceptVector`*` p, `*`ConceptVector`*` m, `*`ConceptVector`*` n)`:

  computes and returns all concept mappings between concept `c` and a set of ontology concepts `n` on the basis of mappings `m` between `c` and its parents `p`. The precondition is that all concepts of `n` are immediate child concepts of all concepts in `p` and themselves generalizations of `c`. The component mappings between concepts of `n` and `p` are part of the ontology.

  Return Value: *`ConceptVector`*.

- `isParticular(`*`Concept`*` c)`:

  tests if concept `c` is a particular.

  Return Value: *`Boolean`*.

- `integrate(`*`Concept`*` c, `*`ConceptVector`*` p, `*`ConceptVector`*` m)`:

  adds a new concept `c`, recursively all of its components, and all associated specializations and component mappings to the ontology. At each step, `p` denotes all parent concepts of `c`, i.e. **ISA** occurrences of the form **ISA:**p:=(2,(c,p)) will be added to the ontology, plus the according component mappings `m`.

  Return Value: *`Void`*.

- `count(`*`ConceptVector`*` v)`:

  returns the number of elements of `v`.

  Return Value: *`Integer`*.

- `isLowestLevelUniversal(`*`Concept`*` c)`:

  tests if concept `c` is a universal whose child concepts are, without exceptions, particulars.

  Return Value: `Boolean`.

```
ConceptVector o;  // o is the ontology representation of the system

acquire( Concept a )
// Concept a: Instance of ACQUIRE, holding new concept c and its n parents
{
    ConceptVector specializations := a[1],
                  mappings := a[2];
    Concept       c := specializations[1][1];
    ConceptVector parents :=(specializations[1][2],…,specializations[n][2]);


    if( IsValidMinting(c,parents,mappings)==false )
    {
      // c does not comply with proposed parents;
      // c will not be integrated into o (closed world paradigm)
      return false;
    }
    else
    {
      // Evaluate the most specialized position of c in ontology:
      return determinePosition(c,parents,mappings);
    }
}

determinePosition( Concept c, ConceptVector parents, ConceptVector m );
// Concept c: New concept whose ontology position has to be determined
// ConceptVector parents: Currently proposed parent concept of c
// ConceptVector m: concept Mappings between c and parents
{
    ConceptVector new_parents := getCommonSpecializations(parents);
    Boolean       continue_search := false;

    forall p in new_parents do
    {
      if( testRules(c,p)==false )
      {
        // skip this universal
      }
```

```
        else
        {
            continue_search := true;

            // Continue search with child concepts:
            new_parents_mappings := propagateMappings(c,parents,m,new_parents);
            determinePosition(c,new_parents,new_parents_mappings);
        }
    }

    if( continue_search==false )
    // c does not comply with any child;
    {
        if( isParticular(c)==false )
        {
            // c is a universal.
            // The search is complete, integrate c with current parent(s):
            integrate(c,parents,m);
            return true;
        }
        else
        {
            // c is a particular.
            // Test integrity of parents:
            if( count(parents)==1 && isLowestLevelUniversal(parents[1]) )
            {
                // We found the immediate parent of c
                integrate(c,parents,m);
                return true;
            }
            else
            {
                // c does not comply with a lowest level universal parent,
                // which violates the ontology model;
                // c will not be integrated into o (closed world paradigm)
                return false;
            }
        }
    }
}
```

Figure 3.8: The Acquisition Procedure in Pseudo-Code

The acquisition procedure as defined in Figure 3.8 will integrate a new concept described by an occurrence of **ACQUIRE** into the system ontology if it complies structurally with this ontology. This means that it must be a valid minting of the proposed parent concepts, which is tested in `acquire()` in method `IsValidMinting()`. The new concept must also be semantically valid, i.e. obey all rules of its parents, which is tested in `determinePosition()` using method `testRules()`, which is executed at each recursive step. In case of conceptual particulars, this method also ensures that the new particular is only accepted, if a single, lowest level universal (cf. Section 2.2.4.2.3) can be determined as its sole parent, i.e. if it can be fully classified according to the ontology (closed world paradigm). Thus, `acquire()` and `determinePosition()` represent a correct realization of the acquisition procedure that processes occurrences of the law **ACQUIRE**.

The method `determinePosition()` computes the adequate parent of a new concept by descending recursively down the **ISA** hierarchy of the ontology graph. As the ontology is finite and the ontology graph is free of loops, `determinePosition()` will trivially reach its result the latest after a full traversal of the graph, i.e. its complexity is O(n), where n is determined by the number of elements of the ontology. Certainly, the ontology will gradually grow with each accepted **ACQUIRE** concept. Yet, when processing a specific occurrence of **ACQUIRE**, for each application of `determinePosition()` the number of ontology concepts and thus the paths from the root concept to the graph leaves stays constant and finite.

Note that the order in which new concepts are passed to an EOS system may be decisive for it to get accepted as a valid ontology concept, e.g. a concept C that is a component of another concept A must be acquired before A can be accepted (as the component kinds must be known in order to verify rules and mappings). This is not a matter of the acquisition procedure that simply works on a single new concept at a time, but of the extraction procedure that has to assure that the concepts it produces are handed over to the system in a sensible order. Usually, the extraction procedure will generate a whole set of concepts that can be expected to be interrelated, e.g. taken from the same Web page. If no precise sequence for passing these concepts to the system can be determined from the source document alone, the extraction procedure may retry to pass over previously rejected concepts in a deterministic fashion. A simple algorithm for this is depicted in Figure 3.9.

```
ConceptVector aset := generate set of acquire concepts from source;
Boolean continue := true;

while( continue==true ) do
{
   continue := false;
   forall a in concepts do
   {
     if( acquire(a)==true )
     {
       eliminate a from aset;
       continue := true;
     }
   }
}
```

Figure 3.9: General Algorithm of the Knowledge Extraction Procedure

The general algorithm of the extraction procedure will continue to pass previously rejected concepts to the EOS system as long as at least one new concept has been accepted in the preceding iteration. This behavior takes into consideration that any newly accepted concept may complete the preconditions for admitting another new concept. Only if an iteration did not yield a successful call of the method `acquire()`, the extraction procedure can finally reject all remaining concepts in `aset`.

3.2.3.3 Knowledge Generation and the Law GENERATE

Knowledge generation is the process of producing new concepts from already existing ones by using according laws, i.e. occurrences of the concept **GENERATE**. The EOS approach promotes a materialized view on the domain knowledge an EOS systems is holding, where the modalities of the materialization is defined by occurrences of **GENERATE**[36]. In particular,

[36] It is possible to define **GENERATE** laws for a specific ontology that would yield an infinite materialization. Practically, an EOS system could include mechanisms for detecting such cases and handle them e.g. in that it stops executing the knowledge generation procedure after a fixed number of iterations. However, the present discussion presents the theoretical framework for knowledge generation and at this point we will not restrict this model to a limited class of **GENERATE** laws

this implies that knowledge generation is not an implicit part of complex procedures (such as knowledge retrieval) that rely on a complete view on ontological knowledge, but is regarded as primordial to these procedures. In these terms, an EOS ontology is regarded as an explicit and materialized representation of the domain knowledge the respective EOS system possesses. This is a valuable property, as it allows for simplified query processing and knowledge acquisition techniques.

Domain knowledge naturally consists of concrete facts, represented by conceptual particulars, and general knowledge about the structure and categories of these facts, formalized in Concept Theory as conceptual universals. It is an important question in terms of knowledge generation, what kinds of concepts can sensibly be produced from other concepts, which addresses the semantics of the law **GENERATE**. On a general level, it is possible to produce both, universals and particulars, but it has to be examined what this presupposes in the context of the closed world paradigm we are committed to in this chapter. The closed world paradigm claims that only particulars of a known kind will be integrated into an ontology during knowledge acquisition. This also implies that no new ontological kinds, i.e. domain universals, that would be apt to describe any particulars of a structure unknown to the domain model as such, are needed. Therefore, concerning the creation of new *ontological* universals, there are two possibilities thinkable within the closed world paradigm:

- *Generating structurally compliant specializations from already existing universals*.

  This method produces new universals by specializing one or several components of an ontology universal. For example, from PERSON:=(2,(NAME,GENDER)) the system could generate specializations PERSON_SPEC1:=(2,(NAME,FEMALE)) and PERSON_-SPEC2:=(2,(NAME,MALE)) if the ontology includes the information that *male* and *female* are *genders*, i.e. **ISA:**1:=(2,(MALE,GENDER)) and **ISA:**2:=(2,(FEMALE,GENDER)). The combination of component specializations can be driven to an arbitrary level, even to particulars – provided that at least one component remains universal, e.g. PERSON_-SPEC3:=(2,(**PARTICULAR:**BARRY_NAME,MALE)) would still be a valid universal derived from PERSON. Yet, the benefit of such new universals is questionable on an ontological level as they do not provide a better or more complete domain model. The number of different particulars that can be categorized according to these new universals does not increase. In fact, all domain particulars have already been classified in a way (i.e. according to the original ontology) that allowed for the generation of structurally compliant universals in the first place. Therefore, this option can be left aside.
- *Generating universals as combinations of already existing ones*.

  The second method for generating ontological universals from ones already existing within an ontology is by introducing new specializations of universals that are themselves not occurrences of each other. In this way, arbitrary parent combinations are possible, yet not necessarily feasible, e.g. concepts COLOR and TALLER_THAN could be combined to yield a common specialization COLOR_TALLER_THAN whose semantics remain questionable. Moreover, there are no clear criteria for component combinations, i.e. questions on how the component order of the new universal should be determined from its parents, or whether parent components of the same kind should be merged to a single occupant in the new universal, are not resolvable. This reveals two basic problems, namely that simply creating the set of all possible universal combinations does not necessarily yield sensible universals, and consequently, that the majority of these new universals would not mirror the structure and semantics of concrete domain facts (i.e. particulars). Yet, this is exactly

that cannot possibly produce infinite materializations. We will further detail in different modes of materialization later on in Sections 3.2.3.3.2 and 3.2.3.3.3.

what an ontology should provide. Since the closed world paradigm demands that universals have to be defined prior to any acquisition of particulars, the structure of unknown but actually occurring domain facts is not at disposition for determining inference rules. For this reason, this option can also be rejected.

In conclusion, neither option of creating new *ontological* universals is advisable with respect to the closed world paradigm. Under this paradigm, the overall ontology structure as defined by conceptual universals will only be altered during knowledge acquisition and remains stable during knowledge generation processes. Hence, on the ontological level, the automatic generation of new concepts is restricted to particulars. Concerning *epistemological* universals, i.e. rules and laws, the closed world paradigm yields a restricted perspective:

- *Generating rules and laws*.

  Ontologies under the closed world paradigm represent static domain models in a sense that they are only altered structurally by knowledge acquisition processes, and quantitatively by adding particulars, which may also be provided during knowledge acquisition, or automatically computed by the knowledge generation procedure. The presumption, here, is that a structurally complete domain model is being handed over to the system, which then has to materialize the maximum set of domain facts, i.e. its particulars. Thus, integrating automatically generated rules or laws into the system ontology is beyond the closed world paradigm. There is one special case, though, that makes creating new occurrences of specific laws feasible, yet only during query processing, and not materialized in the ontology. In particular, the EOS approach allows to define a preprocessing for occurrences of the law **QUERY** that are handed over to the system before the actual processing of the query takes place. The exact procedure for this semantic query rewriting is laid out in detail in the exposition of knowledge retrieval in Section 3.2.3.4. For the present discussion of knowledge generation it suffices to record that the closed model paradigm allows for an restricted automated creation of epistemological concepts, namely **QUERY** occurrences that are not integrated into the system ontology but are only used (temporarily) while executing the knowledge retrieval procedure.

Knowledge generation under the closed world paradigm thus concerns particulars and occurrences of the law **QUERY**. The general definition for the conceptual law that specifies the modalities of knowledge generation is given in Definition 3.9.

Definition 3.9: The Epistemological Concept GENERATE

The law **GENERATE** is defined as the concept

$$\textbf{GENERATE} := (\ 2,\ (\ \text{EXISTENT, EXISTENT}\ )\ )\ \in \Phi_U$$

where

- **GENERATE**[1] specifies either an n_0-ary **AND** condition[37] that is the outset of a concept generation step, i.e. whenever **GENERATE**[1] is *true* within an ontology, the new concepts as determined in **GENERATE**[2] can be generated.
- **GENERATE**[2] is an n_0-ary **AND** condition of concepts that can be generated from **GENERATE**[1].

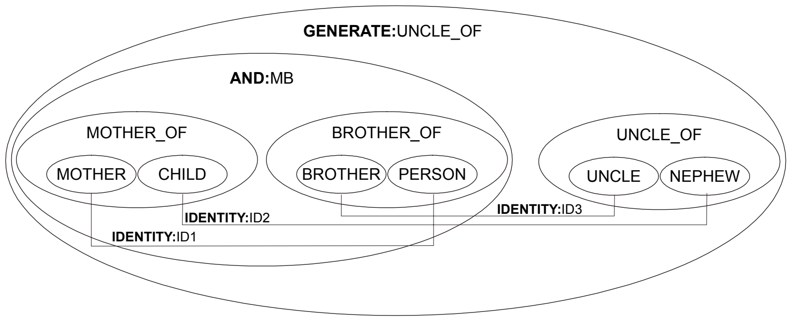
[37] The expression '*n_0-ary **AND** condition*' is a short notation for referring to either a single, non-conditional concept or an n-ary **AND** condition of such concepts.

Note: Correlations between concepts of **GENERATE**[1] and **GENERATE**[2] are expressed using identity conditions, that allow for the concepts of **GENERATE**[2] to be minted according to the particulars that act as an input to **GENERATE**[2].

Definition 3.9 describes the basic structure for defining the knowledge generation procedure in Concept Theory. When applying a specific **GENERATE** law **GENERATE:**G, its first component **GENERATE:**G[1] is minted by an according input set of concepts. This minting can then be propagated to **GENERATE:**G[2] via the respective identity conditions on **GENERATE:**G. The resulting concepts of **GENERATE:**G[2] can subsequently be integrated into the ontology or themselves used as an input to further epistemological processes. An exemplary occurrence of **GENERATE** is demonstrated in Example 3.8.

Example 3.8: The Law GENERATE:UNCLE_OF

Let **O** be a formal ontology, let PERSON, MOTHER, CHILD, BROTHER, UNCLE, MOTHER_OF:=(2,(MOTHER,CHILD)), BROTHER_OF:=(2,(BROTHER,PERSON)) and UNCLE_OF:=(2,(UNCLE, NEPHEW)) be ∈**O**. The law that generates particular instances of UNCLE_OF from particular instances of MOTHER_OF and BROTHER_OF is defined as



The law **GENERATE:**UNCLE_OF determines that an instance of UNCLE_OF can be produced from two instances of MOTHER_OF and BROTHER_OF, given the identity condition **IDENTITY:**ID1 is satisfied between them, denoting that the MOTHER of CHILD has also a brother (that accordingly is the UNCLE of the CHILD). Consequently, **IDENTITY:**ID1 is the precondition that has to be satisfied by a concrete concept minting of **GENERATE:**UNCLE_OF[1], while **IDENTITY:**ID2 and **IDENTITY:**ID3 define how the new instance of UNCLE_OF is to be created from this minting.

Note that particular producing occurrences of **GENERATE** are syntactically similar to occurrences of **RULE**, yet they differ in their semantics. A **RULE:**R states integrity constraints that must be fulfilled by all occurrences of the non-conditional concepts in **RULE:**R[1], while a **GENERATE:**G is producing particulars whenever the condition **GENERATE:**G[1] is met by a set of particulars. It may be unwanted to use the total set of rules of an ontology for automatic knowledge generation, as this might lead to an unwanted materialization of concepts. **GENERATE** laws are a means of controlling this process.

A further difference between conceptual rules and particular producing **GENERATE** laws is that the latter are subject to several syntactical restrictions, such as, e.g. without exception, all concepts of the right-hand side of particular producing **GENERATE** laws must be fully determined by left-hand side concepts. In particular, this means that all universals of the right-hand side of such a **GENERATE** instance must be determined by components of the left-hand side by according identity conditions, either directly, or through their components. As depicted in Example 3.8, this is the case in **GENERATE:**UNCLE_OF: both components of UNCLE_OF are related to concepts of **GENERATE:**UNCLE_OF[1] through identity conditions **IDENTITY:**ID2 and **IDENTITY:**ID3.

3.2.3.3.1. Performing Knowledge Generation

During each knowledge generation step, an occurrence **GENERATE:**G of the law **GENERATE** is used as a schema for creating new concepts on the basis of input concepts that represent a valid minting for **GENERATE:**G[1]. The respective method `generate()` for such a generation step is listed in Figure 3.10. Its short description is:

- `generate(Concept c, Concept g)`:

  processes `g`, an instance of the law **GENERATE** using concept `c` as a minting of `g[1]`, the first component of `g`. The minting of `g[1]` is propagated to `g[2]`, respecting the identity conditions defined for `g`. It returns the resulting minting for `g[2]`.

  Return Value: `Concept`.

Supporting methods in order of appearance in `generate()` are:

- `computeMappings(`*`Concept`*` c, `*`Concept`*` s)`:

  computes the concept mapping of two concepts `c` and `s` according to their definition and the ontology.

  Return Value: `ConceptVector`.

- `mint(`*`Concept`*` t, `*`Concept`*` s, `*`ConceptVector`*` m)`:

  mints a concept `t` according to another concept `s`, respecting the mapping `m` between them.

  Return Value: `Concept`.

```
generate( Concept c, Concept g )
// Concept c: A valid minting of g[1]
// Concept g: Instance of GENERATE
//
// precondition: All concepts of g[2] transitively related to concepts in
//               g[1] by identity conditions within g[2] to concepts that
//               are directly related to concepts in g[2] are themselves
//               directly related to the respective concepts of g[1].
{
   Concept source := g[1],
           target := g[2];

   ConceptVector m := computeMappings(c,source);

   // mint component(s) of source:
   source := mint(source,c,m);

   // mint bound component(s) of target from minted source component(s):
   m := computeMappings(source,target);
   target := mint(target,source,m);

   return target;
}
```

Figure 3.10: General Algorithm for a Knowledge Generation Step

The outset of applying such knowledge generation steps is a 'stable' status of the internal knowledge representation of an EOS system. In terms of the materialized domain model the ontology is representing this means that all **GENERATE** laws have already been applied to all ontology particulars, so that any new employment of these laws will not yield any new concepts. Trivially, this is true with any EOS system before reading in its EOS ontology. Once this initialization phase has been carried out, all generation laws can subsequently be applied to the ontology (provided the ontology already contained particulars), resulting in the stable status of the system. This process is visualized in Figure 3.11. With the acquisition of concept C, this new concept is being integrated into the Ontology. Subsequent knowledge generation steps are using C for materializing more concepts on the basis of according **GENERATE** laws, which finally results in the new stable status Ontology'.

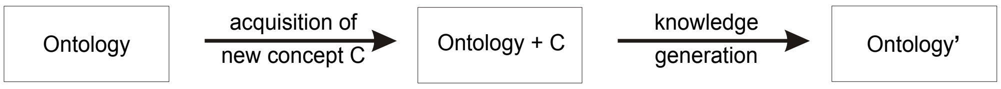


Figure 3.11: Knowledge Generation in EOS Ontologies

This procedure implies that with any newly acquired particular, the ontology's generation laws must be tested in order to guarantee complete information. The algorithmical structure of the knowledge generation procedure is illustrated in Figure 3.12. The method `manageGeneration()` accepts a concept, either a particular or a **QUERY** occurrence, that triggers the according concept generation processes (indicated by the two **`while`** loops). It uses the supporting method `push()`:

- `push(`*`Concept`*` c, `*`ConceptVector`*` v)`:

  adds concepts `c` to the concept vector `v` and returns the enlarged vector.

  Return Value: *`ConceptVector`*.

```
manageGeneration(Concept c)
// Concept c: a new particular or QUERY occurrence
{
   ConceptVector cset := (); push(c,cset);
   Boolean continue := true;

   if( IsParticular(c)==true )
   {
     // perform particular generation:
     while( continue==true ) do
     {
       ...      // see Section 3.2.3.3.2
     }
     // integrate all particulars newly created in this while loop
     // (and stored in cset) into the ontology:
     integrate cset into ontology (omitting c);
   }
   else
   {
     // perform universal generation (semantic query rewriting):
     while( continue==true ) do
     {
       ...      // see Section 3.2.3.3.3
     }
   }
}
```

Figure 3.12: The Overall Structure of the Knowledge Generation Procedure

As suggested by the algorithm structure of Figure 3.12, generating new concepts can be a continued process, i.e. concepts generated by application of a **GENERATE** law may trigger other concept generation steps. It is therefore of great importance to ensure that this overall process terminates in all cases. Hence, a substantial part of this section will thoroughly examine the modalities of such sequences of generation steps within concept generation under the closed world paradigm. Section 3.2.3.3.2 discusses particular generation, while producing **QUERY** instances for semantic query rewriting[38] is the subject of Section 3.2.3.3.3. The line of reasoning in both sections first stresses modeling constraints for **GENERATE** laws and then draws conclusions for the manner in which they should be executed.

3.2.3.3.2. Generation of Particulars under the Closed World Paradigm

GENERATE laws are a means for formulating patterns that define how new particulars can be produced from already existing ones. Knowledge generation, here, means therefore materializing new domain facts that can be deduced from other facts. The aim of the knowledge generation procedure is to maximize the set of domain particulars on the basis of **GENERATE** laws. Hence, using occurrences of the concept **GENERATE**, a domain expert can determine which kinds of facts are to be (automatically) materialized from other ontology particulars. In order to motivate the algorithm of the concept generation procedure that processes occurrences of **GENERATE**, we will briefly sketch the outset for modeling such concepts. An occurrence of the law **GENERATE** depicts a production step, specifying the input concepts in its first component, and the desired output concepts in its second component. For a better readability we will use a graphical notation for illustrating single production steps and sequences.

[38] Semantic query rewriting in EOS systems entails the application of **GENERATE** laws for modifying queries passed to the system in a predefined and structured manner, which allows for amending query semantics and for modeling personalized access to the ontology data. For a more detailed discussion on semantic query rewriting consult Section 3.2.3.4.1 where it is presented in the context of our treatment of knowledge retrieval.

Concepts of **GENERATE**[1] and **GENERATE**[2], along with their components are depicted in the already introduced fashion. **GENERATE** occurrences themselves are represented by labeled arrows. In this way, the law **GENERATE:**UNCLE_OF of Example 3.8 can therefore be expressed as displayed in Figure 3.13.

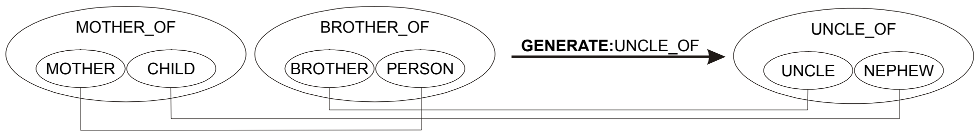


Figure 3.13: Alternative Representation of the Law ***GENERATE:**UNCLE_OF*

The schematic way of representing such an **GENERATE** occurrence is shown in Figure 3.14. We will use this level of abstraction for discussing the different structures of **GENERATE** laws.

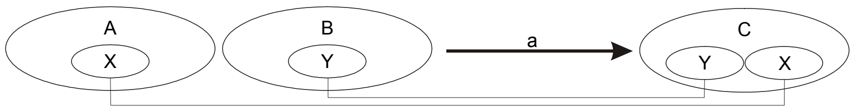


Figure 3.14: Abstract Form of the Law ***GENERATE:**UNCLE_OF*

Note that in this schematic way of expressing **GENERATE** laws, we will generally use single components for portraying sets of components, and only use more than one component in cases where differentiating between different concept components is important for the argument. Making use of this graphical method, we will subsequently develop the constraints and structure of knowledge generation under the closed world paradigm. In the course of this discussion, we will treat (a) determinateness, (b) self-containment of concepts, (c) concept nesting, (d) structurally identical copies of concepts, (e) structural additions, and (f) concept generation chains. In this way, syntactical correctness of **GENERATE** laws (determinateness), general syntactical restrictions (self-containment, concept nesting, structurally identical copies, structural additions), and the impact of repeated executions of **GENERATE** laws (concept generation chains) is being examined. The resulting treatment of particular generation under the closed world paradigm draws upon the two determinants of concepts: specialization (i.e. the **ISA** hierarchy *among* concepts) and parthood (i.e. the component structure *within* concepts).

a) Determinateness

As the closed world paradigm only allows for the generation of particulars, within any **GENERATE** law **GENERATE:**G, the components of **GENERATE:**G[2] must be fully determined:

$\forall$ **GENERATE:**G:

$\forall C \in \phi_{\text{GENERATE:G[2]}}, C \in \Phi_U$:

($\exists P \in \phi_{\text{GENERATE:G[1]}}$:

$\exists$ **IDENTITY:**ID:=(2, (**GENERATE:**G<P>,**GENERATE:**G<C>))) $\vee$

($\forall COMP \in \phi_C, COMP \in \Phi_U \; \exists Q \in \phi_{\text{GENERATE:G[1]}}$:

$\exists$ **IDENTITY:**ID:=(2, (**GENERATE:**G<Q>,**GENERATE:**G<COMP>))), where **GENERATE:**G<X> denotes the concept reference of a concept X within **GENERATE:**G, i.e. all conceptual universals of **GENERATE:**G[2] must be determined by components of **GENERATE:**G[1] by identity conditions, either directly, or via their components. Without loss of generality we presuppose, that these identity conditions between **GENERATE:**G[1] and **GENERATE:**G[2] are materialized, even in cases, where identity conditions within **GENERATE:**G[2] would suffice to propagate an identical **GENERATE:**G[1] minting for a concept of **GENERATE:**G[2] to other concepts of **GENERATE:**G[2]. As already stated, determinateness also forbids **ISA** occurrences in **GENERATE:**G[2], as all **ISA**[2] components are per definition universals.

b) Self-containment

According to the formal definition of concepts, particulars cannot contain themselves (cf. Definition 2.4), an obvious constraint that is depicted in Figure 3.15. Figure 3.15a shows a conceptual universal that contains itself, which is very well possible, e.g. an ORGANIZATION can be part of another, bigger ORGANIZATION. Yet, this ORGANIZATION cannot be a proper part of itself, which renders Figure 3.15b incorrect. The identity condition insinuates that for all concepts that mint A, their component must be identical with themselves. With universals this poses a true statement as the ORGANIZATION example has shown. This is why Figure 3.15a is a semantically correct concept. Only when concepts are minted by particulars, which is the case with **GENERATE** laws under the closed world paradigm, minting concepts must not be self-containing. However, this is what the identity condition expresses for A particulars. Consequently, no occurrence of **GENERATE** may contain concepts of this type.

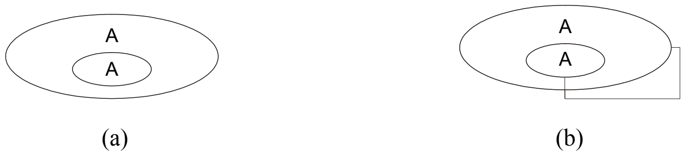


Figure 3.15: Legal and Illegal Self-containment of Concepts

Self-containment of concepts must also be avoided in the context of actually producing concepts, i.e. no **GENERATE** law may be formulated in a way that would create concepts that are nested within themselves. The schema for such illegal nestings is depicted in Figure 3.16. The according integrity constraint for **GENERATE** occurrences is thus:

$\forall$ **GENERATE:**G, $\forall C_i \in \phi_{\textbf{GENERATE:}G[1]}$, $\forall C_j \in \phi_{\textbf{GENERATE:}G[2]}$:

$\neg$ (**IDENTITY:**ID1:=(2,(**GENERATE:**G<C_i>,**GENERATE:**G<C_j>)) $\wedge$
IDENTITY:ID2:=(2,(**GENERATE:**G<C_i>,**GENERATE:**G<C_k>))),

where $C_k \in \phi_{C_j}$,

i.e. no concept in **GENERATE:**G[2] may be related by an identity condition to a concept in **GENERATE:**G[1] if one of its components is also identified with that same concept of **GENERATE:**G[1].

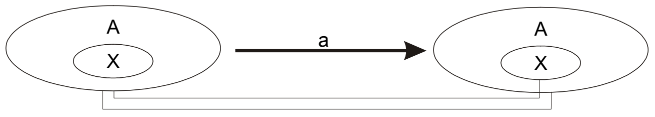


*Figure 3.16: Illegal Self-containment in **GENERATE** Laws*

c) Concept Nesting

Closely related to cases of self-containment in **GENERATE** occurrences is illegal concept nesting as depicted in Figure 3.17. The schema differs from self-containment in that A and B are not set equal by an identity relation. This implies that a new particular of B is generated by nesting A into it, a process that can be infinitely continued if either A = B, or B is a specialization of A, which not only syntactically produces a vicious recursion but also is semantically incorrect. Recurring to the example of an ORGANIZATION nested within an ORGANIZATION, this generation schema would produce an infinite number of encompassing ORGANIZATION instances, i.e. starting from a **PARTICULAR:**O a new **PARTICULAR:**O2 would be created that contains **PARTICULAR:**O, subsequently a **PARTICULAR:**O3 from **PARTICULAR:**O2, etc. Yet, this would not yield a true statement about the original **PARTICULAR:**O but be an arbitrary process. An *organization* **PARTICULAR:**O may very well be nested within other *organization*, however this cannot be decided independently from other domain facts. It follows the integrity constraint for **GENERATE** laws:

$\forall$ **GENERATE:**G, $\forall A \in \phi_{\textbf{GENERATE:}G[1]}$, $\forall B \in \phi_{\textbf{GENERATE:}G[2]}$:

(B is a valid occurrence of A) $\Leftrightarrow$

$\nexists B_i \in \phi_B$: **IDENTITY:**ID$_i$:=(2,(**GENERATE:**G<A>,**GENERATE:**G<B_i>)), i.e. concepts of the same kind may not be nested.

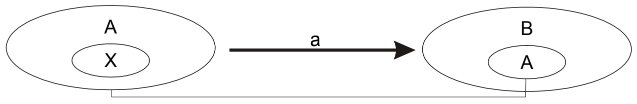


Figure 3.17: Illegal Concept Nesting in ***GENERATE*** *Laws*

d) Structurally Identical Copies

Generating particulars under the closed world paradigm results in new particulars that are numerically distinct[39] from all other concepts of an ontology. Theoretically, it is possible to produce an infinite number of structurally identical particulars from a single concept. The schema of such a **GENERATE** occurrence is illustrated in Figure 3.18. Given that universal B is a specialization of A (or B = A) and the generation of B particulars is solely dependent of A and its components, this **GENERATE** law produces with each application an instance of B that can itself be used again as an input to A. Consequently, this **GENERATE** occurrence produces a vicious recursion, which leads to the integrity constraint

$\forall$ **GENERATE:**G, $\forall A \in \phi_{\textbf{GENERATE:}G[1]}$, $\forall B \in \phi_{\textbf{GENERATE:}G[2]}$:

(B is a valid occurrence of A)

$\wedge\ \forall B_j \in \phi_B\ \exists A_i \in \phi_A$: **IDENTITY:**$ID_i$:=(2,(**GENERATE:**G<$A_i$>,**GENERATE:**G<$B_j$>)) $\Rightarrow$

$\exists$ **IDENTITY:**ID:=(2,(**GENERATE:**G<C_k>,**GENERATE:**G<B>)),

where $C_k \in \phi_{\textbf{GENERATE:}G[1]} \setminus \phi_A$,

i.e. whenever B represents an identical structural copy of A whose components are determined solely from A's components in their original order, it must be assured, that B itself is not generated from A, but depends on another concept C_k that is not a part of A. In other words, B is not *generated* from A, it only receives A's components, and is actually generated (or altered) from C_k. **GENERATE** laws of this kind can be used in order to change the properties of already existing particulars B in terms of a newly acquired concept A, or for generating a new concept B from concepts C_k and minting it according to another concept A.

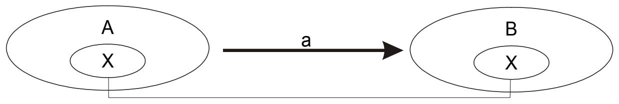


Figure 3.18: Illegal Structurally Identical Copies in ***GENERATE*** *Laws*

Generally, structurally identical particulars of the same universal are very well possible, e.g. two persons may possess the same ADDRESS, as they are roommates (see Figure 3.19). What has to be avoided is that an infinite number of such structurally identical but numerically distinct copies is produced. This is the case with the schema of Figure 3.18, as B is determined

[39] The term *numerically distinct* refers to particulars of the same type and properties, e.g. two undistinguishable wine glasses, or, in terms of object-oriented programming, two objects of the same class with identical attribute values. Numerically distinct entities cannot be distinguished in terms of their own structure, but only using external properties, such as space/time for real-world objects, or object identifiers during program execution for class instances.

by A alone. The generation step of the example in Figure 3.19, on the other hand, is legal because the new instance of PERSON has been delivered from another concept, ROOMMATE.

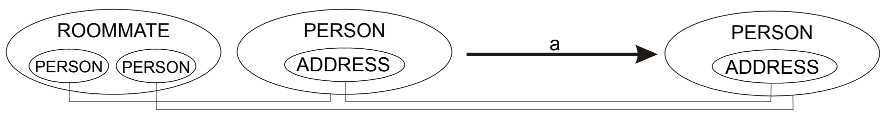


Figure 3.19: A Legal Structurally Identical Copy

e) Structural Additions

Concepts in Concept Theory possess a fixed component structure. Hence, it is not allowed to define structural additions to concepts as shown in Figure 3.20. In both schema examples of Figure 3.20, a new component is added to concept A. Strictly speaking, such additions do not change A's structure but define a new type of concept, i.e. a new universal, that is falsely declared as of kind A. Figure 3.20 demonstrates two variations of structural additions. While Figure 3.20a does define a structurally extended version of A that is independent of any other concept of the left-hand side of the generation schema, the schema of Figure 3.20b assumes a second concept A supplying the additional component Y. Both schemas are illegal because of this supplementary concept Y, where Figure 3.20a also violates the determinateness constraint (given that Y is a particular) that all conceptual universals of the right-hand side of **GENERATE** laws must be determined by their left-hand side components. The general description of this obvious constraint is that no concept C:=(n, L) may be redefined within Φ, and is made explicit here only for reasons of completeness.

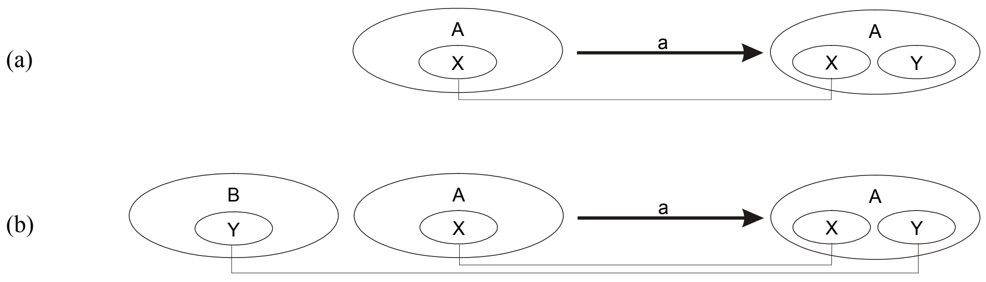


Figure 3.20: Illegal Structural Additions in **GENERATE** *Laws*

f) Concept Generation Chains

The preceding constraints are complete in that they avoid vicious recursion that may be defined within single **GENERATE** laws. The different aspects these constraints are aiming at for any **GENERATE** occurrence are:

- *syntactical correctness* in terms of Concept Theory for **GENERATE** laws and their component, i.e. avoiding illegal self-containment of single concepts, and redefinitions of concepts,
- *determinateness* according to the closed world paradigm,
- *non-recursive generation steps*, i.e. avoiding **GENERATE** laws that may produce particulars that can reenter the same generation steps infinitely.

The possible sources for recursive generation steps are an immediate consequence of the structural definition of concepts as C:=(n, L) in Concept Theory:

- producing (structurally identical) copies of C on the basis of itself,
- illegally adding new components, thus increasing n and L, and
- misusing the component vector L for an infinite nesting of concepts of the same kind.

All of these possibilities for producing infinite loops within single **GENERATE** laws are prevented from actually creating such hazardous concept generation by the previously discussed constraints. The remaining exercise is to assure, that no cause for loops can reappear when executing several different **GENERATE** laws consecutively. As a preliminary consideration in this respect, structural additions to concepts are in itself illegal, i.e. they play no role in concept generation chains as they must not be used in any occurrence of **GENERATE**. Concept nesting in such generation chains can also be disregarded as the according constraint already forbids them. This is illustrated in Figure 3.21 in an example with three generation steps a, b and c, each representing a distinct **GENERATE** occurrence. Step a is correct, provided A ≠ B, and no component of A is a specialization of B. Analogously, step c can only be valid if X ≠ A, and no component of X is a specialization of A. Yet, according to step b, exactly this is the case. In fact, b denotes an illegal **GENERATE** occurrence, as X is a component of its left-hand side concept B (through step a), and B at its part again a component of X according to the right-hand side of b. For nesting A (falsely) into itself, an unbroken chain of generation steps that are nesting concepts into each other is necessary. However, this chain of generation steps must also necessarily contain one step that produces an X, the component of A used in c. Yet, this preceding step would necessarily have to be of form b, and b represents, as just shown, an illegal **GENERATE** occurrence that cannot be modeled at all. Thus, concept nesting holds no risks for loops in concept generation chains.

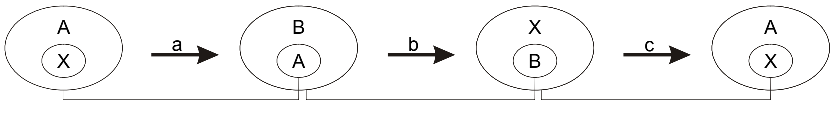


Figure 3.21: Illegal Concept Nesting in a Chain of ***GENERATE*** *Laws*

The only remaining possible cause for a vicious recursion within concept generation chains is therefore the creation of structurally identical copies, which has to be avoided by the knowledge generation procedure. A simple version of such an hazardous chaining is presented in Figure 3.22.

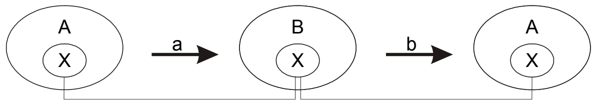


Figure 3.22: Creation of Structurally Identical Copies in a Chain of ***GENERATE*** *Laws*

As Figure 3.22 illustrates, the knowledge generation procedure must assure that generation steps of type b may not be executed with the same particular X that has been produced by a. Put more generally, no particular of kind A may be produced solely from the components (in the original component tuple order) of another particular of A (or a specialization thereof). This is portrayed in Figure 3.23a where generation step b would yield a structurally identical but numerically distinct copy of the original particular of kind A was the outset of the former step a. Note that this is not the case with Figure 3.23b because generation step c yields a new particular of kind A whose components represent a permutation of the components of the original particular, i.e. the new concept is not identical with the original particular. The obvious precondition in this case is that X = Y, but we left the naming conventions of Figure 3.23a in order to stress the argument. A further illustration is given in Figure 3.23c, again presupposing that X = Y. It shows that for a symmetric relation A, one execution of generation step d that yields the symmetric counterpart of an original particular is legal while its second application would, again, produce an illegal copy.

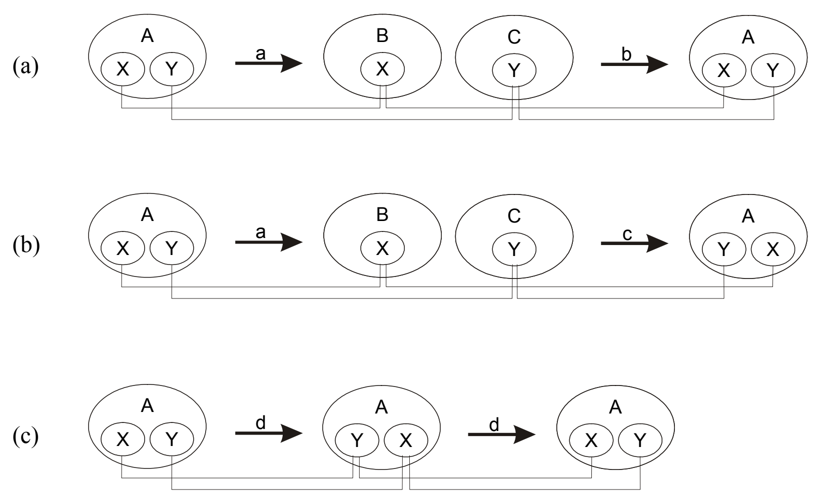


Figure 3.23: Illegal and Legal Concept Creation in a Chain of **GENERATE** *Laws*

Formally, this demands the following constraint for chained executions of **GENERATE** laws:

Let S_i be a sequence of i generation steps $s_j:=(L_j,R_j)$ representing the execution of a respective **GENERATE** occurrence **GENERATE:**G:=(2,(L,R)) with particulars R_j generated from particulars L_j according to **GENERATE:**G, i.e. L_j and R_j are the specific concept mintings of L and R at s_j (after the execution of s_j). Let $\{S_j\}$, $j \leq i$ denote the set of all distinct concept mintings within the sequence $s_1, s_2, \ldots, s_j \in S_i$, and let $\phi_{\{S_j\}}$ be the set of all (recursive) components of $\{S_j\}$. Let K_C be the kind of a concept $C \in \{S_j\}$ as determined by the **GENERATE** occurrences that are used within S_j. Let $\mathbf{O}$ be an ontology and $\mathbf{O}_P$ the set of particulars in $\mathbf{O}$. It follows:

$\forall s_j, s_{j+1} \in S_i$, $1 \leq j < i$, $\forall C:=(n_C,L_C) \in \phi_{\mathbf{R}_{j+1}}$:

$\nexists P:=(n_P,L_P) \in \{S_j\} \cup \phi_{\{S_j\}} \cup \mathbf{O}_P$: $(\exists U \in \mathbf{O}$: K_C, K_P are valid occurrences of $U) \wedge n_C=n_P \wedge L_C=L_P$,

i.e. no new particular may be created that possesses a component tuple identical to any other concept within the sequence or the ontology, given that they are type compatible, thus occurrences of the same universal (which includes identity).

The algorithm of the knowledge generation procedure can use this constraint for avoiding loops in chains of generation steps. As already mentioned, the outset of the knowledge generation procedure is an input concept, a newly acquired particular or an occurrence of the law **QUERY**. Depending on the kind of concept, i.e. whether it is a particular or a query, the procedure will act according to the respective constraint. In case of particulars, the above constraint holds. The procedure thus needs to keep track of the information needed to test this constraints. This can be achieved by marking all concepts that take part in concept generation steps. First of all, this will be the newly acquired particular that has been handed over to the generation procedure, and secondly, all other concepts that join this particular as an input to generation steps. These can be ontology concepts, as well as particulars produced in previous steps.

As particulars are fed to the system by the knowledge acquisition procedure, it is in this context that the knowledge generation procedure is triggered. Specifically, this is done inside the method `integrate()` that is adding newly acquired concepts to the ontology. In case of a new particular `c`, `integrate()` passes `c` to the knowledge generation procedure `manageGeneration()`. The algorithmic solution for managing particular generation that has not been further fleshed out in Figure 3.12 is listed here in Figure 3.24. Supporting methods are:

- `mark(Concept c)`:

  marks `c` (e.g. by inserting it into a global concept vector).

  Return Value: *`Void`*.

- `isMarked(Concept c)`:

  tests if `c` has been marked (e.g. by inspecting the respective global concept vector).

  Return Value: *`Boolean`*.

- `ban(`*`Concept`*` c, `*`Concept`*` g)`:

  marks `c` as already used in **GENERATE** law `g` (e.g. by inserting them as a tuple into a global vector).

  Return Value: *`Void`*.

- `isBanned(`*`Concept`*` c, `*`Concept`*` g)`:

  tests if `c` has been marked with **GENERATE** law `g` (e.g. by inspecting the respective global concept vector).

  Return Value: *`Boolean`*.

- `isStructuralCopy(`*`Concept`*` c, `*`ConceptVector`*` cset)`:

  tests the loop constraint for particular `c` on the set of concepts in concept vector `cset` and the ontology.

  Return Value: `Boolean`.

```
// Concept c: a new particular
// ConceptVector cset: a concept vector containing c
// Boolean continue: determines whether execution continues

while( continue==true ) do
{
    continue := false;
    ConceptVector gset := all GENERATE laws in the ontology that contain c;
    forall g in gset do
    {
      ConceptVector mset := all mintings for g computed from cset∪O_P;
      mark(all particulars in mset);

      forall m in mset do
      {
        if( isBanned(m,g)==true )
        {
          // skip this generation step
        }
        else
        {
          // we found a minting that may be executed,
          // i.e. a new while loop iteration is possible
          // with the concepts that will be now generated.
          continue := true;

          // execute this generation step:
          Concept new := generate(m,g);

          Boolean doban := true; // determines whether m must be banned
          forall n in ϕ_new do
          {
            if( isStructuralCopy(n,cset)==true )
            {
              // skip this concept, it is an illegal structural copy
            }
            else
            {
              // n is a legally created new concept,
              // i.e. m produced a valid new particular with g
              // and must not be banned.
              doban := false;
              integrate n into the ontology;
              mark(n);
              cset := push(n,cset);
            }
          }
          if( doban==true ) ban(m);
        }
      }
    }
}
```

Figure 3.24: The Knowledge Generation Procedure for Particulars

The **while** loop terminates once no application of **GENERATE** laws on particular c and on the concepts produced within the accompanying generation steps yields any new particulars. In order to decide whether the newly generated concepts are structural copies of already existing particulars (in the ontology or in the vector cset of all new concepts), the method isStructuralCopy() is invoked for each of the currently created concepts (see Figure 3.25 for a listing of this method). If all new concepts of a generation step are structural copies, the respective **GENERATE** minting is marked as 'banned' by calling ban(), i.e. the knowledge generation procedure will not use this minting again on the specific **GENERATE** occurrence. This manner of processing **GENERATE** laws ensures that no structural copies of existing concepts are being produced.

```
isStructuralCopy( Concept c, ConceptVector cset )
{
   Boolean return_value := true;

   forall component in $\phi_{c[1]}$ do
   {
      forall x in cset$\cup\mathbf{O}_P$ do
      {
         if( haveCommonParent(c,x)==false ||
               haveIdenticalStructure(c,x)==false )
         {
            return_value := false;
         }
      }
   }
   return return_value;
}
```

Figure 3.25: Testing the Particular Generation Constraint

Thus, in combination with `isStructuralCopy()` of Figure 3.25, the methods shown in Figure 3.24 correctly implement particular generation under the closed world paradigm as motivated in the argument of this section. As looped execution of generation steps is avoided and an ontology is necessarily a finite set of concepts, the algorithm terminates after all possible mintings related to the initial concept `c` that triggered the generation procedure in the first place have been tested. Therefore, the maximum complexity of this procedure is reached in cases where all possible combinations (permutations) of all ontology particulars $\mathbf{O}_P$ and `c` have to be computed for all (non-bare) ontological universals in $\mathbf{O}_U$, i.e. $O(|\mathbf{O}_U|*|\mathbf{O}_P|!)$. Yet, this worst-case is virtually impossible to occur in actual EOS systems, as this would require a domain model where all conceptual relations are of the form

C := (n, (EXISTENT, EXISTENT, …, EXISTENT)),

implying that n is the number of particulars $|\mathbf{O}_P|$, the semantics of these universals allow for arbitrary permutations of all particulars for a minting of their components, and there actually are **GENERATE** laws that produce all component permutations. In real applications, the complexity of generation sequences is certainly considerably smaller, and depends foremost on the number of **GENERATE** laws that are defined on universals that possess large numbers of particulars. As already mentioned, a predominant intention for modeling **GENERATE** laws is to control the materialization of domain facts within the system. It is thus up to the domain modeler to decide on this matter. Concept Theory simply provides the means to do so.

3.2.3.3.3. Generation of Universals under the Closed World Paradigm

The closed world paradigm encompasses the automated generation of new **QUERY** concepts from another occurrence of **QUERY** that has been handed over to the knowledge generation procedure by the knowledge retrieval procedure for semantic query rewriting (cf. Section 3.2.3.4.1 on semantic query rewriting). Handling **QUERY** concepts is similar and less complex than producing particulars for several reasons:

- Within each generation step, exactly one **QUERY** concept is created. This assumption that only single **QUERY** concepts and not sets thereof are being created, poses no restriction in generality, which is due to the fact that any number of coherent queries can also be formulated in a single (more complex) **QUERY** concept (assuming that the definition of generation steps that result in sets of mutually conflicting **QUERY** concepts can be shut out because this would be in itself a sign of an incorrect domain model).

- On a similar line of thought, **GENERATE** laws for **QUERY** concepts can be expected to be sensibly modeled in that the set of all **GENERATE** laws pertaining to one and the same **QUERY** concept will not produce conflicting new concepts. In particular, this implies that
  - the order of execution of the respective **GENERATE** laws is insignificant to the resulting concept, and
  - since the generated **QUERY** concepts are coherent, no other **GENERATE** laws besides the initially determined set will become applicable after any generation step, i.e. this set of laws defines the complete generation sequence.

  If the domain model includes semantically incorrect sets of **GENERATE** laws that do not possess these properties, the generation procedure will still be able to perform semantic query rewriting. Yet, the final **QUERY** concept which is the result of subsequently applying the specific set of **GENERATE** laws may be different for different execution orders. However, as already pointed out, this is a semantical modeling mismatch, and cannot be prevented by the syntactical definitions of Concept Theory that pertain to single concepts alone[40].
- Newly generated **QUERY** concepts will not be integrated into the ontology, as they are only an intermediate byproduct of the knowledge retrieval procedure. Each generation step thus simply substitutes the current **QUERY** concept for another. The result of a sequence of such concept generation steps is therefore a single, final **QUERY** occurrence (that in the end will be processed by the knowledge retrieval procedure), not a set of concepts.

QUERY concepts contain other universals as components, which necessitates an examination of syntactical constraints and how to avoid looped generation sequences, analogously to the discussion of Section 3.2.3.3.2.

a) Determinateness

Determinateness plays no role in semantic query rewriting, as **QUERY** concepts are containing universals. Consequently, a right component minting **GENERATE:**G[2] of a law **GENERATE:**G will comprise universals which do not necessarily have to be determined by the minting of **GENERATE:**G[1]. Theoretically, **GENERATE** laws for **QUERY** concepts do not need to be subject to any identity condition at all, namely in cases where the original **QUERY** concept is substituted by another on behalf of some property (i.e. components), regardless of its other properties.

b) Self-containment

The previously defined self-containment constraint applies to all concepts, regardless of whether they are particulars or universals.

c) Concept Nesting

Concept nesting, too, is independent of the ontological status of the minting concepts. Thus, the according constraint as defined for particular generation also applies here.

d) Structurally Identical Copies

[40] It is of course possible to test the complete set of **GENERATE** laws of an ontology by an according algorithm. In this way, the semantic domain model can be cross-checked syntactically by examining whether the ontology complies with the constraint that different sequential execution of coherent **GENERATE** laws (i.e. applicable to the same **QUERY** concept) will yield different results.

The existence of numerically distinct but structurally identical universals is not possible, i.e. the definition of a metaphysical universal within an ontology is unique. Hence, no structural copy can define a new existent but refers to one and the same metaphysical universal.

e) Structural Additions

Structural additions are illegal, notwithstanding whether the concept is a particular or a universal. The respective constraint, therefore, applies here as well.

f) Concept Generation Chains

As already mentioned, the chained execution of **GENERATE** laws for queries takes on a special form where only a single new **QUERY** instance is created in each generation step, while it is impossible that structural copies are numerically distinct. Therefore, the knowledge generation procedure only has to test whether a newly generated **QUERY** has already been part of the current generation sequence in order to avoid loops during execution. The system ontology need not be tested against these concepts, as they are only created temporarily during query processing, i.e. checking for ontology duplicates is not necessary because the new **QUERY** concepts are not being integrated into the ontology. The loop constraint for **QUERY** universals is thus:

> Let S_i be a sequence of i generation steps s_j:=(L_j,R_j) representing the execution of a respective **GENERATE** occurrence **GENERATE:**G:=(2,(L,R)) with **QUERY** concept R_j generated from **QUERY** concept L_j according to **GENERATE:**G, i.e. L_j and R_j are the specific concept mintings of L and R at s_j (after the execution of s_j). Let $\{S_j\}$, $j \le i$ denote the set of all distinct concept mintings within the sequence $s_1, s_2, \ldots, s_j \in S_i$. Let **O** be an ontology. It follows:
>
> $\forall s_j, s_{j+1} \in S_i,\ 1 \le j < i,\ \forall C:=(n_C, L_C) \in R_{j+1}: \nexists P:=(n_P, L_P) \in \{S_j\}: n_C = n_P \wedge L_C = L_P,$

i.e. no **QUERY** concept may be produced that has already been part of the generation chain. The according algorithmic solution for managing universal generation under the closed world paradigm is presented in Figure 3.26 and completes the definition of the knowledge acquisition procedure that has been introduced in Figure 3.12.

```
// Concept c: a new QUERY concept
// ConceptVector cset: a concept vector containing c
// Boolean continue: determines whether execution continues

while( continue==true ) do
{
    continue := false;
    ConceptVector gset := all ontology GENERATE laws pertaining to c;

    Concept current := c;

    forall g in gset do
    {
       // mark current concept:
       mark(current);

       // execute next generation step:
       current := generate(current,g);

       if( isMarked(current)==true )
       {
          // the current concept has already been part of the
          // generation sequence, i.e. a loop occurred;
          // proposed solution:
```

```
         // stop execution and return the original concept c:
         return c;
      }
      else
      {
         // current is a legally created new QUERY concept,
         // mark it as already produced:
         mark(current);
      }
   }
   // return the final result of the semantic query rewriting process:
   return current;
}
```

Figure 3.26: The Knowledge Generation Procedure for ***QUERY*** *concepts*

The code listed in Figure 3.26 implements the knowledge generation procedure for **QUERY** concepts as motivated in the preceding discussion. Note that the **`while`** loop, unlike with the algorithm for particular generation, actually has no effect on the execution, as the Boolean variable `continue` is set to `false` and not changed afterwards. It has only been included here for keeping the parallel structure of the two parts of the complete knowledge generation procedure, including both, particular and universal generation under the closed world paradigm. Thus, the termination of the algorithm depends solely on the respective constraint that no **QUERY** concepts may occur twice within a generation sequence (defined by the finite set of **GENERATE** laws `gset`) because this would mean a loop. In order to avoid such a loop, all concepts of the generation sequence are marked and the constraint is being tested by checking this marking at each generation step against the newly created **QUERY** concept. The execution complexity is therefore linear, i.e. O(n), with respect to the number of **GENERATE** laws that are being applied.

3.2.3.4 Knowledge Retrieval and the Law QUERY

Knowledge Retrieval encompasses the epistemological processes necessary for querying the knowledge held within an ontology, comprising domain universals and particulars. Laws controlling knowledge retrieval are occurrences of the concept **QUERY**.

Definition 3.10: The Epistemological Concept QUERY

The law **QUERY** is defined as the concept

$$\textbf{QUERY} := (\,3,\,(\,\text{EXISTENT},\,\text{EXISTENT},\,\text{EXISTENT}\,)\,) \in \Phi_U$$

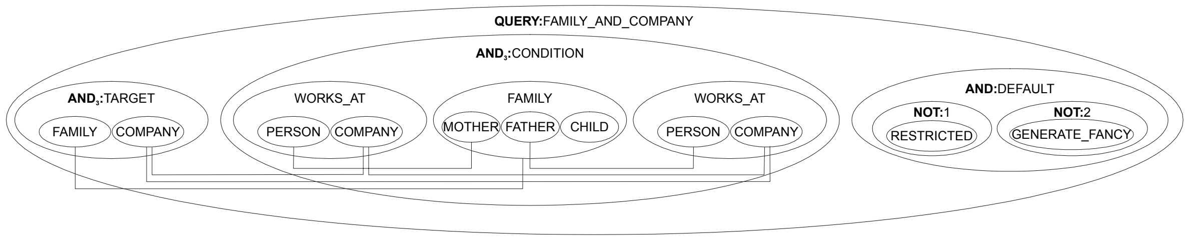
where

- **QUERY**[1] defines the target(s) of the query, ontology concepts whose occurrences make up the result set of the query. Thus, **QUERY**[1] is an n_0-ary **AND** condition of such non-conditional concepts.
- **QUERY**[2] is a condition on **QUERY**[1] that restricts the result set to target occurrences that comply with this condition.
- **QUERY**[3] is a condition on the ontology itself and defines a specific view on the ontology. Without loss of generality this condition can be constructed as a conjunction of **NOT** occurrences, i.e. the condition determines which ontology concepts are *excluded* from this view. Excluded are all non-conditional concepts of **QUERY**[3] and their occurrences.

Occurrences of **QUERY** may be either produced by a query interface for human interaction or passed to the system by other applications. The system then processes this query and returns the results. As mentioned in Section 3.1.3.4, different kinds of return values are possible: sets of concepts, numerical and Boolean values. Yet, on a system internal level it suffices to solely consider sets of concepts as numerical and Boolean values concerning these sets can be computed on their basis. Thus, the definition of **QUERY** need not include any component determining a specific kind of return value. Per default, each query is answered by producing the respective set of concepts, while further result processing is left to other applications.

Example 3.9: The Law QUERY:FAMILY_AND_COMPANY

Let **O** be a formal ontology, let PERSON, MOTHER, FATHER, CHILD, COMPANY, FAMILY:=(3,(MOTHER,FATHER,CHILD)), and WORKS_AT:=(2,(PERSON, COMPANY)) be ∈**O**. The **QUERY** law that represents the query 'Return all families along with companies, where both, mother and father, work at the same company' is defined as



Next to the target concepts in **AND:**TARGET and the target condition **AND:**CONDITION, the law **QUERY:**FAMILY_AND_COMPANY also specifies an ontology view, as defined by **AND:**DEFAULT. Assuming that

- RESTRICTED is an ontology universal that signifies classified information that should not be accessible to all users of the system[41], and
- GENERATE_FANCY is a similar ontology universal that groups a set of **GENERATE** laws for semantic query rewriting,

the condition **AND:**DEFAULT describes that this query may not yield any classified concepts (occurrences of RESTRICTED) and excludes a specific set of **GENERATE** laws for application during query processing.

Note the difference in semantics of the two conditions of **QUERY:**FAMILY_AND_COMPANY. The target condition **QUERY:**FAMILY_AND_COMPANY[2] refers solely to the concepts that are searched for, while the view condition **QUERY:**FAMILY_AND_COMPANY[3] addresses the ontology as such. The decisive difference of the two query conditions, however, lies in the way the query is processed. The target condition is tested on occurrences of the target concept in order to sort out all valid query values, i.e. this condition is an integral part of all iterative steps when traversing the ontology (view) graph. On the other hand, the view condition should be seen as part of the query *preprocessing*. As already discussed in Section 3.2.3.3.3, **GENERATE** laws can also be defined on **QUERY** concepts in order to enable semantic query rewriting (see also Sections 3.2.3.3.3 and 3.2.3.4.1 in this chapter). Obviously, this rewriting is performed *prior* to processing the final query – and the view condition influences this process by defining which **GENERATE** laws are applicable in the context of this query and which are not. The advantage of this way of processing queries is that it allows for *personalized*

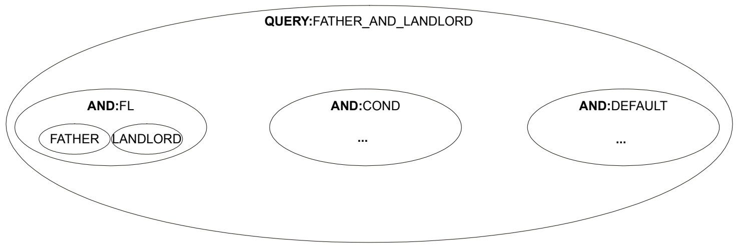
[41] Marking an ontology concept C as 'restricted' is, then done by an according **ISA** occurrence **ISA**:=(2,(C,RESTRICTED)).

knowledge retrieval. Thus, the view condition **AND:**DEFAULT of **QUERY:**FAMILY_AND_COMPANY should be interpreted as an instruction to process the query in the specific way defined for *default users*, while another view condition **AND:**PRIVILEDGED_USER may yield different query results with identical target concept and target condition definitions. Depending on the users (or user groups) included in the ontology, and on the granularity in which they are referring to ontological concepts, an arbitrarily detailed level of personalization can be achieved. Naturally, the view condition for different users will be generated automatically by the query interface (with respect to user profiles), while the target concepts and their condition can be individually defined by the users of the EOS system.

As Example 3.9 implies, the target condition **QUERY**[2] of a **QUERY** concept can be used to formulate arbitrary complex statements about the target concepts in **QUERY**[1]. Specifically, the target condition may be useful for defining intersections and unions between target concepts. This is feasible whenever the target concepts have common occurrences, e.g. the universals FATHER and LANDLORD within an ontology will most probably have a common subset of concepts that contains *fathers* that also are *landlords*. According queries are discussed in Example 3.10.

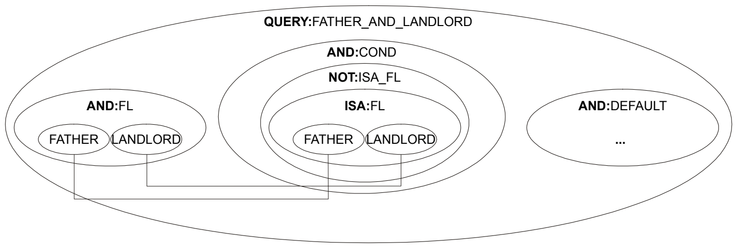
Example 3.10: The Law QUERY:FATHER_AND_LANDLORD

Let **O** be a formal ontology, let

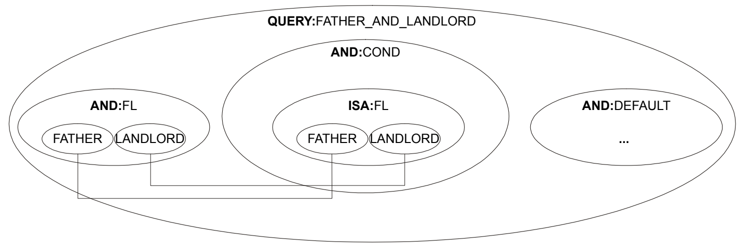


be a query against **O** that yields all occurrences of the concepts FATHER and LANDLORD. The target condition **AND:**COND contains statements that restrict the result set of the query. Specifically, it may comprise restrictions that pertain to the target concepts alone, thus defining different intersections:

- *Return occurrences of FATHER and LANDLORD, omitting concepts that are both.*
  An according condition is **NOT:**ISA_FL:=(1,(**ISA:**FL)), with **ISA:**FL:=(2,(FATHER, LANDLORD)), where the components of **ISA:**FL are related through identity conditions with their partner target concepts:



- *Return only occurrences of FATHER and LANDLORD that are both.*
  A possible condition here is simply **ISA:**FL:=(2,(FATHER,LANDLORD)), where the components of **ISA:**FL are again related through identity conditions with their partner target concepts:



The cases where one of the two target concepts is completely shut out from the result set, e.g. by using **NOT:**FATHER:=(1,(FATHER)), as well as the case where neither occurrences of FATHER nor of LANDLORD are searched for at all, are obsolete, as the this would contradict the specification of target concepts.

Technically, in comparison to relational database systems, **QUERY** concepts are similar to SQL queries. However, the semantics of **QUERY** concepts cannot be simply translated into SQL as the relational algebra knows no taxonomic structure but only relations. Concepts, on the other hand, next to defining *relations*, must also be viewed as parts of a *hierarchical* domain model. For example, the **QUERY:**FAMILY_AND_COMPANY of Example 3.9 could be paraphrased using SQL syntax:

SELECT *
FROM FAMILY, COMPANY
WHERE **AND:**CONDITION AND **AND:**DEFAULT

where the impact of **AND:**DEFAULT in terms of semantic query rewriting, of course, cannot be accounted for. This statement regards FAMILY and COMPANY as relations that are being joined in order to produce the desired result. Concept components, here, are interpreted as the attributes of relational algebra. Note that this analogy does not hold without restraints, as in Concept Theory, occurrences of a concept may very well possess more components, which exceeds the notion of relations in relational algebra. Depending on their context, concepts are either relations, or objects that are subject to other relations, i.e. attributes. As the preceding discussion has shown, this is a very useful property for modeling ontological, as well as epistemological knowledge.

Another way of translating **QUERY:**FAMILY_AND_COMPANY into SQL syntax would be:

SELECT FAMILY, COMPANY
FROM *ontology (view)*
WHERE **AND:**CONDITION AND **AND:**DEFAULT

where the target concepts are seen as attributes of the ontology (view), interpreted as a single relation encompassing all concepts. Again, the exact semantics of nested components within concepts are not satisfactorily mirrored in this approach. The reason for this lies in the different semantic models of Concept Theory and the relational algebra. Relational algebra draws a clear line between schema and facts, while this distinction is 'soft' in Concept Theory. Although there is an acknowledged difference between ontology universals and particulars, it is very well possible to define a universal that contains particulars as components. For example, in 3.2.2.4 we introduced a universal MARY'S_FAMILY:=(3,(Mary, FATHER, CHILD)). Relations do not recognize such a coupling between schema and data.

A direct analogy between concepts and relations according to the relational algebra can be built on a restricted level, though: lowest level universals may be interpreted directly as relations on their instances. Each particular instance, then is a tuple of this relation. The hierarchical domain model on top of these two lowest ontology graph levels, then, is strictly speaking beyond the relational model. This is also the case with most epistemological concepts. While rules in Concept Theory may be interpreted as functional dependencies or constraints in relational systems, there is no equivalent to laws that model the system behavior.

3.2.3.4.1. Semantic Query Rewriting

During semantic query rewriting, a query that is handed over to an EOS system is being transformed into another, similar query with amended semantics. As already stated in the previous section, query rewriting does not affect the view condition of a **QUERY** concept that stays the same throughout all generation steps (i.e. **GENERATE** laws that do change the third component **QUERY**[3] of a query are semantically incorrect). What semantic query rewriting affects, are thus the target concepts and the respective condition as determined in **QUERY**[1] and **QUERY**[2].

- *Modifying target concepts.*

  Possible modifications concerning target concepts are concept additions and deletions. For example, it might be useful to define **GENERATE** laws that add concepts (semantically) related to the target concepts, as it is done with online shopping or information services: customers that are looking for e.g. books of a certain author are also presented with books falling in the same category, or with audio and video products that are associated with this author. On the other hand, it may also be feasible to replace or simply remove target concepts, if it can be assumed that these concepts will yield a huge number of (actually unwanted) hits that might blur the query result. This is seen analogously to the common practice of search services to ignore 'stop words' like 'and', 'the', etc. that are part of virtually all Web documents. In the context of ontologies, one could e.g. avoid to search for a concept whose specializations are also part of the target set. As a simple example, if a target set consists of FATHER, LANDLORD and PERSON, it may be wise to omit PERSON completely as the target condition will probably also be valid for occurrences of PERSON that are neither FATHER or LANDLORD, yet it can be assumed from this target set, that only their occurrences are searched for.

- *Modifying target conditions.*

  The target condition can also be altered, for reducing or expanding the number of hits, or ameliorating the quality of the return set. Typically, the aim will be to render queries more restrictive for keeping the result set manageable.

The extent of semantic query rewriting might differ from system to system. Especially EOS systems where personalization and user profiles play an important role, queries will naturally be modified more often. In these cases, **GENERATE** laws will not only be defined with respect to target concepts and conditions but also depending on the view condition that defines the (user) perspective taken for the respective query.

3.2.3.4.2. Processing EOS Queries

The knowledge retrieval procedure `query()` that is processing **QUERY** concepts is demonstrated in Figure 3.27. Supporting methods not already discussed in the previous sections are:

- `createView(`*`ConceptVector`*` o, `*`Concept`*` c)`:

  computes an ontology view from an ontology `o` on the basis of condition `c`. The knowledge retrieval procedure uses this view as its private 'ontology'. Note that it is not necessary to materialize this view as the view condition can also be tested during program execution whenever the target condition is tested. Yet, assuming a materialized ontology simplifies the code of `query()`, which is the only reason why we chose to use it here.

Return Value: `ConceptVector`.

- `getConcepts(Concept t, Concept cond, ConceptVector v)`:

  computes the set of concepts in ontology (view) `v` that are occurrences of `t` complying with condition `cond` and returns this set (including t itself).

  Return Value: `ConceptVector`.

- `IsElement(Concept c, ConceptVector v)`:

  tests whether concept `c` is part of the concept vector `v`.

  Return Value: `Boolean`.

- `satisfiesCondition(Concept t, ConceptVector cond, ConceptVector v)`:

  tests whether concept `t` satisfies condition `cond` within an ontology (view) `v`.

  Return Value: `Boolean`.

```
ConceptVector o;

query( Concept q )
// Concept q: instance of QUERY
{
    Concept view_condition := q[3];

    // create ontology view according to view_condition
    ConceptVector view := createView(o,view_condition);

    // perform semantic query rewriting:
    Concept query := manageGeneration(q),
            target := query[1],
            condition := query[2];

    ConceptVector target_set :=();
    forall t in target[1], …, target[n] do
    {
      target_set := push(t, target_set);
    }

    forall t in target_set do
    {
      result_set := push(getConcepts(t,condition,view),result_set);
    }

    return result_set;
}

getConcepts( Concept t, Concept cond, ConceptVector v )
// Concept t: a target concept of a query
// Concept cond: the condition on t
// ConceptVector v: the ontology (view) for the query
{
    if( isElement(t,v)==false )
    {
      // t is not an element of view v;
      // reject t (and its occurrences):
      return;
    }
    else
    {
      // test cond on t:
      if( satisfiesCondition(t,cond,v)==false )
      {
        // t does not comply with cond;
        // reject t (and its occurrences):
        return;
      }
      else
      {
        // t is part of ontology view v and complies with condition cond;
        // return t and test its child concepts:
        ConceptVector result_vector := push(result_vector,t);
        forall s in getCommonSpecializaions(t);
        {
          result_vector := push(result_vector,getConcepts(s,cond,v));
        }
        return result_vector;
      }
    }
}
```

Figure 3.27: The Knowledge Retrieval Procedure

The algorithm shown in Figure 3.27 determines the set of all valid occurrences of the (universal) target concepts by traversing their respective concept graphs all the way down to their particulars, testing the target condition of the query in the process for filtering out occurrences that do not satisfy this condition. As all concept graphs are subgraphs of the ontology (view) graph, and the ontology is free of loops, so are the concept graphs. Thus, the algorithm terminates after the all particulars of all target concepts have been determined. The maximum complexity, therefore, is a complete linear traversal of the whole ontology, i.e. O(n).

These remarks round up the discussion of knowledge retrieval within EOS systems and, on a more general level, concludes the treatment of the realization of epistemological processes as defined by the EOS framework that we have laid out at the beginning of this chapter (in Section 3.1.3). This completes the epistemological model of the EOS approach and its integration into the formal framework of Concept Theory.

Chapter 4 EOS Systems

This chapter will present central aspects concerning the practical realization of an actual EOS system whose domain knowledge and behavior is based on an EOS ontology. For this purpose we will motivate document management on the World Wide Web as a possible EOS application scenario. In this context, an EOS system must perform knowledge extraction on Web pages, a procedure that may take advantage of document markup as found e.g. in XML documents. Based on this discussion, specific knowledge acquisition, generation and retrieval procedures will be exemplified. We will also motivate our preferred representation techniques for EOS ontologies, which encompasses an EOS markup language as well as a translation of EOS ontologies into a relational database schema. Remarks on our prototype implementation of an EOS system that operates within the application scenario will round up this chapter.

4.1 Application Scenario

In Chapter 1 the notion of ontologies was introduced by stressing the role of metadata in digital libraries (DL) and, as one prominent application area of metadata in this context, how bibliographical records are being used for classifying library stocks. Ontologies in this context can be employed in a variety of ways. Apparently, ontologies may be used to model different cataloging schemas (such as MAB and MARC), e.g. for classifying new records according to these schemas. On the other hand, an ontology may function as a general model for bibliographic data that includes and harmonizes different cataloging schemas. This would enable a DL system to manage records of incompatible schemas, e.g. in order to translate records from one format to another, or to search, compare and organize bibliographical data originating from heterogeneous record sets. As the discussion of Chapter 1 has shown, cataloging formats exhibit very strict category schemas that have been defined in specific international standards, e.g. MARC. Yet, in information processing, it cannot be generally presumed that data is always complying to such well-known standards. In fact, the common situation seems to be that a knowledge processing system must be able to handle a huge variety of different data formats where single records (or documents) may not be easily related to an immediately determinable schema that would allow for a correct semantical interpretation of the information content and context of the data. Here, ontologies covering domain knowledge, i.e. formally defined concepts relating to the intellectual content, may pose a means to enable machine-processing tasks such as classifying, organizing and indexing this data based on its semantics, not on syntactical features alone.

An interesting application area for ontology-driven document management is the World Wide Web and its information environment. The Web can be regarded as a vast source of information that consists of a huge set of (partially interlinked) electronic documents of different formats (e.g. HTML, XML, PDF, MSWord, etc.) covering practically any topic of human interest. The sheer size and heterogeneity of the Web poses a serious problem as to how its intellectual content can be successfully managed and put at disposal for human users. This concerns e.g. managing linguistic diversity (relating documents written in different languages and coping with synonyms, homonyms, etc. within the same language), automated document classification, and providing enhanced search services, to name but a few challenges in this area. In order to provide a basis for more sophisticated Web services of this kind, the Web is currently starting to move from its first generation to the second generation, the *Semantic Web* [5]. Tim Berners-Lee characterizes this new Semantic Web as one that should supply much more automated services based on machine-processable semantics of data and heuristics that make use of these metadata [6]. In this context, ontologies providing explicit shared domain models are regarded as key assets as they supply the basic vocabulary and coinciding semantics that are necessary to effectively access and communicate the information stored on the Web. In a nutshell, ontologies may provide the basis for linking related Web documents based on their *content*, regardless of the physical topology of the Web (as defined by URIs and Web links). And this, exactly, is what the Semantic Web is supposed to offer – a *semantic* linking of human knowledge, complemented with machine-processability.

The next sections will discuss key aspects that must be considered if the transition of the World Wide Web into a Semantic Web using ontologies shall be successful. As a practical example we suppose an example EOS system, i.e. a knowledge processing system complying to the architecture as described in Section 3.1.3. This EOS system is used for classifying Web documents according to their intellectual content, which results into a semantic linking as it is demanded for the Semantic Web. For this purpose the EOS system uses an according EOS ontology representing domain knowledge that is used for identifying concepts within Web documents. Based on this ontology, the documents are classified according to an interpretation of their content, a process that relies on a suitable mapping between document data and ontology concepts. Section 4.1.1 analyzes different levels of structure found in Web documents, which leads to a detailed discussion on document markup within semi-structured data in Section 4.1.2. Section 4.1.3, then, examines different approaches for utilizing document markup in Web documents in order to support information extraction, which will finally lead to our suggested technique for coupling markup tags and EOS ontology concepts.

4.1.1 Data on the Web

Considering the topology and information content on the Web, an EOS system managing Web documents is confronted with a vast and very heterogeneous information environment. Even within the same domain, documents typically exhibit very different properties concerning their format and linguistic ambiguities of their textual content. Regarding these properties, we can differentiate between several levels of data organization and their impact on knowledge extraction:

- *Unstructured data*. Plain texts in natural language, as well as binary data formats for images, video or audio, offer no explicit structure in form of a schema that could be used to syntactically interpret them. As such they consist of schemaless, or unstructured, data. This means that with unstructured data contextual information to help interpreting the data is missing. The context, and therefore assumptions about the meaning of the data, thus has to be determined (or conclusively guessed) by other means. Next to pattern detection in images, the central research effort concerns natural language processing of plain texts. The aim, here, is to find and link relevant information in natural-language documents while ignoring extraneous and irrelevant information. In order to achieve this goal, these techniques use context and string patterns within linguistic analysis that produce a valid interpretation of unstructured documents. Known obstacles in this area are linguistic ambiguities, such as homonyms and synonyms that are usually resolvable for humans but pose great difficulties in automatic processing. As a consequence, determining the semantical context of the intellectual content of the data in automated processing, which could provide a reliable basis for correctly interpreting the data, remains an error-prone procedure.

- *Semi-structured data*. The most prominent examples of semi-structured data formats are those used in the Web environment, such as HTML and XML. The specialty of semi-structured documents is that they can be described by schemata (DTDs in case of XML), yet they are not fully determined by them. For example, pieces of data (as expected by a schema) may be missing, and on the other hand, documents can contain extra information like annotations. Moreover, there is no strict typing in semi-structured data, and the structure of such documents is usually partial, i.e. they can contain elements which are not accessible through the schema e.g. graphics, or plain text that are possibly holding very detailed, even schema-related information which, nevertheless, is not mirrored in a schema compliant way. The most common approach to extract information from semi-structured data, in particular Web documents, is through software agents called wrappers [23]. Wrappers parse source documents in order to provide a basis for mapping source data into a structured, or another semi-structured format with native semantics. In other words, the task of wrappers is to guess the schemata of the documents they are parsing, which yields a concise interpretation of this data.
- *Structured data*. Fully structured information, such as data stored in relational database systems, complies to a predefined schema. This schema is first determined (during a modeling phase) and later populated with domain facts. Thus, all data follows, without exception, the structural pattern of the schema. If the schema is publicly available and the data accessible, it can be queried by standard query languages such as SQL. Yet, this is usually not the case, as data is usually only held in a structured way within a information system, while it provides this information to the outside world in a semi-structured format (e.g. in the form of Web documents) or even unstructured format (e.g. as a PDF text file).

Structure is an important means for making information machine-processable. While structured data offers best exploitability in terms of correctly interpreting data, e.g. according to a relational database schema, the vast majority of publicly available data consists of semi-structured data, i.e. Web pages containing document markup. Document markup describes the logical structure of Web pages, thus providing metadata on the respective documents along with the actual information content. Recently, research activities have been concerned with the development of higher standard techniques for providing a more sophisticated access to the information found on the Web [1]. This encompasses e.g. better search services, comprehensive Web directories, multi-lingual querying and browsing, etc. Document markup has become an acknowledged enabling technique for realizing such services. In order to utilize document markup effectively and thus provide a means for putting the vision of the Semantic Web into effect, we propose the employment of EOS ontologies that are offering a practical way of interpreting semi-structured data containing document markup according to the ontological and epistemological knowledge that is supplied by these ontologies. Underlining the points made here, in this chapter we will discuss document markup from two different perspectives. On the one hand, document markup can be used to add semantic metadata to Web documents, and on the other hand, EOS ontologies themselves may be expressed using semi-structured data.

4.1.2 Semi-structured Data and Document Markup

Document markup is used to provide metadata for electronic documents as found in public or restricted networks. Syntactically, markup elements often take the form of *tags*, reserved words put between pointy brackets e.g. tags like <title>, <table>, or <form>. A *markup language*, then, is a specification of a particular set of markup tags, how they may be combined, along with a fixation of their meaning. Text can be marked by enclosing it with an opening tag (<…>) and an according closing tag (</…>), e.g.

<title>Der aus dem Grab der Vergessenheit wieder erstandene Simplicissimus</title>

marks the words between the two tags as a (book) title. Markup elements can be nested, which produces hierarchical structures such as

<title>…<b>…</b> …<b> .. </b> … </title>,

where <title> tags are enclosing several occurrences of <b> tags. Stripped from its textual content, the logical makeup of any such document is laid out by the tree structure of its markup tags. As parts of nested structures the elements of markup languages can be used to indicate document content of the same kind (e.g. several text passages that are enclosed in <b> tags) and to group document content into more complex units (e.g. all text and markup information within <title> tags). In this sense, documents containing markup are structured. Nevertheless, such documents are commonly regarded as *semi-structured* data, because their markup must not necessarily instantiate a specific schema as is the case with structured data such as relations in RDBS. For example, markup elements may be omitted or repeated arbitrarily within a document, and usually they are not strictly typed.

The most wide-spread and also well-known markup language on the Web is HTML (HyperText Markup Language). It provides rendering information about the content of Web pages, e.g. text between the tags <b> and </b> is marked to be displayed in bold face by Web browsers. However, HTML tags are not suited for describing the information content itself, e.g. the HTML specification does not contain any <author> tags that could be used for marking the names of authors occurring in the text. Moreover, HTML offers no mechanisms for defining new markup elements. This must be done using a *meta-language* that, in turn, can be used for defining markup languages such as HTML. The most prominent such meta-language is the eXtensible Markup Language (XML). XML has become the universal format for structured documents and data on the Web[42]. Using XML, it is possible to define the syntax and structure of a markup language by specifying a corresponding Document Type Definition (DTD) that outlines a schema for a class of documents. DTDs consist of a structured enumeration of XML elements and their attributes, which defines markup tags, and, in its entirety, the complete syntax of a markup language. Although, historically, the original HTML specification emerged before XML, it is technically possible to define HTML using an XML DTD. Another example XML DTD is given in Section 4.3.3. The following sections will present how specific markup languages have been used in the past in order to introduce metadata to Web documents.

4.1.3 Coupling of Formal Ontologies and Document Markup

There are two main approaches to combine ontologies and markup languages, firstly, by defining new markup which is directly tied to a predefined ontology or, secondly, by translating

[42] Originally, XML was developed as a subset of SGML, providing the benefits of SGML concerning markup definition, yet stripped of some of SGML's features for a simplified, lean markup language for the Web. In this respect, XML has been designed for ease of implementation and for interoperability with both SGML and HTML. The base specifications of XML, as provided by the World Wide Web Consortium (W3C), are XML 1.0, W3C Recommendation Feb '98, and Namespaces, Jan '99. The XML Activity Statement explains the W3C's work on this topic in more detail.

foreign markup into native concepts of the local ontology. Related research projects can be found in the area of information extraction from Web documents, e.g. SHOE [53] and Ontobroker [30], [33]. The most common technique for automated information extraction from Web documents is using software agents called wrappers that are able to interpret the content of Web pages [23], [54]. Simple wrappers may only be able to parse a small class of Web documents complying to a specific structure or a set of keywords they have been defined to identify (e.g. for extracting standardized information from a DL interface). More advanced approaches are trying to make use of (semantic) document markup whose correct interpretation relies on a semantic model, e.g. supplied by an ontology. Then, successful information extraction depends, on the on the one hand, on the scope of the ontology and, on the other hand, on the ability of the software agent to map markup tags into ontology concepts. The following discussion will detail on some important approaches in that field of research, focusing on the manner in which they are employing document markup and ontologies[43]. Starting from simple HTML annotations, we will subsequently take a closer look at two ontology-driven approaches, SHOE and HTML^A.

4.1.3.1 HTML

HTML, is used to indicate the structure and layout of Web documents, i.e. their syntactical features. Accordingly, HTML consists of a fixed set of tags that, in the first place, allow for a syntactical document markup. Yet, some of these tags have also been used to transport semantic annotations. Historically, the first attempts at representing semantic aspects within Web documents relied on HTML META tags [7]. Although their expressiveness is limited, META tags can be used for stating global document properties that apply to an entire Web document, such as Dublin Core [22] information about author, date of creation, etc.:

```
<META NAME="author" CONTENT="Barry Smith">
<META NAME="date" CONTENT="2001">
```

With the HTML 4.0 specification [62] two new markup elements have been added that are candidates for semantic annotations: SPAN and (the practically equivalent) DIV. The SPAN element is defined as a generic container of any text element offering a generic mechanism for adding structure to documents. In this sense it has been designed in the first place for specifying layout, i.e. simple generic stylesheet mechanisms. Nevertheless, the HTML 4.0 specification expressly suggests the use of the SPAN element to express the semantic structure of a document, e.g.:

```
<SPAN CLASS="author">Barry Smith</SPAN>
<SPAN CLASS="date">2001</SPAN>
```

Yet, despite its possible uses, SPAN has hardly been used at all in Web documents. HTML META tags, on the other hand, found some limited application, e.g. in assisting Web crawlers with identifying annotations on Web pages.

4.1.3.2 SHOE

SHOE is an acronym for <u>S</u>imple <u>H</u>TML <u>O</u>ntology <u>E</u>xtension. As the name implies, SHOE is used to specify machine-readable extensions, i.e. annotations that can be incorporated into HTML files. The aim is to provide semantic content that can be parsed (and understood) by Web agents (like crawlers). The assumption here is, of course, that the annotations used in Web pages are known to the agent, e.g. through a corresponding ontology the agent is aware

[43] A related question is how suitable a specific ontology is for interpreting a set of Web pages (following some markup scheme). Resolving this question is beyond the scope of this thesis. Instead, we refer to an heuristic approach for evaluating the suitability of an ontology in].

of. In this way, agents are able to directly gather meaningful information, metadata, about the Web documents they are parsing. This is used to facilitate knowledge acquisition procedures and search mechanisms, as document semantics are presented in a format that allows agents to interpret it correctly without any need for syntactical or linguistic analysis.

The semantics of Web annotations are predefined in a formal ontology that can be parsed by a Web agent. SHOE ontologies encompass taxonomic classifications, i.e. specialization hierarchies of classes. Additionally, it is possible to define relationships and inferences in the form of horn clauses (where SHOE does not allow disjunction or negation). Other SHOE ontologies or previous versions of an ontology can be addressed from within a current ontology. Employing SHOE is a two-phase process (cf. Example 4.1 and Example 4.2): first, a SHOE ontology is defined describing a valid classification of objects and their relations, and second, HTML pages are annotated according to this ontology. These Web annotations refer to arbitrary data entities within an HTML page and can be used to categorize them according to an explicitly declared SHOE ontology. Data entities can also be related to each other by using relationships defined within such an ontology.

Example 4.1: A Simple SHOE Ontology

```
<HTML>
  <HEAD>
    <!-- Document is conformant with SHOE 1.0: -->
    <META HTTP-EQUIV="SHOE" CONTENT="VERSION=1.0">
    <TITLE> A Simple Bibliographic Ontology </TITLE>
  </HEAD>
  <BODY>
    <!-- The current ontology's name and version: -->
    <ONTOLOGY ID="biblio-ontology" VERSION="1.0">
      <!-- SHOE's "base-ontology": -->
      <USE-ONTOLOGY ID="base-ontology" VERSION="1.0" PREFIX="base"
              URL="http://www.cs.umd.edu/projects/plus/SHOE/base.html">

      <!-- Excerpt from the ontology's classification hierarchy: -->
      <DEF-CATEGORY NAME="Person" ISA="base.SHOEEntity">
      <DEF-CATEGORY NAME="Author" ISA="Person">
      <DEF-CATEGORY NAME="Date" ISA="base.SHOEEntity">

      <!- - A relationship declaration: - ->
      <DEF-RELATION NAME="Publication">
          <DEF-ARG POS="1" TYPE="Author">
          <DEF-ARG POS="2" TYPE="Date">
      </DEF-RELATION>
    </ONTOLOGY>
  </BODY>
</HTML>
```

Example 4.1 indicates that entity Person subcategorizes from “`base.SHOEEntity`”, the category “`SHOEEntity`” as defined in the fundamental SHOE ontology “`base-ontology`”. “`SHOEEntity`” is the uniform root category for all categories declared in any SHOE ontology. With the `biblio-ontology` defined shown in this example, its categories and relations can be used within the same or other HTML files in order to annotate them. A portion of such a Web page is presented in Example 4.2. Note that the information on this document has, strictly speaking, been duplicated because the author Barry Smith is mentioned as such in both, SHOE markup and the text passage. Generally, SHOE markup is purely additional to any actual Web page content (in natural language) intended for human users.

Example 4.2: Employing a SHOE Ontology

```
<HTML>
…
  <BODY>
    <!-- A unique identifier (e.g. the URL of the Web page itself): -->
    <INSTANCE KEY="http://any.unique.url/">
      <!-- Reference to the location of the "society-ontology": -->
      <USE-ONTOLOGY ID="biblio-ontology" VERSION="1.0" PREFIX="biblio"
                    URL="http://unique.url.of/biblio-ontology.html">

      <!-- A relationship instance: -->
      <RELATION NAME="biblio.Publication">
          <ARG POS="1" VALUE="Barry Smith">
          <ARG POS="2" VALUE="2001">
      </RELATION>

    </INSTANCE>
…
Barry Smith is the Author of this book which was published in 2001. His
treatment of…
</BODY>
…
</HTML>
```

In terms of the EOS approach, SHOE annotations could in principle be used for knowledge acquisition. Concepts may be expressed through annotations and interpreted directly as such. In this way, the KEP could ignore all other Web page content and concentrate solely on SHOE markup, which has to comply with an according EOS ontology (e.g. expressed in SHOE syntax). An obvious drawback of SHOE, as indicated above, is that the information of Web documents has to be duplicated, i.e. SHOE annotations are pure additions to Web pages, encapsulated within `INSTANCE` tags. This means that despite the fact that a specific information may already be stated within the text passages of a document, information providers must observe that it still has to be formulated a second time using SHOE syntax.

4.1.3.3 HTML^A

HTML^A is an annotation language used in the project Ontobroker [33]. As the name implies it represents an extension to the common HTML syntax, namely an extra `ONTO` attribute for HTML anchors. Similar to SHOE, an ontology consisting of classes, attributes and relations must be defined in Frame Logic [49] for representing some domain of interest. With Ontobroker, this ontology is declared globally for a community that may use ontology classes in their Web pages using HTML^A syntax, e.g.

```
<A ONTO="http://www.smithsite.com/Barry/myPub.html :Publication">
<A ONTO="http://www.smithsite.com/Barry/myPub.html [Author='Barry Smith']">
<A ONTO="http://www.smithsite.com/Barry/myPub.html [Date='2001']">
```

specifies an instance of class Publication whose Author attribute is set to the value `Barry Smith` and whose Date is defined as `2001`. As indicated in this example, each class instance is required to be defined using a unique identifier, here realized through the URI "`http://www.smithsite.com/Barry/mayPub.html`". An excerpt of an Ontobroker ontology describing class `Publication` and its attributes could be:

```
Object[ ].
Publication::Object.
Publication[
  Author =>> STRING;
  Date =>> STRING].
```

While an Ontobroker ontology is held separately, complying HTML^A annotations are integrated into the Web pages themselves. Restricting annotations to HTML anchors, though, poses a certain drawback, as these tags cannot be nested. As a consequence, in the above example the global identifier "`http://www.smithsite.com/Barry/myPublication.html`" must be used in all value assignments (i.e. `[Author='Barry Smith']` and `[Date='2001']`) for attributes of the `Publication`. As with SHOE, Ontobroker relies on a specific syntax in order to be able to identify and parse document markup correctly.

4.1.3.4 Document Markup in the EOS Approach

The previous examples for realizing semantic annotations on Web pages have shown different forms of document markup and its relation to domain knowledge. While the HTML 4.0 specification does not mention any model for specifying the semantics of META, SPAN and DIV instances, SHOE and HTML^A annotations refer to particular ontologies that allow for a correct interpretation of the markup elements they define. Yet, the ontologies are used differently in these approaches. SHOE ontologies are embedded in Web pages, whereas HTML^A annotations used in Ontobroker rely on a centrally stored (within the Ontobroker system) ontology – and are defined using a different representation language[44]. Both, SHOE and Ontobroker, provide their own syntax extensions to HTML and suffer from similar disadvantages:

- Neither SHOE nor HTML^A emerged to become part of any standard or industry initiative. Consequently, their application remains very limited.
- Web page content must be explicitly duplicated, i.e. all ontology-related information has to be expressed using specific SHOE or HTML^A syntax, regardless of the remaining document content, in particular further markup information (provided most commonly using XML tags).
- SHOE and Ontobroker do not make use of any other document markup besides their own HTML extensions.

Considering these drawbacks of previous ontology approaches, we promote no specific EOS syntax but propose to use XML markup for realizing semantic document annotations. XML has already become a widely accepted standard, including tool support. Moreover, since XML seems to become the new de-facto standard for Web documents, one can expect that a rapidly growing portion of information on the Web will contain XML markup. It is only reasonable to make use of this development.

EOS markup, essentially, consists of XML tags that are providing meta information on the textual content of Web pages. Similar to approaches like SHOE and Ontobroker, an EOS system relies on its own ontology. Apparently, a mapping between ontology concepts and document markup must be defined. This mapping, in its simplest variant, may consist of a 1–1 mapping between concepts and identically named tags, e.g. AUTHOR – <AUTHOR>. Yet, the employment of a mapping function also allows for more flexible 1–n relations between one concept and several tags, e.g. AUTHOR – <AUTHOR>, <Author>, <author>, <writer>, <auteur>, <Verfasser>, etc. As this example indicates, a mapping like this could be used to cope with different spellings, multilingual information environments, and may even allow for

[44] We will not go into detail concerning the specifics of the ontology representations used in SHOE and Ontobroker. We agree with [32] that both representations are essentially equal in expressiveness.

interpreting foreign documents that were not tagged with respect to a specific EOS ontology[45]. Thus, an XML document containing the following text passage could be directly interpreted by an EOS system (assuming there exists a suitable mapping to ontology concepts):

```
<Publication>
  <Source>http://www.smithsite.com/Barry/myPublication.html</Source>
  <Author>Barry Smith</Author>
  <Date>2001</Date>
</Publication>
```

The possibility of directly using Web page content reveals some decisive advantages. Firstly, there is no need for defining a distinct EOS markup language as any document markup may be the target for the EOS knowledge extraction procedure. Consequently, document information need not be duplicated, which allows information providers to keep their documents compact and easier to manage, relieving them from the burden to express document content in a foreign syntax. For these reasons, the EOS approach endorses to exploit document markup using an appropriately defined mapping function (addressing markup elements of any kind) and not to fixate the system on one specific syntax. For our discussion of the EOS application scenario we will assume that the information environment the exemplary EOS system is working in is (syntactically) restricted to a large set of XML documents (possibly with multilingual content and complying to a number of different DTDs). Thus, the mapping function uniformly consists of a set of XML tags and their corresponding ontology concepts. The EOS system uses this mapping to parse XML documents for identifiable information and subsequently classifies them according to its internal ontology.

4.2 Classifying Web Pages Using an Exemplary EOS System

The preceding discussion has shown how a mapping between EOS ontologies and document markup can be realized for accessing the semantic content of Web pages. In this section we will now turn to describing the functionality of the exemplary EOS system for managing Web documents. Naturally, examining how such an EOS system works, reflects EOS epistemology as introduced in Chapter 3:

- Classifying Web documents is one example for *knowledge acquisition* that involves on the one hand a knowledge extraction procedure (KEP) and on the other hand the knowledge acquisition procedure itself, represented by the law **ACQUIRE**. Making use of the according mapping, the KEP translates document information into concepts. In our example, these concepts are typically new particulars for universals as defined by the EOS ontology, e.g. an *author* or *title* previously unknown to the ontology (where AUTHOR and TITLE are ontology universals). These concepts are passed to the EOS system using the law **ACQUIRE**, i.e. for each new concept the KEP constructs an according **ACQUIRE** particular that is then processed by the EOS system. In addition to the concepts referring to document contents, the KEP also creates a new DOCUMENT particular for each Web document it parses. These, too, are passed to the EOS system that organizes them conforming to the internal ontology. This results in a document classification dictated by the ontology, e.g. a Web document containing information that has been identified as MP particulars could reasonably be categorized as a particular of the universal POLITICS_DOCUMENT. Knowledge acquisition is discussed in more detail in Section 4.2.2.

[45] Of course, a mapping to foreign tagging schemas can be error prone and may involve some tuning of the mapping function. Nevertheless, the possibility of making use of document markup of foreign documents offered on the Web is a promising perspective.

- Human users (or applications such as software agents) can access the system via an interface that allows for querying the EOS system about the documents it manages, a *knowledge retrieval* procedure driven by the law **QUERY**. The query interface takes a user request and transforms it into an equivalent **QUERY** particular which is then passed to the EOS system. Typical queries in this context would be e.g. "*Return all documents on topic X*", or "*Return all (particular) concepts of type Y satisfying condition Z in documents on topic X*". Knowledge retrieval is the topic of Section 4.2.3.
- In order to provide a better service to its users, the EOS system performs semantic query rewriting, which is an application area for *knowledge generation* using the law **GENERATE**. During query processing the EOS system is checking whether the internal ontology contains **GENERATE** particulars that can be applied to the current query. If this is the case, the original queries (represented by according **QUERY** particulars) are transformed into semantically different **QUERY** particulars. For example a query "*Find all documents on members of the political party CDU*" could be transformed into the rewritten query "Find all documents on members of the political parties CDU or CSU". Semantic query rewriting as an example of knowledge generation is presented in Section 4.2.4.

Beforehand, the basis for any epistemological process in this application scenario must be provided by a consorting EOS ontology. The outline of such an ontology that may be utilized for document classification is given in the following Section 4.2.1.

4.2.1 An EOS Ontology for Document Classification

This section will sketch the makeup of an EOS ontology used for document classification. For this purpose an ontology engineer must commit to a specific and explicit domain model. According to the scenario of Section 4.1 the EOS ontology is intended for classifying news articles and related documents published on the Web. This means that the ontology should include a concept describing a (news) document, along with a concept hierarchy that organizes a number of document categories (e.g. politics, economics, science, sports, etc.) and a portion of "world knowledge" that contains concepts relating to the topics addressed by the document categories (e.g. the concepts PARLIAMENT, MP or **PARTICULAR:**FISCHER[46] are candidates for identifying a POLITICS_DOC). All of these concepts belong to the purely ontological side of the domain model. Additionally, the EOS ontology will include EOS rules (e.g. for specifying constraints for document categories) and EOS laws (e.g. for personalized query rewriting). The outline for such an ontology is given in Figure 4.1.

[46] Mr. Joseph (Joschka) Fischer, member of the German Green Party (Bündnis90/Grüne) has been the German Federal Foreign Minister since October 1998.

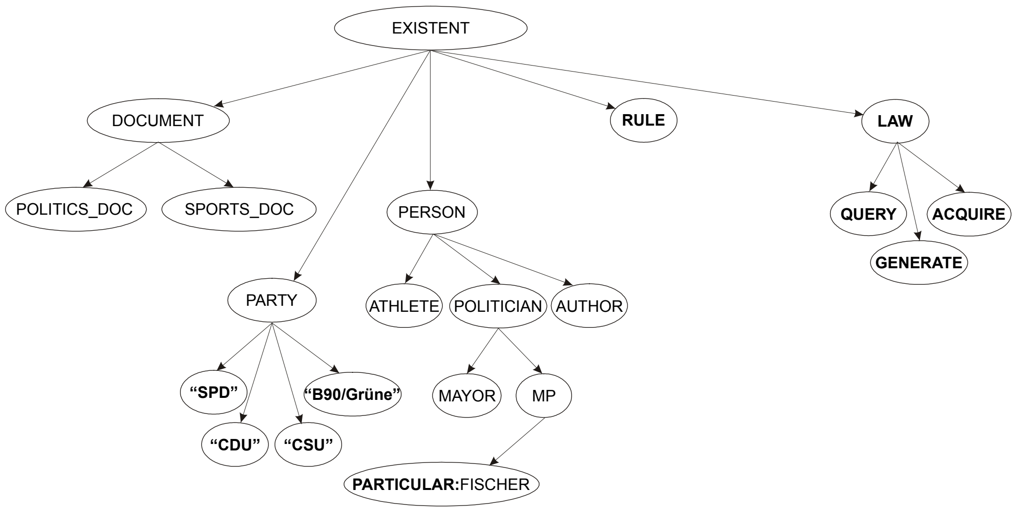


Figure 4.1: Outline of an EOS Ontology for Document Classification

For better readability, this ontology excerpt does not yet include specific EOS laws and shows only the overall concept hierarchy, omitting the internal component structure of these concepts. As we move on with our EOS application example we will go into more detail concerning the (ontological and epistemological) concepts involved if necessary. Here, we will only motivate the component structure of the concept DOCUMENT in order to point out the basic understanding of *documents* that we will follow in the classification scenario. Generally, a Web document may be characterized through different properties, e.g. its location, author or content. Thus, for modeling the concept DOCUMENT some components such as URI, AUTHOR and TOPIC suggest themselves. In fact, as our discussion of standardizing bibliographical data using cataloging formats in Section 1.1.3 has shown, there are very elaborate cataloging schemas that are suggesting several hundred bibliographic categories (all of which could be represented by components of DOCUMENT).

Yet, in our application scenario we are actually intending to stress a different point. The example EOS system is dealing with documents published on the Web, i.e. with a very heterogeneous environment where information is presented with very different degrees and quality of usable markup. The EOS application scenario does assume some degree of document markup, which allows to define a direct mapping between ontology concepts and markup tags. In order to keep the mapping function simple we assume that the Web documents in the application scenario are complying to XML syntax. Yet, we do not restrict the scenario to a set of documents following the same schema (which would apply for software tools like wrappers who are able to parse and interpret only specific classes of documents), or some limited set of schemas. Moreover, the information provided by markup tags is usually not a satisfactory description of a Web document in terms of giving a complete account of its bibliographical data and intellectual content. The example EOS system should be able to accept basically any Web document that provides some markup which can mapped to its internal ontology. Therefore, we should not assume any particular document structure at all, which leads to the notion of DOCUMENT as a set of (presumably correlated) concepts of arbitrary size.

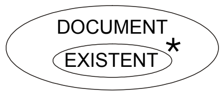


Figure 4.2: A General Definition of the Concept DOCUMENT

The conceptual definition of DOCUMENT in Figure 4.2 reflects this perspective. It is held in a most unrestricted form, determining neither the exact number of nor any category for DOCUMENT components that is more specific than EXISTENT, the root concept and therefore the most general concept of the ontology. This essentially defines DOCUMENT as a container for any type of information, which is the basic assumption underlying the discussion of knowledge processing procedures that, in the succeeding sections, will be used to describe the behavior of the example EOS system in more detail.

4.2.2 Knowledge Acquisition: Classification of Web Documents

Knowledge Acquisition in EOS is a two-phase process involving a knowledge extraction procedure (KEP) and the knowledge acquisition procedure (KAP) that accepts **ACQUIRE** concepts as an input. As already mentioned, the KEP must parse Web documents, identify markup tags and map these tags to ontology concepts. The EOS framework does not specifically demand this, but it is very well possible to formulate the mapping function used in this context using Concept Theory and make it part of the internal ontology of the EOS system, e.g. using the concept MAPS_TO as exemplified in Figure 4.3.

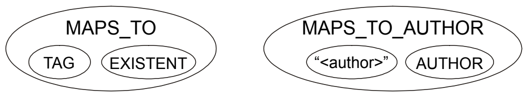


Figure 4.3: Mapping Markup Tags and Concepts

Clearly, the KEP must be aware of the semantics of these ontological concepts and interpret them as mapping between tags and concepts – something the EOS system itself, of course, is not able to do. According to Concept Theory, the EOS system only knows the semantics of a specific set of concepts, such as **ISA**, **RULE** and **LAW**. Purely ontological concepts can only be interpreted in terms of their position within the ontology graph and their ontological category, i.e. whether they are representing universals or particulars. Therefore, the relationship between the KEP and the EOS system is that of a software agent using the EOS system for gathering specific information, i.e. occurrences of the concept MAPS_TO, which the KEP (not the EOS system) understands as extension of the mapping function. The KEP may thus formulate queries like "*Return all tags that map to concept* AUTHOR". The representation of such a query in Concept Theory is shown in Figure 4.4. In this section we will not yet flesh out the manner in which queries are processed by the EOS system as our focus will remain with knowledge extraction and acquisition. Knowledge retrieval will be the topic of Section 4.2.3. For the current discussion it suffices that the KEP may use the EOS system in order to retrieve mapping information.

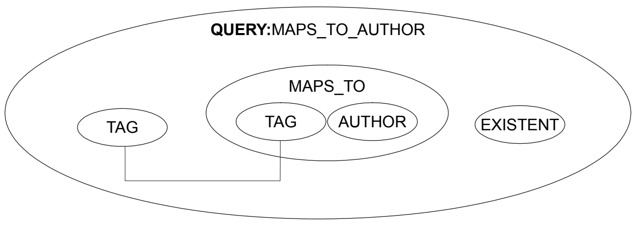


Figure 4.4: Example Query of the Knowledge Extraction Procedure

Using this mapping information, the KEP is able to detect concepts within tagged Web documents. A short excerpt of such a document is shown in Figure 4.5. The passages show two tagged strings ("Angela Merkel" and "CDU") that must be tested against the mapping function.

```
 …
 On Friday <MemberOfParliament>Angela Merkel</MemberOfParliament> announced
that…
 …
 <Party>CDU</Party>
 …
```

Figure 4.5: Sample Web Document

Provided that the KEP actually can find an appropriate mapping, e.g. because the internal ontology of the EOS system contains concepts as shown in Figure 4.6., it can now start to pass document information to the EOS system.

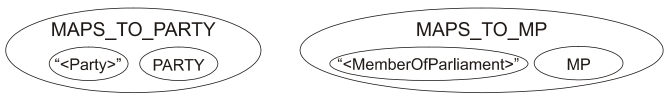


Figure 4.6: Mapping Concepts

According to the application scenario, when parsing a Web document the KEP will suggest new concepts to the EOS system. On the one hand this will cover particulars it found within the document, and on the other hand it will create a new DOCUMENT particular representing the Web page itself. Passing new concepts to the EOS system is done using the knowledge acquisition procedure that processes **ACQUIRE** concepts. Thus, the KEP must construct **ACQUIRE** occurrences according to the information it identified within a Web page. Applied to the sample Web document of Figure 4.5 this means e.g. that the KEP can assume that "Angela Merkel" is the value of a new **PARTICULAR:**ANGELA_MERKEL ("Angela Merkel") that is a specialization of the universal MP. An according **ACQUIRE** concept is shown in Figure 4.7. **ACQUIRE** concepts consist of two components. The first component is an **ISA** occurrence suggesting the category (MP) of a new concept (**PARTICULAR:**ANGELA_MERKEL) that is going to be passed to the EOS system. The second component is intended for providing a component mapping between the new concept and its suggested generalization. Assuming that MP is a bare concept not possessing any components, no component mapping has to be specified, as is indicated in Figure 4.7 by leaving the second component (EXISTENT) unspecialized.

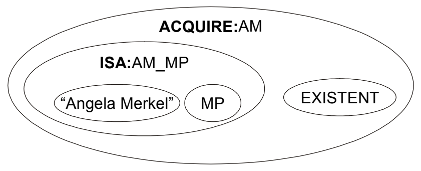


Figure 4.7: An **ACQUIRE** *concept for acquiring* **PARTICULAR:**ANGELA_MERKEL

The knowledge acquisition procedure accepts **ACQUIRE:**AM as an input and checks whether the suggested new concept (**PARTICULAR:**MERKEL) makes a valid concept according to its internal ontology. In this example, this is the case, semantically, if all conceptual rules of the ontology pertaining to the suggested parent (MP) are also true for the new concept, and, structurally, if **PARTICULAR:**MERKEL is a valid minting of MP[47]. For keeping this initial example simple, we are presuming that there are no rules on MP. During the knowledge acquisition process the EOS system also tries to find occurrences of the suggested parent, that are candidates for more specialized parent concepts for the new concept. If the ontology does contain such concepts, the EOS system will check their rules on the new concept, too. The overall goal of the knowledge procedure is to find the most specialized position within the ontology graph the new concept complies to. If the new concept is not valid according to the respective rules of a tested parent, it must be rejected as its specialization (and the new concept will become the child concept of the formerly tested parent). This means that the new concept must be rejected completely if it does not comply to the rules of the parent concept originally suggested in the **ACQUIRE** concept. The ontology presented in Figure 4.1 contains neither universal specializations of MP, neither any rules against the new **PARTICULAR:**MERKEL. Therefore, the EOS system will integrate it into the ontology as proposed by **ACQUIRE:**AM. The second new concept the KEP detected in the Web page, **PARTICULAR:**CDU ("CDU"), on the other hand, will be rejected as it is already part of the ontology[48].

Next to passing concepts representing newly found information to the EOS system, the KEP also creates conceptual representations of the Web pages it parses[49]. These DOCUMENT particulars are also handed to the EOS system using according **ACQUIRE** concepts and subsequently integrated into its internal ontology at the most specialized position the KAP can determine. In this way, the KAP classifies documents in accordance with the EOS ontology it

[47] For the remainder of this chapter we will not detail any further into structural aspects. Instead, we will focus on the role of EOS rules and other semantic facets of knowledge processing. Our examples are held in a way that structural differences between new concepts and their proposed parents are of minor importance.

[48] Note in this context that an ontology may very well contain several distinct particulars having the same value, i.e. a new particular is not necessarily rejected if another particular with the same value exists within the ontology. This is, for example, the case with homonyms. The term "kiwi" may depict a bird, a fruit, and (then spelled with a capital 'K') an Australian citizen. For each of these different meanings we can presume that the KEP will produce **ACQUIRE** concepts suggesting different parent concepts for the respective particulars bearing the same value, e.g. "kiwi" could once be suggested to fall into category ANIMAL, and then into category PLANT. This would result in two distinct ontology particulars (**PARTICULAR:**KIWI_BIRD and **PARTICULAR:**KIWI_FRUIT) that both carry the same value "kiwi". This actually shows one of the advantages of ontology-based knowledge processing: linguistic ambiguities can be resolved by respecting the semantic context of terms.

[49] Note that in the current application scenario only conceptual representations of Web documents (i.e. occurrences of DOCUMENT) are being integrated into the ontology, not the Web pages themselves. This is due to the general definition of the concept DOCUMENT as given above. A more detailed definition of DOCUMENT could, of course, contain components such as URI (for storing the Web location of the document) and CONTENT (for storing a file copy of the document itself). In any case, the EOS system will always and exclusively work with *concepts*, in our example DOCUMENT particulars (that may or may not contain the original Web page content).

is working on. For example, it is desirable that a Web document such as the one depicted in Figure 4.5 will become an occurrence of POLITICS_DOC. What is needed to successfully achieve this classification are ontology rules indicating that a Web page exhibiting some set of properties (identified through its markup information) falls into a specific category. One such rule is shown in Figure 4.8 (left). **RULE:**POLDOC expresses that any occurrence of POLITICS_DOC must possess a POLITICIAN concept (or a specialization thereof) as a component. The second rule of Figure 4.8, **RULE:**SPORTSDOC pertains to occurrences of SPORTS_DOC and states that sports documents are expected to mention either an *athlete* or a *sport*. As indicated in this example, the right-hand side of **RULE** concepts may consist of a simple or composite condition (of arbitrary complexity).

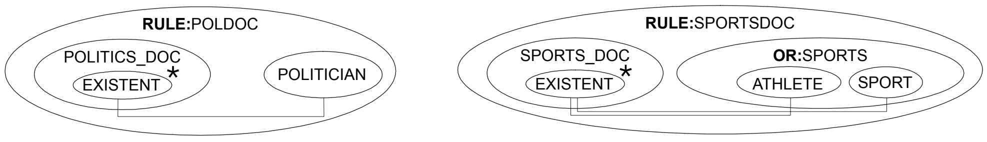


Figure 4.8: Simple rules for the concepts POLITICS_DOC and SPORTS_DOC

Generally, the KEP will not provide a complete classification but only suggest a new DOCUMENT particular[50], in our example **PARTICULAR:**AMDOC as depicted in Figure 4.9. Structurally, **PARTICULAR:**AMDOC contains all concepts the KEP could identify within the Web page. This conceptual information about the document content becomes the basis for the classification process.

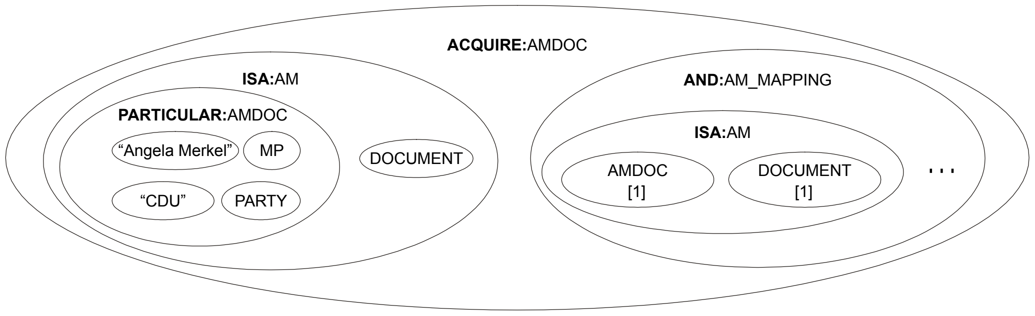


*Figure 4.9: An **ACQUIRE** concept for acquiring **PARTICULAR:**AMDOC*

Whenever the KAP processes a candidate for DOCUMENT it will traverse the ontology graph along the specialization hierarchy, testing all rules that are defined for the respective concepts on the new particular (and rejecting it for a parent whose rules it violates). This process ends at the lowest universal level within the ontology graph where concepts may possess only specializations that are particulars. In some cases, the KAP may find several parent universals, which may be due to an incomplete or possibly poorly designed ontology (i.e. the ontology does not provide a domain model that is specific enough to successfully classify new concepts), or due to insufficient input from the KEP (i.e. the KEP was not able to determine

[50] Suggesting that a new particular falls into some category (such as DOCUMENT) of course already is a classification of some sort, yet a preliminary one. Evidently, the KEP and the KAP approach concepts from different perspectives. The KEP operates in some specific application context (in our example it parses Web pages) and may use this context (e.g. it is aware that the files it processes are documents and that text enclosed within markup tags translates to concept values) for proposing the ontological kind of a new concept. The KAP, on the other hand, uses the ontology as its context. Depending on the ontology graph structure it may refine a classification suggested by the KEP.

enough useful information, which could stem from a poor document markup or an incomplete tag–concept mapping). There are different ways of dealing with such a situation. The EOS system could e.g. extend its ontology by creating a new universal that specializes all parents detected by the KAP and will inherit all their child particulars, plus the new particular. On the other hand, the EOS system might not alter the ontology as such but either duplicate the new particular and assign a copy[51] to each of the possible parent concepts, or (arbitrarily) sort out one single concept among them as the only parent. The current KAP definition in Section 3.2.3.2 suggest this latter approach (as it presupposes a complete ontology model), yet a different behavior in this respect may very well be arguable. The decision mainly depends on a trade off between storage space (creating additional concepts causes costs here) and access time (the more information is materialized within the ontology the faster query return values can be computed). In any case, the KAP will classify incoming DOCUMENT concepts according to the internal EOS ontology. This information, organized in the form of an ontology graph, may subsequently be queried using the knowledge retrieval procedure.

4.2.3 Knowledge Retrieval: Querying Information on Web Documents

Knowledge Retrieval in an EOS system is entirely based on the notion of concepts and their organization into an ontology graph. The EOS system accepts **QUERY** concepts that will be processed by the knowledge retrieval procedure (KRP). **QUERY** concepts own three components specifying the query target (a set of concepts the query is aiming at), the query condition (on the target concepts), and a view condition (on the ontology itself). In this section we will mainly concentrate on the usage of the first two components of **QUERY** concepts. How ontology views may be used in EOS queries will be outlined later in the context of semantic query rewriting (see Section 4.2.4). As a return value the KRP computes a set of ontology concepts that are representing the outcome of the query, including all (particular and universal) occurrences of the target concept that are satisfying the query condition within the query view. Human users may want to use a query interface that translates their information needs into **QUERY** concepts which are then passed to the EOS system. In terms of the EOS system, the query interface is a software agent calling the in-built KRP that will answer the query. The concepts yielded by the KRP can then be used by the query interface to produce a result representation intended for human users.

An example query that refers to the document classification ontology is presented in Figure 4.10. The semantics of **QUERY**:CDU_DOCS are such that it asks for occurrences of DOCUMENT (query target) mentioning members of parliament (MP) that are members of the political party CDU (query condition) without any restriction on the ontology (view condition is the root concept EXISTENT). Thus, **QUERY**:CDU_DOCS may be translated to "*Return all documents (i.e. document representations) reporting on CDU members*".

[51] Note that "copying" the particular would not result in an ontology where one and the same particular occupies several ontology graph leaves but in an integration of several distinct particulars (possessing different labels) of identical component structure. This would be analogous to the customary employment of several reference cards (e.g. one for an author index and another for a title or subject index) for the same book in library card catalogues.

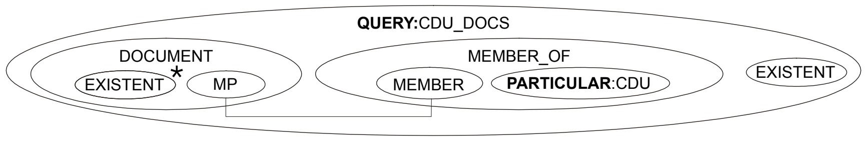


Figure 4.10: Querying the Document Classification Ontology

In the general form of the KRP we introduced in Section 3.1.3.4, the result set for this query contains the query target DOCUMENT and all its particular and universal occurrences. In our document classification scenario, this would minimally yield concepts DOCUMENT, POLITICS_DOC and the newly acquired **PARTICULAR:**AMDOC (cf. Figure 4.11). Recall that the KRP traverses the ontology graph in a top-down manner, checking whether the query condition is violated. If this is not the case, the respective concept (be it universal or particular) will be added to the result set. This may very well include universal specializations of DOCUMENT like SPORTS_DOC (unless there was a **RULE** concept within the ontology ruling out that members of the CDU party can appear in occurrences of SPORTS_DOC), and also SPORTS_DOC particulars (e.g. if the respective news document reported that a CDU party member participated in a marathon, or that some sport event was sponsored by the party).

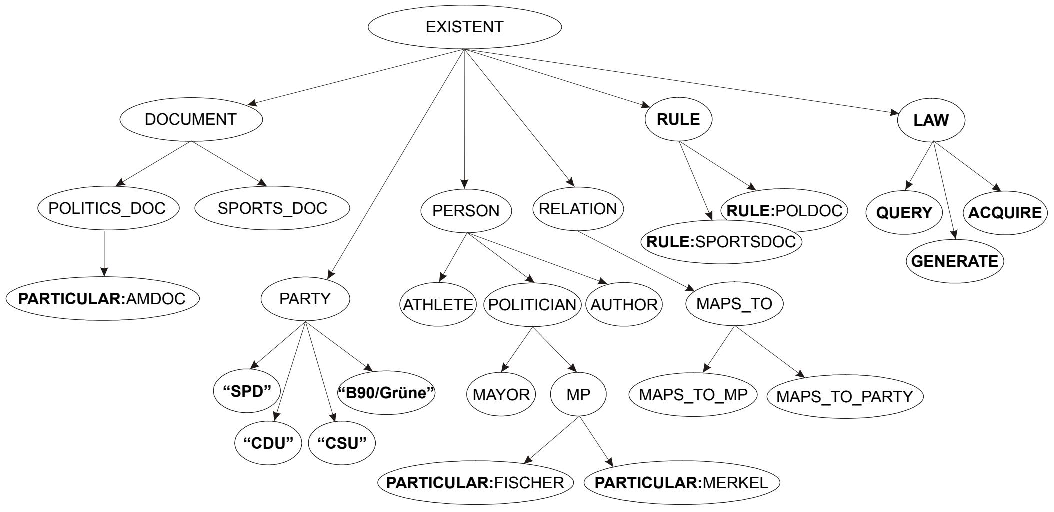


Figure 4.11: Extended Version of the Document Classification Ontology

It is a preliminary decision whether the query result set should include all **ISA** occurrences needed to construct the ontology subgraphs the query is addressing, i.e. if the return value also includes the arcs of the respective subgraphs. We suggest that the query interface is demanding this for supplementary processing (e.g. for generating a browsable graphical display of the query result in the form of concept hierarchies). Therefore, it may augment the target concept set in order to obtain the required information. A generic solution to this problem involves adding **ISA** occurrences to the target set, along with according **IDENTITY** concepts. The basic idea is to couple each target concept with an extra **ISA** concept whose second component (that generalizes the first component) is identified with the target concept (through an **IDENTITY** concept). Figure 4.12 shows this for **QUERY:**CDU_DOCS (where components new to Figure 4.10 are shaded grey in order to highlight the additions made).

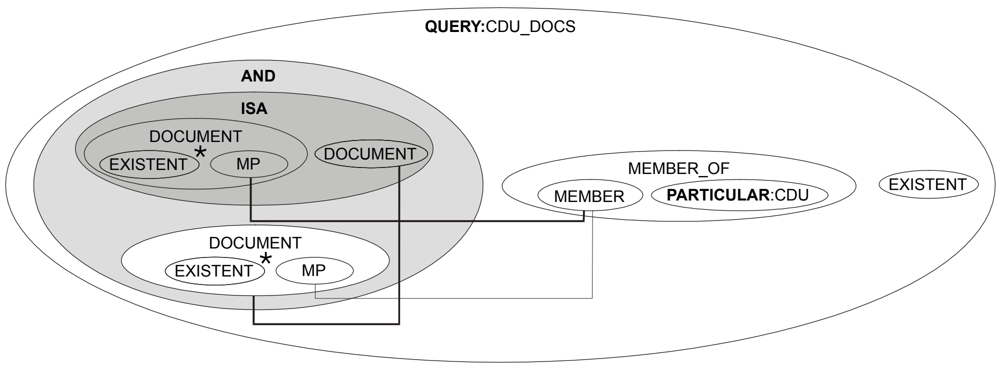


Figure 4.12: Extended Target Set for ***QUERY:****CDU_DOCS*

This extension to the original target set is semantically equal to the statement "*Next to computing occurrences of the original concepts, return also all* ***ISA*** *occurrences that depict specializations of these concepts*". Practically, this would suffice in order to supply the query interface with the complete subgraph information to the original query. In fact, it would even produce too many **ISA** occurrences, namely those who mention specializations of target concepts that were eventually rejected from the result set because they violated the query condition. Thus, in order to omit returning redundant information of this kind, all conditions pertaining to the original target concepts must also be tested for the first component of the respective **ISA** concept. In Figure 4.12 this is indicated through the identity conditions equating MP and MEMBER for both, the DOCUMENT *and* the **ISA** concept. Hence, the query condition is now also being tested within the **ISA** target concept, and unnecessary **ISA** occurrences are going to be filtered out from the result set.

As already mentioned, the query interface could change any user query in this manner and thus obtain the complete subgraph information for the query result. This example was constructed from the external perspective of a software application (here, the query interface) that uses the services of the EOS system in a specific way. Taking an internal perspective, i.e. concerning the actual ontology the EOS system is using, we can identify another solution to achieve the same effect, namely through semantic query rewriting. If the EOS ontology contained an according **GENERATE** law, all queries against the EOS system could be modified to also return the complete subgraph information (see Section 4.2.4 for a more detailed discussion on this topic). Yet, before we turn to discussing semantic query rewriting in the application scenario, we will shortly stress some general aspects concerning the KRP.

A number of basic types of queries against an EOS system are conceivable (cf. the discussion in Section 3.1.3.4). EOS queries can be differentiated according to the return values they are expecting:

- Sets of concepts, e.g.: "*Return all documents reporting on CDU members*"
- Numerical values, e.g. "*Return the number of all documents reporting on CDU members*"
- Truth values, e.g. "*Return whether there are documents reporting on CDU members*"

As already mentioned, the general version of the KRP as presented in Section 3.2.3.4 only allows for queries expecting sets of concepts, which does not affect the applicability of an EOS system (as, firstly, numerical and truth values can be easily computed – e.g. by the query interface – from queries targeting sets of concepts, and, secondly, the KRP can well be extended to handle such queries). A distinction of another kind between EOS queries (on sets of concepts) concerns the type of concepts expected within the return set:

- Particulars, e.g. "*Return all particular persons (within the ontology)*", which addresses the information status of the EOS system – the more particulars an EOS system possesses in addition to the overall domain model, the more concrete information it has gathered, e.g. the number of particular documents categorized by the document classification ontology.
- Universals, e.g. "*Return all universal persons*", which is asking about the role and extension of specific concepts (here PERSON) within the specialization hierarchy of an EOS ontology. For example, a domain expert could be interested in finding out whether the ontology contains all relevant specializations of a domain concept (and possibly extend the ontology if necessary).
- Universals and particulars, e.g. "*Return all persons*", which encompasses both, domain particulars and their ontological relation to the query target concepts.

We have shown that the KRP in its basic form supports the latter, which leaves it to the query interface to filter out universals or particulars if the user query is demanding this. Again, only slight changes to the general KRP would be necessary to provide this functionality. The last classes of EOS queries we want to point out are constituted by the semantics of their conditions:

- Conditions on the target concepts alone, e.g. "*Return all sports documents that were published in 2002*". Most user queries will be of this type.
- Conditions referring to the hierarchical makeup of an EOS ontology, e.g. "*Return all concepts that have the same ontological status as sports documents (i.e. all sister concepts to sports documents according to the ontology)*". These are queries aiming at the structure of the ontology graph.

Queries against the structure of the domain model, represented by the ontology graph, will fall into three main categories, depending on their orientation in relation to the target concept(s):

- Downwards, concerning child concepts
- Upwards, concerning parent concepts
- Sideways, concerning sister concepts

The KRP, by definition, operates in a downward fashion, returning all occurrences, i.e. child concepts, of a target concept that are satisfying the query condition. In order to obtain parent or sister concepts, one must explicitly navigate through the ontology graph. A generic way to do so is depicted in the subsequent figures.

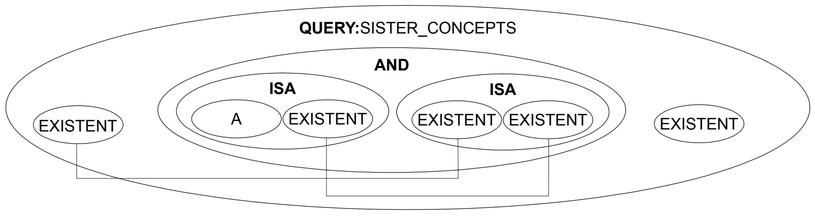


Figure 4.13: Determining Sister Concepts of Concept A

Figure 4.13 shows how a query on the sister concepts of some concept A can be formulated using according **ISA** and **IDENTITY** concepts in the query condition. Similarly, Figure 4.14 exemplifies how to determine parent concepts of A within the ontology graph. Note that **QUERY:**ORDER2_PARENT_CONCEPTS yields the second generation parents (i.e. the "grand parents") of A. The order (i.e. the path length that is being traversed within the ontology graph) depends on the number of **ISA** concepts that are tied together in the query condition. Immediate parent concepts may accordingly be found employing only one **ISA** concept, grand-grandparents using three **ISA** concepts, etc.

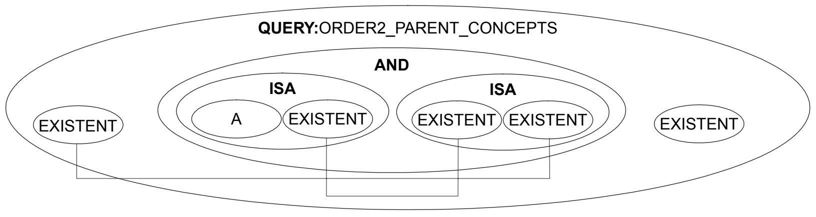


Figure 4.14: Determining Second Generation Parent Concepts of Concept A

This concludes the general discussion of knowledge retrieval using EOS system. The following section will now highlight the impact of EOS query rewriting, a technique which may be used to transform user queries according to epistemological guidelines (i.e. expressed through **GENERATE** laws that make part of the ontology).

4.2.4 Knowledge Generation: Semantic Query Rewriting for Querying Documents

In Section 3.2.3 knowledge generation in EOS systems was described in two different contexts, firstly for deducing new concepts from information stored in the ontology applied to newly acquired concepts (then triggered by the KAP), and secondly for semantic query rewriting (then invoked by the KRP). In any case, the knowledge generation procedure (KGP) uses **GENERATE** laws for producing new concepts. For our discussion of the document classification scenario we will concentrate on using EOS knowledge generation for semantic query rewriting (SQRW). The general idea behind SQRW in the EOS framework is that an EOS system should be able to transform user queries depending on the context provided by its internal ontology for providing a better service to the user. SQRW does not tackle physical query processing and its optimizations (this is the task of the underlying DBMS used by the EOS system) in terms of e.g. return times, but it is concerned with the *semantic* implications of queries. Hence, SQRW aims at optimizing the output an EOS system is producing for specific user queries. Depending on the ontology structure the focus of a user query can e.g. be enlarged or narrowed down by adding, deleting or replacing target, condition and/or view components of the original **QUERY** concept. The domain knowledge that advises such query transformations (presumably defined by a domain expert) is part of the internal ontology of the EOS system and represented by respective **GENERATE** concepts. During SQRW the KGP is checking whether there exists a **GENERATE** law that may be exerted on the original **QUERY** concept. If this is the case, the KGP computes the new **QUERY** concept, which then will replace the original one. Figure 4.15 shows an example of a semantically interesting query transformation that could reasonably be applied to **QUERY:**CDU_DOCS. The figure indicates that the original **QUERY:**CDU_DOCS is being transformed by applying **GENERATE:**CDU_IMPLIES_CSU to it. The transformation brings about that the semantics of **QUERY:**CDU_DOCS ("*Return all documents mentioning members of parliament that are members of the CDU party*") are extended to **QUERY:**CDU_OR_CSU_DOCS ("*Return all documents mentioning members of parliament that are members either of the CDU or of the CSU party*")[52].

[52] In Germany, within the parliamentary context (indicated through the target concept component MP – the query requires the target documents to provide information on members of parliament) it is reasonable not to differentiate between CDU and

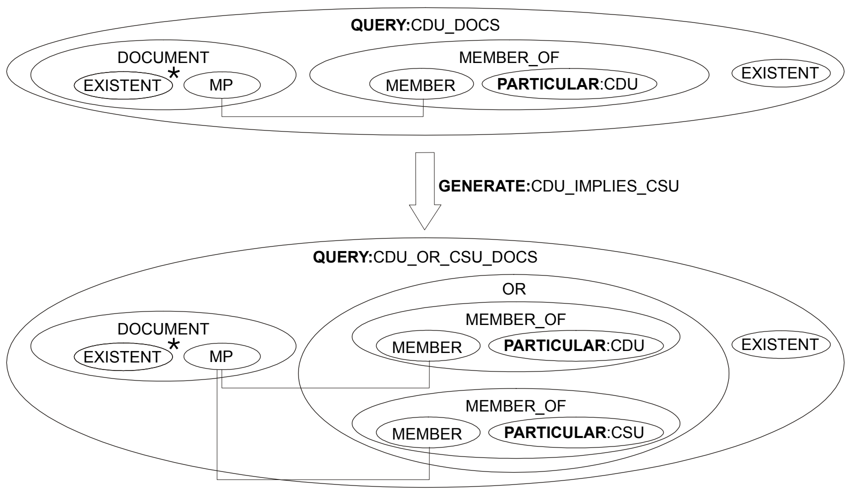


Figure 4.15: Semantic Query Rewriting on ***QUERY:**CDU_DOCS*

In order to achieve this change in semantics, the example query transformation presented in Figure 4.15 only needs to address the query condition of **QUERY:**CDU_DOCS. Other examples of SQRW may also involve modifying the query target and/or query view concepts. Figure 4.16 displays how the law **GENERATE:**CDU_IMPLIES_CSU may be formulated. The overall structure of **GENERATE:**CDU_IMPLIES_CSU shows a **QUERY** concept as its first component, i.e. it may only be applied to concepts of type **QUERY**, which implies that it will only be utilized during SQRW. **GENERATE** laws (or, concepts of Concept Theory in general) are built in a constructive way, based on structural patterns. The first component indicates the architecture some concept must follow for the **GENERATE** law to be applicable to it. **GENERATE:**CDU_IMPLIES_CSU, for example, expects a **QUERY** concept whose condition (its second component) contains e.g. a MEMBER_OF concept of the given form. The second component of a **GENERATE** law, then, describes the component structure of the desired outcome of the concept generation process it is representing. With laws designed for SQRW, the first and second component will mostly depict very similar **QUERY** concepts, as it is the case with **GENERATE:**CDU_IMPLIES_CSU (the design of its two **QUERY** concepts is identical, except for a disjunctively used new MEMBER_OF and an additional **IDENTITY** concept, both part of the query condition).

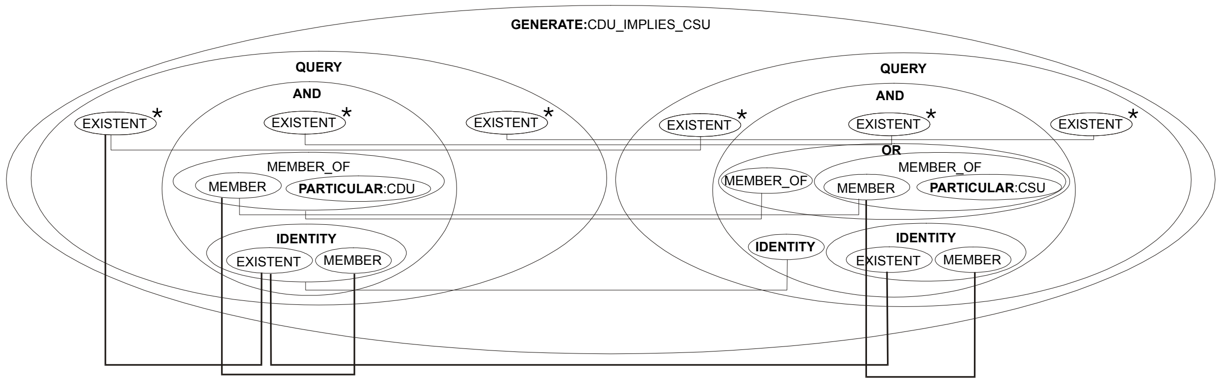


Figure 4.16: Concept Structure of ***GENERATE:**CDU_IMPLIES_CSU*

CSU members as they are forming one single parliamentary group. Only on a (federal) state level the distinction between

The task of the KGP, now, is to generate a new **QUERY** concept out of an actual **QUERY** concept it accepted from the KRP that, in our example, matches the structure of the first component of **GENERATE:**CDU_IMPLIES_CSU. The KGP does so by respecting the **IDENTITY** conditions between the components the **GENERATE** law (depicted through connecting lines between these components), i.e. the KGP takes the design plan of the second **QUERY** concept of **GENERATE:**CDU_IMPLIES_CSU and fills in particular and universal values (as provided by the input concept) according to the instructions given by the consorting **IDENTITY** conditions. As Figure 4.16 displays, query target and view concepts are being copied directly without any alterations. SQRW in this example concerns only the query condition. Figure 4.17 presents a detail view of this, leaving all concepts of **GENERATE:**CDU_IMPLIES_CSU aside that are not involved in the construction of the new query condition.

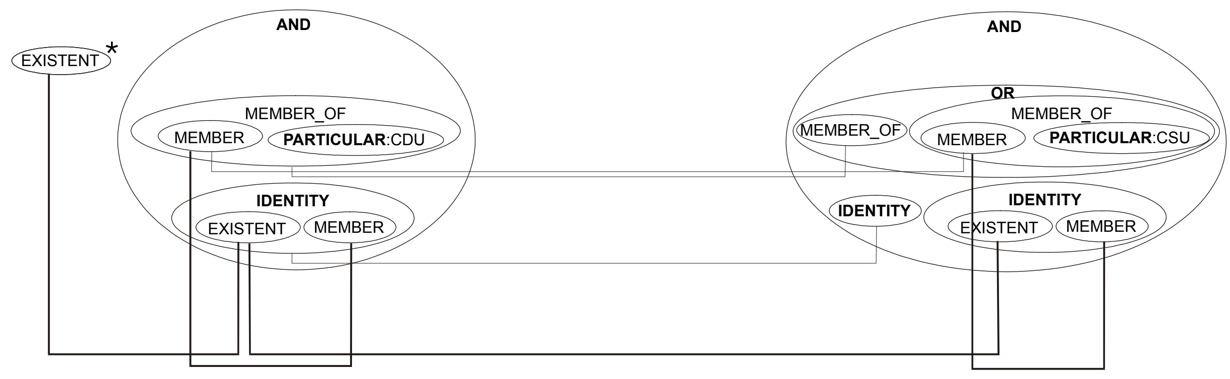


Figure 4.17: Detail View of ***GENERATE:****CDU_IMPLIES_CSU*

GENERATE:CDU_IMPLIES_CSU is intended to transform a query pertaining to CDU members into one that also asks for CSU members (cf. Figure 4.15). Therefore, some existing MEMBER_OF concept relating to **PARTICULAR:**CDU must be converted into an **OR** condition comprising two MEMBER_OF occurrences, one addressing **PARTICULAR:**CDU and one **PARTICULAR:**CSU. Straightforwardly, this is done in Figure 4.17, and **IDENTITY** conditions (the lines between the MEMBER_OF on the left-hand side and the first component of the **OR** concept on the right-hand side, and between the two MEMBER concepts) signify how the KGP should copy the concepts involved. Next to the MEMBER_OF information of the **QUERY** concept being processed, the accompanying **IDENTITY** conditions (of this **QUERY** concept) must be observed. Taking a look at **QUERY:**CDU_DOCS in Figure 4.15, one can see that there exists an **IDENTITY** condition between the MP component of DOCUMENT and MEMBER of MEMBER_OF. This **IDENTITY** condition within **QUERY:**CDU_DOCS is generically represented in **GENERATE:**CDU_IMPLIES_CSU using the **IDENTITY** concept that is explicitly stated inside the left-hand side query condition. This **IDENTITY** concept is identically copied to the right-hand side, where also an additional **IDENTITY** concept is created. This new concept is added in order to represent identities such as the one introduced in **QUERY:**CDU_OR_CSU_DOCS, the transformation result after the KGP applied **GENERATE:**CDU_IMPLIES_CSU on **QUERY:**CDU_DOCS. Thus, the structure of the **GENERATE** law ensures that the **IDENTITY** condition expressing "*members of parliament that are members of the CDU party*" in the original **QUERY:**CDU_DOCS will be turned into "*members of parliament that are members of the CDU or of the CSU party*" in **QUERY:**CDU_-OR_CSU_DOCS.

CDU and CSU becomes important.

Following the instructions of a **GENERATE** law on **QUERY** concepts, the KGP can thus realize SQRW for an EOS system. In this context, it should be noted that SQRW is a service to the user, defined by domain experts and offered within an EOS ontology in the form of **GENERATE** laws. Yet, SQRW is not a dictate but may be suppressed if a (human or software) user should choose to do so. This can be achieved using the view condition of **QUERY** concepts. View conditions determine which parts of an ontology are visible for knowledge processing (cf. 3.2.3.4). A generic example of an EOS query that prevents SQRW is presented in Figure 4.18. The view condition of **QUERY:**OMIT_SQRW excludes all **GENERATE** laws from being applied to this query, and hence prevents any SQRW that wholly depends on them. Generally, ontology views – especially in combination with **GENERATE** laws – can be used to manage the behavior of the KRP and, thus, are a means for realizing e.g. access restrictions and personalized knowledge retrieval.

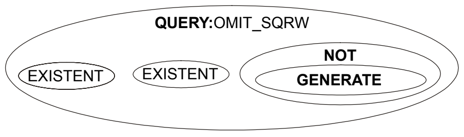


Figure 4.18: A Generic Query Suppressing Semantic Query Rewriting

The focus of **GENERATE** laws may be arbitrarily narrow or general, i.e. it is possible to define SQRW guidelines for a very specific set of queries (e.g. **GENERATE:**CDU_IMPLIES_CSU only refers to queries on documents that are satisfying a particular condition), on the other hand a **GENERATE** concept may pertain to **QUERY** concepts in general. Figure 4.19 shows an example for such a general SQRW definition. **GENERATE:**SUBGRAPH resumes the discussion on query return values of the previous section. The KRP, by default, returns a set of result concepts that are occurrences of the target concepts, yet without supplying **ISA** occurrences that determine the respective subgraph structures (cf. Figure 4.12 where it is shown how to extend **QUERY:**CDU_DOCS in order to obtain this information). Figure 4.19 demonstrates how **QUERY** concepts can be generically altered to return all according **ISA** occurrences by using **GENERATE:**SUBGRAPH in SQRW.

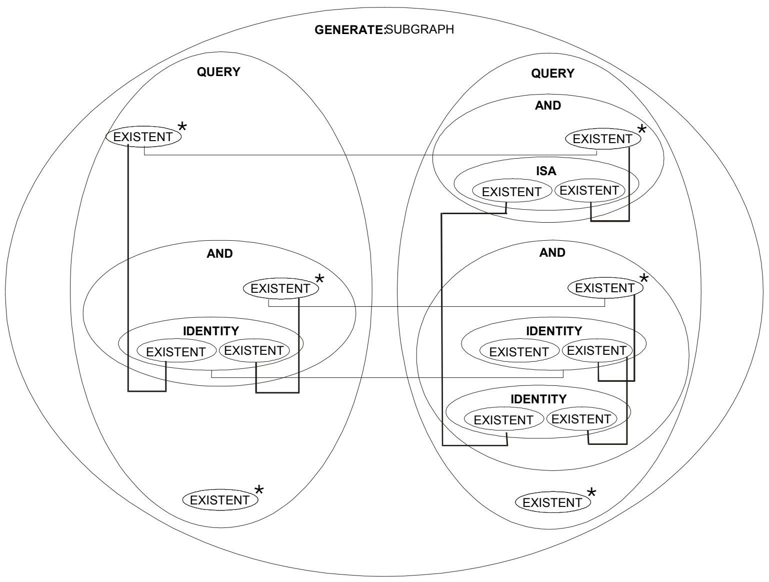


Figure 4.19: Adding Subgraph Information To Arbitrary Queries

GENERATE:SUBGRAPH determines that the target concept set is supplemented by **ISA** concepts and also defines according **IDENTITY** conditions. These are ensuring that the components of the newly added **ISA** concepts will be correctly minted. If **GENERATE:**SUBGRAPH is defined for an EOS ontology, all queries (as the first component of **GENERATE:**SUBGRAPH in its generic form applies to any **QUERY** concepts) will be changed to return the complete subgraph information.

4.3 Representing EOS Ontologies in XML

Ontology engineering requires predefined languages for representing ontologies. There are several possible approaches to define such languages. Following [7], current ontology representations fall into two main groups that are either Web-based (e.g. using RDF, XML, HTML), or based on logic rules (e.g. first-order logic or frame logic languages). Each approach promotes its own perspective on formalizing knowledge. Web-based ontology representations put emphasis on the 'static' domain model, i.e. the definition of domain concepts and their interrelations. Common application areas are specifications of catalogues or indexes for information retrieval, and thesauri in computational linguistics. Rule based languages stress the inference semantics characterizing a domain and therefore focus on the 'dynamic' domain model and how it may be used, e.g. by a knowledge based system. The EOS framework acknowledges static and dynamic aspects equally and combines them in a uniform model based on Concept Theory (cf. 3.1.3). The concept hierarchy spanned by an EOS ontology graph encompasses the hierarchy of all domain concepts. This includes the 'static' ontological and onto-epistemic concepts (such as BOOK, AUTHOR, TITLE and appropriate EOS rules and conditions), as well as 'dynamic' epistemological concepts (namely EOS laws) with distinct semantics as laid out by Concept Theory. The discussion of Concept Theory in Chapter 2 and Chapter 3 has shown that any object of knowledge (be it ontological, onto-epistemic or epistemological) within the EOS framework is expressed through the same construct, a concept. Thus, a language for defining EOS ontologies can be held very compact, which facilitates the process of ontology engineering. The following sections give an outline on how and on what grounds such a representation for EOS ontologies could be defined.

EOS Ontologies contain metadata on objects of a certain knowledge domain, e.g. a "Bibliography Ontology" could consist of all concepts denoting bibliographic entities (e.g. AUTHOR, TITLE, ITEM, WORK), their interrelations in the form of EOS rules (e.g. COPY_OF, SECONDARY_TO) and EOS laws determining how to apply them in knowledge processing tasks. Being metadata, from a knowledge processing point of view the concepts of an EOS ontology themselves are only a specific kind of data. They must be represented using a predefined terminology and can be created, altered, communicated to third parties (e.g. software agents), parsed and stored in electronic form. Thus, the same considerations that are valid for information representation in general are also pertaining to the representation of ontologies. Significantly, the most prominent examples for defining ontologies can be found in the area of Web languages and associated techniques. We will therefore discuss the key aspects of this field of research, taking the perspective of ontology engineering, which will then lead to our preferred representation technique for EOS ontologies.

4.3.1 RDF and RDFS

The Resource Description Framework (RDF) has gained some importance as a format for representing machine-processable semantics of on-line information resources (cf. [7]). It is actually an application of XML, introduced for standardizing the representation of metadata for Web pages. The data model behind RDF consists of three basic types: resources, properties and statements. A *resource* is an arbitrary entity that is addressable by a URI, e.g. a Web page or specific parts of it. *Properties* are attributes or relations (1- or n-ary properties) used to describe a resource. Finally, *statements* combine resources, properties and their values, i.e. a statement describes the value of a property for a particular resource. In order to specify ontologies, however, a fixed set of modeling primitives (classes, relations, etc.) is needed. Such a set is provided by the RDF Schema (RDFS) specification. However, these extensions have not proven to be sufficient for rendering RDF a suitable ontology representation language. Ontologies formulated in RDF are still limited in their expressiveness, e.g. it is not possible to derive properties of properties, or subproperties thereof (cf. [58]), there is no support for Boolean operators, etc. For this reason several higher-level ontology languages have been proposed, either on top of RDF (such as DAML+OIL) or based on (first-order) logic.

4.3.1.1 Logic-based Languages

Description logics (cf. [8], [3]), also known as terminological logics, form a class of logic-based knowledge representation languages. They originated in the area of semantic networks and define a formal and operational semantics for them. Description logics are commonly based on a fragment of first-order logic with acceptable expressive power which still allows for decidable inference procedures (cf. [56]). Systems operating on description logics, such as LOOM, KL-ONE and FaCT, provide formal semantics with reasoning support.

The second important class of logic-based approaches in ontology engineering are based on *frame logic*. Frame logic has been used to express ontologies in projects such as Ontobroker (see also Section 4.1.3.3). Frames, essentially, are classes that may possess properties called attributes. In this way, a frame provides a certain context for modeling one specific aspect of a domain, similar to classes in object-oriented approaches. Ontology representation languages for frame-based systems are XOL [48] and the Open Knowledge Base Connectivity (OKBC) [13], [14].

An example for a language based on *predicate logic* is the *Knowledge Interchange Format* (KIF) (cf. [34], [35]), which has been designed for interchanging knowledge among disparate systems. Semantically, KIF knows four categories of modeling primitives, called constants: objects, functions, relations, and logical constants expressing truth conditions. While KIF has rarely been used to express ontologies, it is the formal basis of *Ontolingua*, a representation language that found some recognition within the ontology engineering community. The discussion on Ontolingua has shown the dilemma of full-fledged logic-based approaches to defining ontologies. One main reason for introducing such languages in the first place, their high expressive power, has also become one of their decisive disadvantages. [47] argues that the expressiveness of such languages in ontology engineering has not been met by adequate software tools that could supply means to control their complexity. In the case of Ontolingua, for example, no reasoning support has ever been provided. For that reason, the ontology engineering community has lately been focusing its research efforts towards representation languages offering only restricted expressive power in terms of reasoning capacity, and whose syntax is based on Web standards (such as XML and RDF). The most prominent example of such a rather compact ontology language is DAML+OIL.

4.3.2 DAML+OIL

The Defense Advanced Research Projects Agency (DARPA) Agent Markup Language and Ontology Interface Layer (DAML+OIL) is an ongoing research effort that is trying to combine aspects provided by different communities. It offers an XML- and RDF-based syntax in combination with modeling primitives borrowed from logic-based systems. Although DAML+OIL is an important research project, its designers have not yet decided on a definite set of modeling primitives that would allow for a fixed epistemological interpretation of the language. Extensions to DAML+OIL will be the result of further research and more practical experience with the current specification (cf. [17]). However, it is interesting to note that the designers of DAML+OIL chose to base the representation format for this language on XML syntax and RDF in order to align with current standards of the (Semantic) Web community.

4.3.3 An XML DTD for EOS Ontologies

Various formalisms and data formats for expressing ontologies have been proposed. Among the most prominent representation approaches are RDF, RDFS and DAML+OIL, all of which are build upon XML and associated with their specific semantics. The very basic and simple structure of EOS ontologies, which understands a domain formalization as a set of concepts, could be expressed in any of these languages. Yet, EOS actually does not depend upon the semantics underlying such languages. In fact, it is feasible to describe EOS ontologies directly using an appropriate XML DTD for EOS ontologies, such as shown in Figure 4.20.

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE ONTOLOGY [
    <!ELEMENT CONCEPT     (NAME? | DESCRIPTION? | VALUE? | COMPONENT*)>
    <!ELEMENT NAME        (#PCDATA)>
    <!ELEMENT DESCRIPTION (#PCDATA)>
    <!ELEMENT VALUE       (#PCDATA)>
    <!ELEMENT COMPONENT   (NAME? | DESCRIPTION? | REFERENCE? | COMPONENT*)>
    <!ELEMENT REFERENCE   EMPTY>

    <!ATTLIST CONCEPT id       ID #REQUIRED>
    <!ATTLIST CONCEPT category (particular|universal) #REQUIRED>
    <!ATTLIST CONCEPT type
            (isa|identity|and|or|not|rule|acquire|generate|query) #IMPLIED>
    <!ATTLIST COMPONENT id    ID #REQUIRED>
    <!ATTLIST COMPONENT type
            (isa|identity|and|or|not|rule|acquire|generate|query) #IMPLIED>
    <!ATTLIST COMPONENT card  (#PCDATA) #IMPLIED>
    <!ATTLIST COMPONENT idref IDREF #IMPLIED>
    <!ATTLIST REFERENCE idref IDREF #REQUIRED>
]>
```

Figure 4.20: The EOS DTD, an XML DTD for EOS Ontologies

The XML elements CONCEPT, COMPONENT and REFERENCE of this DTD mirror the basic constituents of Concept Theory as presented in Chapter 2. NAME and DESCRIPTION are optional elements that can be utilized for explicating CONCEPT and COMPONENT elements in a human readable form. The VALUE element is used for specifying actual values of CONCEPT elements that are representing particulars. An according graphical outline of the element hierarchy as defined by the EOS DTD of Figure 4.20 is displayed in Figure 4.21.

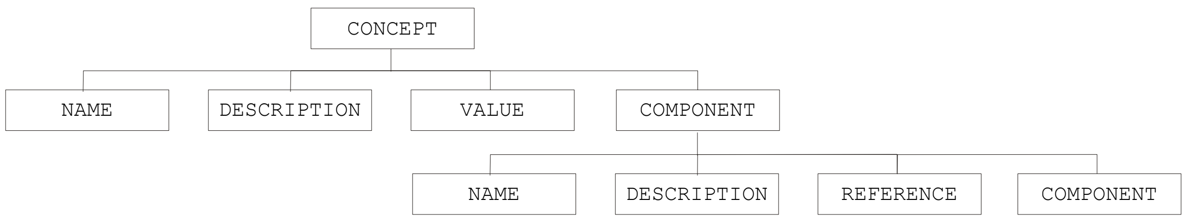


Figure 4.21: The Element Hierarchy of the EOS DTD

Next to the element hierarchy, the EOS DTD also defines element attributes and their values. The core elements CONCEPT, COMPONENT and REFERENCE possess various attributes in order to define them precisely according to the semantics of Concept Theory. The XML attribute `category` allows for determining whether a given `CONCEPT` represents a universal or a particular, while the `type` attribute pertains to the different kinds of concepts as recognized by Concept Theory. The cardinality of concept components can be specified using `card`. Finally, `id` and `idref` specify unique element identifiers and their references that are used for modeling Concept Theory's concept labels and references. The following sections give a concise introduction to the syntax definitions of the EOS DTD that have been briefly sketched in this outline.

4.3.3.1 The XML Document Type ONTOLOGY

The root element of an XML document compliant with a given XML DTD is specified by the document type declaration (marked with the key word `DOCTYPE`). The EOS DTD of Figure 4.20 defines the root element as

```
<!DOCTYPE ONTOLOGY [
   …
]>.
```

Strictly speaking, this document type *declaration* is not part of the actual document type *definition* (DTD) that is given within its opening and closing parenthesis. Thus, in its strict sense, the DTD of `DOCTYPE ONTOLOGY` must be equated with the listing of all XML elements and attributes specified *in between* the document type declaration parenthesis. Following common practice, though, we will identify the listing of Figure 4.20 *as a whole* (presumably stored in a separate file) with the *EOS DTD* that defines the syntactical composition of XML documents containing EOS ontologies. The basic structure of any such XML document then is

```
<?xml version="1.0" ?>
<ONTOLOGY>
   …
</ONTOLOGY>
```

where all associated concept definitions (represented by `CONCEPT` elements) must be given subsequent to the starting tag `<ONTOLOGY>` and preceding to the ending tag `</ONTOLOGY>`. Naturally, as Concept Theory is unicategorical, any EOS ontology expressed in XML will consist simply of an arbitrary number of such `CONCEPT` elements, possibly containing subelements. A tabular overview of this structural makeup is presented in Table 4.1.

Document Type `ONTOLOGY`			
Element	*Arity*	*Description*	*Example*
`CONCEPT`	0-n	The DTD element representing a (universal or particular) concept C.	`<CONCEPT id="20" category="particular">` … `</CONCEPT>`

Table 4.1: The Document Type `ONTOLOGY`

The complete explanation of all EOS DTD elements, including their attributes and subelements, will be delivered in the following sections.

4.3.3.2 The XML Element CONCEPT

The XML Element `CONCEPT`, as the name implies, is representing the notion of concepts as defined by Concept Theory. The EOS DTD definition of the element `CONCEPT` (cf. Figure

4.20) specifies its four subelements and their arities. This specification determines that for any XML file conforming to the EOS DTD, within the starting and ending tags of a CONCEPT element, one may place one or zero instances of each of the first three subelements NAME, DESCRIPTION and VALUE. Additionally, an arbitrary number (n or zero) of instances of the subelement COMPONENT can be added. An overview of all subelements of CONCEPT and their semantics is listed in Table 4.2.

Element CONCEPT			
Subelement	*Arity*	*Description*	*Example*
NAME	0-1	The concept name of a concept C in natural language.	`<NAME>PARTICULAR:FISCHER</NAME>`
DESCRIPTION	0-1	A description of concept C in natural language.	`<DESCRIPTION>` `The German politician` `Joschka Fischer` `</DESCRIPTION>`
VALUE	0-1	The (particular) value of the concept C.	`<VALUE>Joschka Fischer</VALUE>`
COMPONENT	0-n	A component (subconcept) of concept C.	`<COMPONENT … >` see *Table 4.5* `</COMPONENT>`

Table 4.2: Subelements of the DTD Element CONCEPT

The subelement NAME identifies a concept in natural language, while DESCRIPTION is intended for giving an explanation of the concept for human users. A VALUE subelement is used for specifying particular values. Finally, the COMPONENT subelement is depicting concept components. The structure of such a COMPONENT will be discussed separately later on in Section 4.3.3.3. Next to subelements, the representation of EOS concepts in XML also makes use of a specific set of XML attributes providing additional information for the CONCEPT element, namely id, category and type. Table 4.3 lists these attributes of CONCEPT and their usage in XML files.

Element CONCEPT			
Attribute	*Status*	*Description*	*Values*
id	#REQUIRED	A unique identifier for a CONCEPT element.	0, 1, 2, 3, …
category	#REQUIRED	The ontological category (kind) a CONCEPT element is representing.	particular, universal
type	#IMPLIED	The type of the concept C according to Concept Theory. If left unspecified, C is regarded as representing a common existent (e.g. PERSON or **PARTICULAR:**FISCHER) as opposed to semantically distinct concepts recognized by Concept Theory, i.e. occurrences of **ISA**, conditions, rules or laws.	isa, identity, and, or, not, rule, acquire, generate, query

Table 4.3: Attributes of the DTD Element CONCEPT

The XML representation of any EOS concept is based on this set of subelements and attributes of the element CONCEPT. The first attribute, id, determines a unique identifier for the respective CONCEPT. Attributes category and type are used for specifying its EOS classification. Table 4.4 shows the particular **PARTICULAR:**FISCHER and the universal PERSON in XML syntax.

Concept	*XML Syntax*
“Joschka Fischer”	`<CONCEPT id="20" category="particular">` `<NAME>PARTICULAR:FISCHER</NAME>` `<DESCRIPTION>The German politician Joschka Fischer.</DESCRIPTION>` `<VALUE>Joschka Fischer</VALUE>` `</CONCEPT>`
PERSON	`<CONCEPT id="2" category="universal">` `<NAME>PERSON</NAME>` `<DESCRIPTION>A person.</DESCRIPTION>` `</CONCEPT>`

Table 4.4: Two EOS Particulars in XML Syntax

Both, **PARTICULAR:**FISCHER and PERSON, in this example are defined as bare concepts, i.e. they possess no components of their own. Modeling more complex concept structures in XML, including components and references, will be the subject of the following section.

4.3.3.3 The XML Elements `COMPONENT` and `REFERENCE`

EOS concepts usually exhibit some internal structure which is expressed using components. According to the EOS DTD, concept components can be represented by the subelement `COMPONENT`. Table 4.5 gives an overview on possible subelements of `COMPONENT`. The subelements `NAME` and `DESCRIPTION` are used analogously to their employment in `CONCEPT` elements. `REFERENCE`, on the other hand, depicts a concept references, as needed e.g. in **IDENTITY** concepts. `COMPONENT` elements themselves may also contain other `COMPONENT` subelements.

Element `COMPONENT`			
Subelement	*Arity*	*Description*	*Example*
`NAME`	0-1	The name of a component D of a concept C in natural language (corresponds to a concept name within the same ontology).	`<NAME> … </NAME>`
`DESCRIPTION`	0-1	A description of the concept component in natural language.	`<DESCRIPTION>` … `</DESCRIPTION>`
`REFERENCE`	0-1	A reference to a concept C.	`<REFERENCE idrefs=…` (see *Table 4.9*) `/>`
`COMPONENT`	0-n	A subcomponent of component D.	`<COMPONENT>` … (see *Table 4.7*) `</COMPONENT>`

Table 4.5: Subelements of the DTD Element `COMPONENT`

The attributes of the `COMPONENT` element are listed in Table 4.6. As with `CONCEPT` elements, any `COMPONENT` is assigned a unique `id`. The type and cardinality of a `COMPONENT` may be stated using `type` and `card`, respectively. Finally, the `idref` attribute (realized through an XML IDREF) is a means to address any `CONCEPT` element within a given ontology.

Element COMPONENT			
Attribute	*Status*	*Description*	*Values*
Id	#REQUIRED	A unique identifier for a COMPONENT element.	0, 1, 2, 3, …
Type	#IMPLIED	The type of the component D.	isa, identity, and, or, not, rule, acquire, generate, query
Card	#IMPLIED	The cardinality of a COMPONENT.	*, 2, 3, …
Idref	#IMPLIED	A link (IDREF) to a CONCEPT or COMPONENT.	0, 1, 2, 3, …

Table 4.6: Attributes of the DTD Element COMPONENT

An example for a concept with two components is **ISA:**PE, an **ISA** occurrence stating that a *person* is an *existent*. The XML representation of this concept is developed in Table 4.7.

Concept	*XML Syntax*

	`<CONCEPT id="1" category="universal">`
	`  <NAME>EXISTENT</NAME>`
	`  <DESCRIPTION>`
	`    An existent, the most general concept`
	`    and therefore the root element of the`
	`    ontology.`
	`  </DESCRIPTION>`
	`</CONCEPT>`

	`<CONCEPT id="2" category="universal">`
	`  <NAME>PERSON</NAME>`
	`  <DESCRIPTION>A person.</DESCRIPTION>`
	`</CONCEPT>`

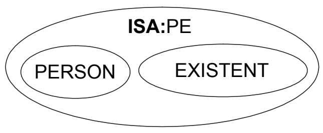
	`<CONCEPT id="3" type="isa" category="universal">`
	`  <NAME>ISA:PERSON_EXISTENT</NAME>`
	`  <DESCRIPTION>A PERSON is an EXISTENT.</DESCRIPTION>`
	`  <COMPONENT id="4" idref="2">`
	`    <NAME>PERSON</NAME>`
	`  </COMPONENT>`
	`  <COMPONENT id="5" idref="1">`
	`    <NAME>EXISTENT</NAME>`
	`  </COMPONENT>`
	`</CONCEPT>`

Table 4.7: Specialization among Universals in XML Syntax

As shown in Table 4.7, an idref of a COMPONENT element (e.g. idref="2" of the PERSON component whose own identifier is id="4") links to another COMPONENT or CONCEPT id (e.g. id="2" of the PERSON concept definition), thus realizing EOS concept labels. EOS concept references, on the other hand, are expressed by the idref attribute of REFERENCE elements (cf. Table 4.8). As all CONCEPT and also COMPONENT elements possess their own unique id, a REFERENCE need not be formulated using the complete component path but can directly address the respective concept or (sub)component. For example, **ISA:**PE of Table 4.7 possesses two components, PERSON and EXISTENT, which can be expressed as **ISA:**PE[1] and **ISA:**PE[2]. Yet, as the components of **ISA:**PE are assigned their own unique `ids` in the XML representation, these `ids`, namely `"4"` and `"5"`, can be used directly in `REFERENCE` elements[53].

Element `REFERENCE`			
Attribute	*Status*	*Description*	*Values*
`idref`	`#IMPLIED`	A link (IDREF) to a `CONCEPT` or `COMPONENT`.	`0, 1, 2, 3, …`

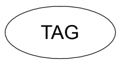
Table 4.8: Attributes of the DTD Element `COMPONENT`

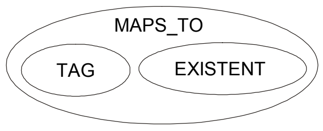
Table 4.9 shows a complete example in XML notation, taken from the document classification scenario. Listed are **QUERY:**MAPS_TO_AUTHOR (see also Figure 4.4 in Section 4.2.2) and the set of concepts it refers to, including the **IDENTITY** condition between its TAG components.

Concept	*XML Syntax*

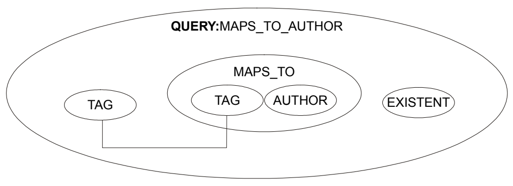
	`<CONCEPT id="1" category="universal">`
	`  <NAME>EXISTENT</NAME>`
	`  <DESCRIPTION>`
	`    An existent, the most general concept`
	`    and therefore the root element of the`
	`    ontology.`
	`  </DESCRIPTION>`
	`</CONCEPT>`

	`<CONCEPT id="6" category="universal">`
	`  <NAME>AUTHOR</NAME>`
	`  <DESCRIPTION>`
	`    The author of a (Web) document.`
	`  </DESCRIPTION>`
	`</CONCEPT>`

	`<CONCEPT id="7" category="universal">`
	`  <NAME>TAG</NAME>`
	`  <DESCRIPTION>An XML tag.</DESCRIPTION>`
	`</CONCEPT>`

	`<CONCEPT id="8" category="universal">`
	`  <NAME>MAPS_TO</NAME>`
	`  <DESCRIPTION>`
	`    The mapping between XML tags and ontology`
	`    concepts.`
	`  </DESCRIPTION>`
	`  <COMPONENT id="9" idref="7">`
	`    <NAME>TAG</NAME>`
	`  </COMPONENT>`
	`  <COMPONENT id="10" idref="1">`
	`    <NAME>EXISTENT</NAME>`
	`  </COMPONENT>`
	`</CONCEPT>`

[53] Note that this is possible because the component `ids` in the XML representation are distinct from concept `ids`, e.g. the concept definition of `PERSON` in Table 4.7 has been attributed with `id="2"`, while the component `PERSON` of `ISA:PERSON_EXISTENT` has been assigned the value `id="4"`.

 QUERY:MAPS_TO_AUTHOR
MAPS_TO
TAG
TAG
AUTHOR
EXISTENT
	`<CONCEPT id="11" type="query" category="universal">`
	`<NAME>QUERY:MAPS_TO_AUTHOR</NAME>`
	`<DESCRIPTION> A KEP query[54]. </DESCRIPTION>`
	`<COMPONENT id="12" idref="7">`
	`<NAME>TAG</NAME>`
	`</COMPONENT>`
	`<COMPONENT id="13" idref="8">`
	`<NAME>MAPS_TO</NAME>`
	`<COMPONENT id="14" idref="7">`
	`<NAME>TAG</NAME>`
	`</COMPONENT>`
	`<COMPONENT id="15" idref="6">`
	`<NAME>AUTHOR</NAME>`
	`</COMPONENT>`
	`</COMPONENT>`
	`<COMPONENT id="16" idref="1">`
	`<NAME>EXISTENT</NAME>`
	`</COMPONENT>`
	`</CONCEPT>`

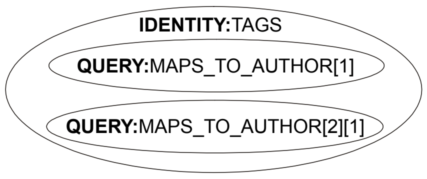
 IDENTITY:TAGS
QUERY:MAPS_TO_AUTHOR[1]
QUERY:MAPS_TO_AUTHOR[2][1]
	`<CONCEPT id="17" type="identity" category="particular">`
	`<NAME>IDENTITY:TAGS</NAME>`
	`<DESCRIPTION>`
	`The identity condition valid between the TAG components of QUERY:MAPS_TO_AUTHOR.`
	`</DESCRIPTION>`
	`<COMPONENT id="18">`
	`<NAME>QUERY:MAPS_TO_AUTHOR[1]</NAME>`
	`<REFERENCE idref="12"/>`
	`</COMPONENT>`
	`<COMPONENT id="19">`
	`<NAME>QUERY:MAPS_TO_AUTHOR[2][1]</NAME>`
	`<REFERENCE idref="14"/>`
	`</COMPONENT>`
	`</CONCEPT>`

Table 4.9: Specialization among Universals in XML Syntax

In this manner, all ontology concepts can be represented conforming to the XML DTD for EOS ontologies. While this representation is useful as an input/output format for exchanging data with and among EOS systems, it is not necessarily an ideal internal storage format. For small ontologies referring to a domain with only a limited number of particulars it might suffice to store an EOS ontology in the form of an XML document and query this information using an appropriate XML parser, e.g. Xerces[55]. However, the size of an ontology will usually grow quickly with the acquisition of new particulars, the actual domain data. Simple parsing techniques on large XML files will therefore soon prove inadequate for providing fast access to bigger ontologies. Here, database technology including indexing techniques, standardized query language support and physical query optimization offers a better solution. Thus, database technology can be seen as an enabler for efficient ontology data management in EOS systems. For this reason, in the following section we will sketch how EOS ontologies could be represented in a RDBS.

4.4 Representing EOS Ontologies in a RDBS

An EOS system such as the one motivated in the document classification example of this chapter must be able to persistently store, alter and efficiently query its internal ontology.

[54] See also 4.2.2 for a discussion on how such a query may be used by the knowledge extraction procedure (KEP) in the context of knowledge acquisition.
[55] http://xml.apache.org/xerces2-j/index.html

Requirements of this kind are classically met by the services of database management systems (DBMS). DBMS offer persistent storage of data, along with higher-level query language support, transaction management, data integrity, as well as data security and recovery mechanisms, etc. Our discussion will focus on storing EOS ontologies using relational DBMS (RDBMS) that are generally supporting SQL as a standardized query language. In this section, we will motivate how the structure of EOS ontologies may be translated into a relational database schema and how an EOS system may make use of an relational database that stores ontology concepts according to this schema. In particular, we will show how ontology concepts stored in a relational database can be indexed for efficient access. As indicated in Sections 3.1.3.4 and 4.2.3, the main classes of query operations we can expect against an ontology are:

- Returning occurrences of a concept C (with a possibility to determine whether particulars, universals or both kinds of concepts should be included in the result set).
- Returning (mediate and immediate) parent concepts of a concept C.

Our relational approach to EOS ontologies recognizes this and provides a solution with regard to these kinds of queries. The indexing strategy we are employing takes advantage of the specific structure of EOS ontology graphs in order to support these queries effectively.

4.4.1 Graph Representation

The overall structure of an EOS ontology is that of a directed acyclic graph[56] (DAG) that mirrors the specialization hierarchy (cf. Chapter 2), starting from a root node (the most general ontology universal which, throughout this thesis, has been EXISTENT) to the leaf nodes (either universals possessing no further specializations, or ontology particulars). Commonly, within an EOS ontology the share of particulars will outnumber that of universals by a considerable margin. This is consistent with the general idea underlying EOS that universals are used for providing domain categories (e.g. PERSON) while particulars are representing numerous concrete domain facts (e.g. **PARTICULAR:**FISCHER, **PARTICULAR:**MERKEL, etc.) reflecting the current information extent of an EOS system. Thus, the outset of representing EOS ontologies in a relational way is that we can expect a limited number of universals that are describing the domain model in the form of a specialization hierarchy, and a comparably large number of particulars that are instantiating these universals. Figure 4.22 shows a corresponding schematic ontology graph representation. From Section 2.2.4.2.3 we can assume that any particular of the ontology graph possesses only one immediate parent. For universals this may not be the case as Concept Theory allows for multiple inheritance, e.g. node E in Figure 4.22 has two parent nodes, A and B. Furthermore, only nodes representing universals can have child nodes (as particulars cannot be further instantiated). Considering their dissimilar properties, it is reasonable to treat these two kinds of concepts differently in terms of their role in the ontology graph. While universals define the structured part of an ontology graph (the actual specialization hierarchy), particulars will only mark immediate extensions to ontology universals. For the following discussion on effectively indexing an ontology graph for efficient access when stored in a RDBS, we will therefore focus on the basic structure as determined by universals.

[56] An ontology graph is acyclic in that it represents a domain's specialization hierarchy where arcs are corresponding to **ISA** occurrences that are relating concepts of any other kind, the nodes of the graph. The semantics of **ISA** (cf. Section 2.2.4.2) rule out that concepts are being defined as generalizations of more general concepts, i.e. graphically speaking, outgoing arcs will always be oriented in the direction away from the root node. Trivially, there is one possible exception to this rule, namely arcs that are pointing back to the concept node C they originated from, which presupposes **ISA** occurrences of the kind **ISA:**SELF:=(2, (C,C)) expressing the trivial statement "*a C is a C*". Without loss of generality we will leave aside such self-references and regard an ontology graph as a DAG.

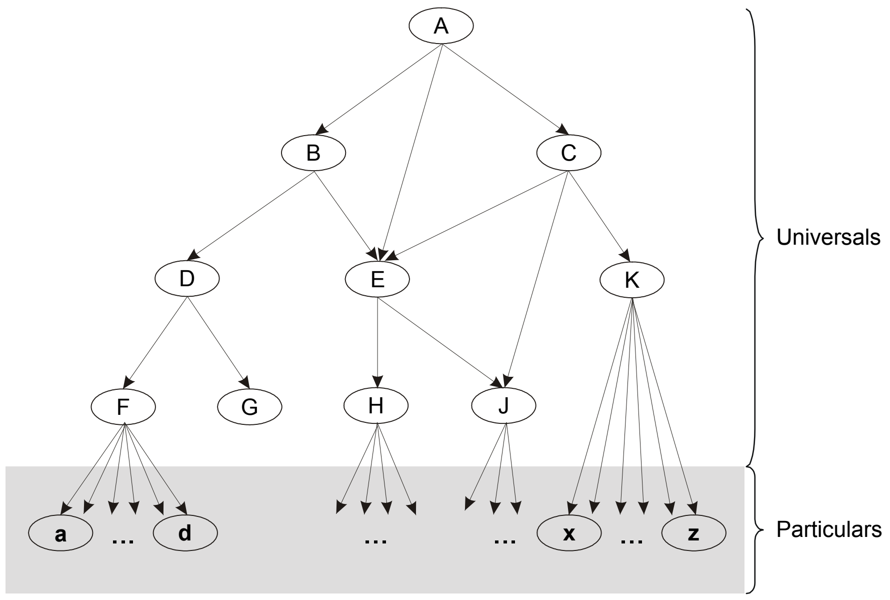


Figure 4.22: Structure of Ontology Graphs

As already pointed out in Section 2.2.4.2.2, within an ontology graph each node belongs to a *level*, where graph levels are made up of nodes with equal distance from the root node. This implies that levels are reflecting the path length from any given graph node to the root node, A in case of Figure 4.22, where a path consists of the **ISA** arcs leading from A to the graph node, and the path length is determined by the number of **ISA** arcs involved.

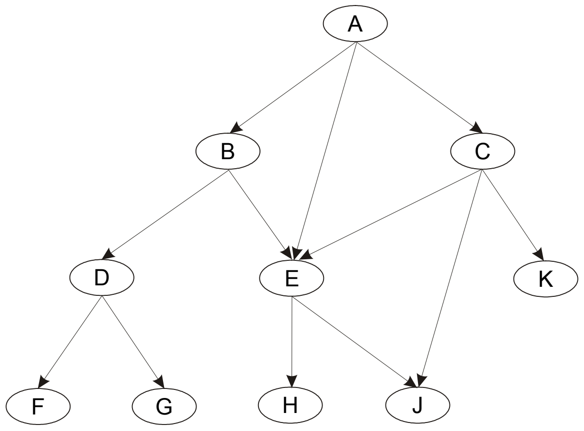


Figure 4.23: The Ontology Graph of Universals

Figure 4.23 depicts the graph of universals of Figure 4.22, amplified by graph levels (denoted by dotted lines).We informally introduce the notation

$(\text{NODE}_1, \text{NODE}_2, \ldots, \text{NODE}_n)$

for referring to a path that is delineated by arcs representing the **ISA** occurrences

ISA:1:=(2,(NODE_1,NODE_2)), **ISA:**2:=(2,(NODE_2,…)), …, **ISA:**N-1:=(2,(…,NODE_n)).

Concerning the assignment of nodes into their levels we can distinguish two special cases. Graph nodes possessing more than one immediate parent may be accessible from the root node by paths of different lengths, e.g. the paths (A,B,E,H) and (A,E,H) both lead to node H. In such a case, the respective node, here H, is situated in the level determined by the *longest* path, 3 for node H. On the other hand, nodes that could be situated in two (or more) different levels, e.g. C may theoretically be attributed to both, level 1 and 2, will be placed in the uppermost possible level (which corresponds to their actual distance from the root node), i.e. C is placed in level 1.

This graph structure must be translated into a relational schema. Straightforwardly, this may be done by simply storing it directly in relations, e.g. by associating child nodes with their parents as done in Figure 4.24. The schema for the relations *Universals* and *Particulars* is identical with attributes *Concept*, *Parent* and *ISA Arc*. Each tuple of these relations relates a concept node to one of is parents, reporting also the specific **ISA** occurrence involved. Note that all concept names are unique within an ontology, i.e. the attribute *Concept* can serve as part of the primary key for these relations[57].

Universals		
Concept	**Parent**	**ISA Arc**
A	*NULL*	*NULL*
B	A	ISA1
C	A	ISA2
D	B	ISA3
...	...	...

Particulars		
Concept	**Parent**	**ISA Arc**
a	F	ISA12
d	F	ISA13
x	K	ISA14
z	K	ISA15
...	...	...

Figure 4.24: Relations for a Simple Ontology Graph Representation

As reported in Section 3.1.3.4 and further exemplified in Section 4.2.3, the main task for an EOS system in knowledge retrieval is to find a specific concept within the ontology graph and return it along with all of its occurrences (provided they are consistent with the query condition). Following the simple solution of Figure 4.24 for representing an ontology graph using relations, it is obvious that locating a specific concept C poses no problems, as concept names are unique and can be ordered alphabetically. However, computing mediate parent concepts, as well as the complete set of occurrences of C would imply a series of costly self-joins within the *Universals* relation. For providing a more efficient solution to this problem that is avoiding multiple self-joins, we propose an extra indexing strategy for ontology graphs. The basic idea is to index the ontology graph in a way that each concept is assigned a numerical concept identifier (CID) and an end identifier (EID) that together mark a range that comprises all CIDs of the concept's occurrences. In other words, let $G_O:=(V,E)$[58] be an ontology graph, let CID_n and EID_n be the CID and EID of a node $n \in V$. It follows that

$\forall a,b \in V, a \neq b: CID_a < CID_b \leq EID_a \Leftrightarrow$ a is an (immediate or mediate) ancestor of b,

[57] Note also that at this point we are only focusing on representing the graph structure as such in relational tables. For a complete translation of an EOS ontology into relations more attributes and/or tables are needed for storing information on concept components and particular values. A complete relational schema for EOS ontologies will be developed later in Section 4.4.4.

[58] See Definition 2.13 for the concise definition of ontology graphs.

a property that allows to select the occurrences of a concept C using a single, simple SQL statement.

Universals				
Concept	**CID**	**EID**	**Parent**	**ISA Arc**
…	…	…	…	…

Particulars				
Concept	**CID**	**EID**	**Parent**	**ISA Arc**
…	…	…	…	…

Figure 4.25: Modified Relations for an Optimized Ontology Graph Representation

Supposing correspondingly modified relations *Universals* and *Particulars* as depicted in Figure 4.25, all universal (or particular) occurrences of concept C can then be queried according to this schema:

SELECT Concept
FROM Universals
WHERE $(CID_C < \text{CID})$ AND $(\text{CID} \leq EID_C)$

SELECT Concept
FROM Particulars
WHERE $(CID_C < \text{CID})$ AND $(\text{CID} \leq EID_C)$

The union of both statements produces all particular and universal occurrences of C. For computing all ancestors of C we can use the same index property, which leads to the SQL statement schema:

SELECT Concept
FROM Universals
WHERE $(\text{CID} < CID_C)$ AND $(CID_C \leq \text{EID})$

SELECT Concept
FROM Particulars
WHERE $(\text{CID} < CID_C)$ AND $(CID_C \leq \text{EID})$

In this way, the most frequent and important types of queries an EOS system depends on can be efficiently processed using an index that provides the above property. The following sections will show how such an index can be computed for an EOS ontology graph.

4.4.2 The Indexing Algorithm

The ontology graph is a leveled directed DAG that may possibly include nodes with multiple parents, e.g. node J in Figure 4.23 has two incoming edges. In a first approach to an indexing technique for arbitrary ontology graphs, we will first regard a specific class of such graphs, namely trees where each node may only possess one incoming edge (which corresponds to a domain model that does not allow for multiple inheritance). The sought for index property

$\forall a,b \in V,\ a \neq b$: $CID_a < CID_b \leq EID_a \Leftrightarrow$ a is an (immediate or mediate) ancestor of b,

for a complete ontology tree $G_O:=(V,E)$ can then be achieved by using a depth-first numbering scheme. Note that the indexing algorithm of this section only considers the graph $G_{\text{Universals}}$ spanned by nodes representing universals alone, i.e.

$G_{\text{Universals}}:=(V,E)$, and $\forall v \in V$: v represents a concept $\in \Phi_U$.

Graph nodes representing particulars are assigned the CIDs and EIDs of their parents. An algorithm that computes node CIDs and EIDs for $G_{\text{Universals}}$ is displayed in Figure 4.26. The recursive algorithm `index_tree()` traverses an ontology tree in a depth-first manner, using a global index `i` that is incremented in each recursion for computing the values $\texttt{CID}_n$ and $\texttt{EID}_n$ of the current graph node `n`. The current value of `i` determines $\texttt{CID}_n$ and is afterwards incremented. This assures that each node is assigned a unique $\texttt{CID}_n$ that is greater than the `CID` of the parent node. For leaf nodes[59] the algorithm determines equal values for their $\texttt{CID}_n$ and $\texttt{EID}_n$. If the current node `n` possesses child nodes, `index_tree()` is called for them, too[60]. Then, $\texttt{EID}_n$ equals the `EID` of the last child node. In this way, the index property holds for all tree nodes. Trivially, `index_tree()` terminates as the ontology tree is acyclic and finite.

```
Integer i:= 0;
index_tree( root_node );

index_tree( Node n )
{
    CID_n := i;
    i     := i+1;

    forall( c in child nodes of n ) do { index_tree(c); }

    EID_n := i;
}
```

Figure 4.26: Indexing Algorithm for Ontology Trees

This algorithm indexes the ontology tree spanned by universals in the desired manner. However, if the domain model should change and new universals have to be integrated into the ontology tree, this would mean a costly recomputation of CID and EID values. Although it is rarely the case that the domain model changes structurally – usually only particulars are added to an otherwise static domain model, and particulars possess no independent CIDs and EIDs – it is advisable to modify the increment of `index_tree()` so that (a limited number of) new universals can be added to the graph without having to change the CID and EID indexes of other tree nodes. Deciding on the exact *range* of a tree node `n` (i.e. the maximum number of child nodes it may possess) is a choice that a domain expert who is designing an ontology should at least to some degree be able to influence. We propose to define a uniform range for all nodes of an ontology tree level, while the ranges of different levels may vary. The *capacity* of a tree node `n` in level `k`, i.e. the maximum number of all siblings of `n`, can then be recursively formulated as

$$\begin{array}{ll} \text{capacity}_k := 0 & \text{for } k=\text{MaxULevel} \\ \text{capacity}_k := \text{range}_k + \text{range}_k*\text{capacity}_{k-1} = \text{range}_k * (1 + \text{capacity}_{k-1}) & \text{for } k<\text{MaxULevel}. \end{array}$$

As capacity_k denotes the number of possible siblings of a node `n` in level `k`, it follows that

$$\text{EID}_n = \text{CID}_n + \text{capacity}_k.$$

The capacities of each ontology tree level can be computed separately and then be used in function `index_tree()` of the indexing algorithm. This is depicted in Figure 4.27. A complete example for an indexed ontology tree using the improved indexing algorithm is given in Section 4.4.4.

```
Integer i:= 0;

compute capacities for all levels;
index_tree( root_node );
```

[59] Note that these leaf nodes are representing *universals*, not particulars, as in this section we are only regarding $G_{\text{Universals}}$ (cf. Section 4.4.1).

[60] Note that this algorithm is not apt for parallelization as it depends on the consecutive incrementation of index `i`.

```
index_tree( Node n )
{
    CID_n := i;
    EID_n := CID_n + level capacity of n;
    i     := i+1;

    forall( c in child nodes of n ) do { index_tree(c); }

    i     := EID_n+1;
}
```

Figure 4.27: Improved Indexing Algorithm for Ontology Trees

4.4.3 The Transformation Algorithm

The indexing algorithm of the previous section has been defined for ontology trees that do not allow for multiple inheritance. In this section we are presenting a transformation algorithm that restructures an arbitrary ontology graph into an equivalent tree representation, so that the indexing algorithm can be applied to it. The transformation algorithm eliminates multiple incoming edges for a graph node by introducing a single, new parent node that is representing the set of former parent nodes. Figure 4.28 shows the amendments that have to be done for transforming the graph of Figure 4.23 into an ontology tree, where changes are highlighted in gray. Arcs that are being eliminated are crossed out, while the names of newly created parent nodes indicate the set of former parents they are substituting.

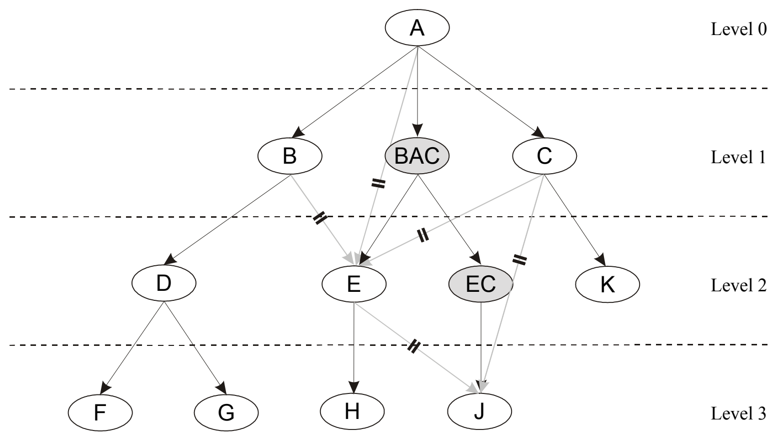


Figure 4.28: Transforming an Ontology Graph into an Ontology Tree

The transformation algorithm is depicted in Figure 4.29. The function `transform_tree()` presupposes the existence of two node vectors, `concept_vector`$_n$ and `parent_vector`$_n$, for each graph node `n`. Initially, the `concept_vector`$_n$ of all original ontology graph nodes will only contain one single concept name, namely of the concept that `n` is representing. After transformation, the ontology tree will contain nodes whose `concept_vector` may include more than one concept name. This is the case with all nodes that have been newly created by the transformation algorithm. The other vector, `parent_vector`$_n$, comprises all immediate parent nodes of `n`.

```
Integer l:= MaxLevel;     // start with graph level containing universals
                          // with longest distance from root node

forall( n in nodes ) do
{
   NodeVector concept_vector_n[]:= all concepts involved in n;
   NodeVector parent_vector_n[] := all immediate parents of n;
}

transform_tree( Node n )
{
   do
   {
     forall( n in nodes of level n ) do
     {
       if(number of nodes in parent_vector > 1) // n has multiple parents
       {
         delete all incoming arcs of n;
         if( a node p with concept_vector_p[]==parent_vector_n[]
              does not yet exist in level l-1 )
         {
           create new parent node p in level l-1;
           concept_vector_p[] := parent_vector_n[];
           parent_vector_p[]  := all parents of nodes in concept_vector_p[];
         }
         parent_vector_n[]  := (p);
       }
     }
   l  := i;

   } while(l>0)
}
```

Figure 4.29: Transformation Algorithm for Ontology Graphs

The algorithm processes the ontology graph level by level in a bottom-up fashion, i.e. starting from the deepest level (`MaxLevel`) of graph nodes still representing universals. A new parent node `p` is created whenever a node `n` possesses more than one parent (unless there already exists a parent candidate with the desired `concept_vector`). Note that the `parent_vector`$_p$ of parent node `p` is then assigned all parent nodes of all nodes it is substituting. For example, node J in the original graph of Figure 4.23 has parents E and C. The new node EC with `concept_vector[]:=(E, C)` will consequently get a `parent_vector[]:=(A, B, C)` because the parent of C is A, and additionally those of E are A, B and C. Hence, this method ensures that the new path from the root node to `n` includes nodes bearing all former ancestors of `n` in their `concept_vector`s, i.e. the complete generalization information for `n` that was originally expressed by the ontology graph structure is now condensed in the `concept_vector`s of nodes belonging to the direct path to the root node. Taking this into consideration, the resulting ontology tree bears information that is equivalent to the original graph. The transformation algorithm also terminates because it iterates through a finite number of graph levels, and the number of new parent nodes created in one iteration is also finite (as a maximum, within levels `MaxLevel` to `2`, each node may be assigned a new parent in an iteration step, while at levels `1` and `0` no new nodes are being created – the root node in level `0` possesses no parents, and all nodes of level `1`, per definition, have the root node as their only parent).

4.4.4 Relational Schema for Ontology Graphs

The preceding sections have presented the basic outset for translating an ontology graph into relations that can be stored in a RDBS. An ontology graph is first transformed into a semantically equivalent ontology tree. Afterwards, the CIDs and EIDs for tree nodes are computed using the indexing algorithm. Figure 4.30 shows the indexed[61] ontology tree for the graph example of Section 4.4.1 (CIDs are attached to the left-hand side of tree nodes and EIDs on their right-hand sides). On this basis we will now develop a complete relational schema for ontologies, based on the specific structure of ontology trees that are containing nodes representing parent node sets of the original graph. For example, node EC in Figure 4.30 substitutes nodes E and C of the original ontology graph of Figure 4.23. The special semantics of these nodes must be observed firstly in the schema design and secondly when querying the respective relations.

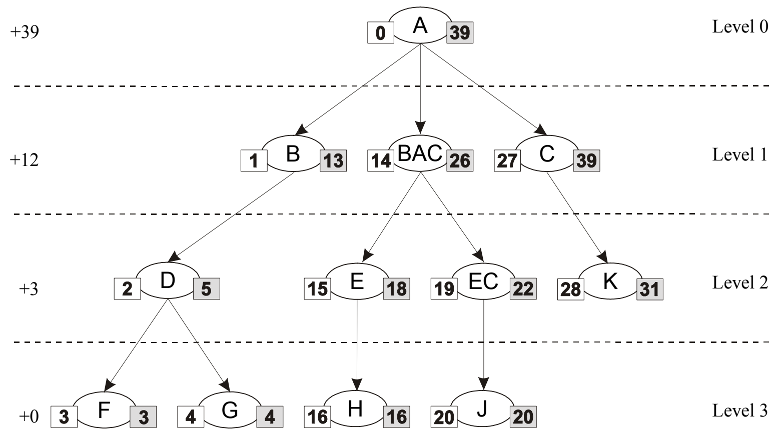


Figure 4.30: An Indexed Ontology Tree

In our approach, such nodes are not stored as independent entries of the relation *Universals* but they are used to provide extra indices on the original concepts they are representing. We will further illustrate this method with an example referring to the indexed ontology tree of Figure 4.30. The attributes of relation *Universals* are described in Figure 4.31.

Relation Universals

Attribute	Description	Type
<u>Concept</u>	The unique concept name of the universal.	CHAR(*)
Type	The type of the universal (NULL, ISA, etc.).	CHAR(*)
<u>CID</u>	A concept identifier of the universal, computed by the indexing algorithm.	INTEGER
EID	An end identifier of the universal, computed by the indexing algorithm.	INTEGER
Status	Reports whether CID and EID belong to an original graph node (Status=1), or to a node created by the transformation algorithm (Status=0).	INTEGER

[61] In this example we chose range=3 for all levels of the ontology graph. The resulting level capacities are listed on the left-hand side of Figure 4.30.

Relation Universals Parents

Attribute	Description	Type
<u>Concept</u>	The unique concept name of the universal.	CHAR(*)
Parent	An immediate parent concept of the universal.	CHAR(*)

Figure 4.31: The Relations Universals and Universals Parents

Following this schema, the relation *Universals* will contain several entries for nodes E and C, one for each node they are part of, e.g. three different tuples refer to node C because it is part of the tree nodes BAC, C and EC. Where the value for the attribute "Status" is 1, the tuple is describing the original position of node C[62]:

Universals				
Concept	**Type**	**CID**	**EID**	**Status**
...	...	...	...	...
C	NULL	14	26	0
C	NULL	19	22	0
C	NULL	27	39	1
E	NULL	15	18	1
E	NULL	19	22	0
...	...	...	...	...

This method of representing graph nodes allows to formulate graph queries according to the schemas that were introduced in Section 4.4.1:

- Return all siblings of C (that are universals):

```
SELECT   U2.Concept
FROM     Universals U1, Universals U2
WHERE    U1.Concept EQUALS "C"
         AND (U1.CID < U2.CID) AND (U2.CID ≤ U1.EID)
         AND U2.Status = 1
```

- Return all ancestors of C:

```
SELECT   DISTINCT U2.Concept
FROM     Universals U1, Universals U2
WHERE    U1.Concept EQUALS "C"
         AND (U2.CID < U1.CID) AND (U1.CID  ≤ U2.EID)
```

A corresponding schema for the relation *Particulars* is depicted in Figure 4.32, along with a second relation for storing the specific values of particulars.

[62] Note that the composite nodes BAC and EC of the ontology tree are not stored at all in relation *Universals*, as they are only required for providing extra index information on concepts.

Relation Particulars

Attribute	Description	Type
<u>Concept</u>	The unique concept name of the particular.	CHAR(*)
Type	The type of the particular (NULL, ISA, etc.).	CHAR(*)
Parent	The parent universal of the particular.	CHAR(*)
CID	A concept identifier of the parent universal, computed by the indexing algorithm.	INTEGER
EID[63]	An end identifier of the parent universal, computed by the indexing algorithm.	INTEGER

Relation Particulars Values

Attribute	Description	Type
<u>Concept</u>	The unique concept name of the particular.	CHAR(*)
Value[64]	The value of the particular.	CHAR(*)

Figure 4.32: The Relations Particulars and Particulars Values

The relations mentioned so far are describing universals and particulars as nodes in an ontology graph and are indexed by CID and EID values in a way that allows for an efficient retrieval of occurrences and generalizations of concepts. Hence, these relations pertain to the overall structure of an ontology. The internal structure of concepts, i.e. their component information, can be extracted from the relation in Figure 4.33.

Relation Components

Attribute	Description	Type
<u>Concept</u>	The unique concept name of the universal or particular.	CHAR(*)
Component	A component of the concept.	CHAR(*)
<u>Position</u>	The position of the component within the concept.	INTEGER

Figure 4.33: Components Relation for Universals and Particulars

Finally, concept descriptions in natural language for human readers are being stored in a separate relation called *Descriptions* as presented in Figure 4.34. This completes the relational schema for EOS ontologies.

[63] Strictly speaking, the *EID* attribute for relation *Particulars* is redundant as CID = EID for all particulars. However, we chose to keep the EIDs for *Particulars* as this allows us to formulate queries pertaining to relations *Universals* and *Particulars* symmetrically.

[64] The *Value* attribute has not been integrated into the *Particulars* relation because not every particular has necessarily a value of its own. Particulars that have other particulars as components will not be listed in relation *Particulars Values*.

Relation Descriptions

Attribute	**Description**	**Type**
<u>Concept</u>	The unique concept name of a universal or particular.	CHAR(*)
Description	A description of the universal or particular in natural language.	CHAR(*)

Figure 4.34: Descriptions for Universals and Particulars

This relational database schema, along with the transformation and indexing algorithms for preparing an ontology graph to be loaded into a RDBS can be used by an EOS system in the way we just described. The following section will present an EOS prototype that is using these techniques for internally storing and querying its ontology.

4.5 Prototype Implementation

This section is intended to offer a more technical view on EOS systems. In the beginning of this chapter, we have already presented an application scenario for EOS systems that involves classifying Web documents on the basis of an EOS ontology. Now, we will sketch how a concrete implementation of such a system can be designed and successfully put into practice. For providing an actual means for testing the theoretical points made in this thesis, we implemented a prototype EOS system that we used to categorize the information displayed on a restricted class of university Web sites.

All active system components were implemented in Java[65], version 1.3.1, using the development environment Borland Jbuilder4[66]. The employment of Java as a programming language allowed us to develop the EOS prototype as a platform independent application. The primary test environment for the prototype implementation was Microsoft Windows 2000. For the persistent storage of the EOS ontology we utilized a database that complies with the relational schema as presented in Section 4.4. The RDBMS we used for managing this database was TransBase[67] (version 5.2.2) of TransAction Software for Microsoft Windows NT/2000. Active system components of the EOS prototype can access the database using the TransBase JDBC driver that implements the standard JDBC (Java DataBase Connectivity) interface. Figure 4.35 shows the overall system architecture of the EOS prototype.

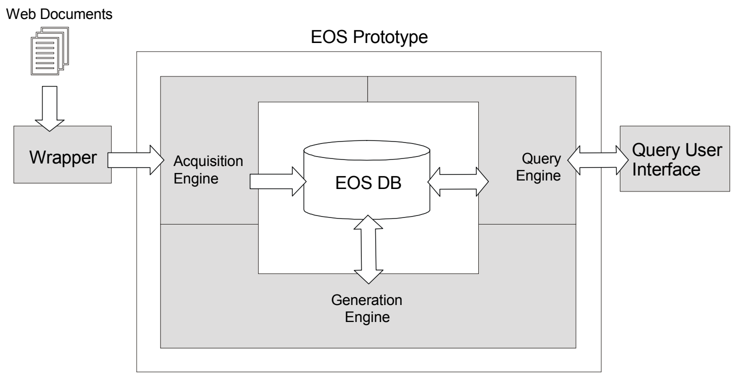


Figure 4.35: The EOS Prototype Architecture

[65] http://java.sun.com/
[66] http://www.borland.com/jbuilder/
[67] http://www.transaction.de/de/home/home.html

As lined out in Section 4.3, we propose XML as an exchange format for ontologies. The EOS prototype accepts an input ontology expressed in XML according to the EOS DTD, and creates a respective graph representation. This ontology graph is then indexed by the transformation and indexing algorithms we introduced in Sections 4.4.2 and 4.4.3, and finally stored inside the database the EOS prototype is using for persistent storage. Next to the ontology itself, the EOS prototype also accepts **ACQUIRE** and **QUERY** concepts as an input and processes them accordingly. **ACQUIRE** instances are created by an independent wrapper component that implements the knowledge extraction procedure (cf. Sections 3.2.3.2 and 4.2.2). Another separate system component, the query user interface of the EOS prototype, provides facilities to formulate queries which are then translated into **QUERY** concepts and passed to the query engine. Both, the acquisition engine and the query engine may trigger the generation engine that performs knowledge generation steps as defined by the **GENERATE** concepts of the ontology.

The core Java classes and methods that were developed to implement this functionality for the EOS prototype are grouped into two main packages, *eos.graph* and *eos.wrapper*:

- **eos.graph.*** comprises classes and methods concerning the structure of ontology graphs. It also provides the functionality of the knowledge acquisition, generation and retrieval engines. In particular, it contains an interface for translating between the XML, graph and relational representations of EOS concepts and ontologies.
- **eos.wrapper.*** includes all classes and methods needed for parsing Web sites and extracting new concepts from them.

In the following sections we will lay out the essential Java classes of these two packages and describe their attributes and methods where appropriate.

4.5.1 Java Classes in Package eos.graph

The classes of this Java package implement the core functionality of the EOS prototype. Classes eos.graph.Node and eos.graph.Graph define the structure of ontology graphs and provide methods for converting an XML representation of an EOS ontology into an equivalent ontology graph, and vice versa. As an auxiliary tool for parsing XML files, eos.graph.Graph uses Apache's Xerces Java Parser v.1.3.1, which supports the XML 1.0 recommendation of W3C and contains advanced parser functionality, such as DOM Level 2 v.1.0, and SAX v.2. All methods needed to store EOS ontologies in and to extract them from a relational database are included in class eos.graph.Graph2Db. Finally, classes eos.graph.RuleChecker and eos.graph.executeQuery provide methods to perform knowledge acquisition, generation and retrieval.

- **Class eos.graph.Node**

  Represents an ontology graph node.

  <u>*Attributes*</u>:

  Next to graph specific attributes (hashtables for successor and predecessor nodes, etc.), eos.graph.Node also contains attributes pertaining to concept attributes and EOS graph algorithms:

  String name, description, value: the concept name and its natural language description
  String type, category: the ontological classification of the concept
  Vector components: the components of the concept the node is representing
  String value: the value of a particular
  int cid, eid: the CID and EID of the node

int status: reports whether the node has been created by the transformation algorithm
Vector concepts: the concepts such a node is substituting

Methods:

The methods for graph nodes mainly concern updating the graph specific attributes when the ontology graph is built up, transformed or indexed.

- **Class eos.graph.Graph**

  Represents the overall ontology graph structure.

  Attributes:

  eos.graph.Graph contains different data structures for efficiently accessing nodes during graph operations. Additionally it comprises ontology graph specific information:

  String root: the root node of the graph
  int numLevels: the number of levels of the graph
  Vector capacities: the node capacities (cf. 4.4.2) for all graph levels

  Methods:

  Next to methods managing graph modifications, eos.graph.Graph also provides facilities for importing and exporting ontology information in XML:

  public void loadGraphXml(String XmlFile): reads in an XML file containing an EOS ontology and builds an equivalent ontology graph.
  Public String putGraphToXml(): outputs the ontology graph in XML format.

- **Class eos.graph.Graph2Db**

  Manages the persistent storage of an EOS ontology in a RDBS. Class eos.graph.Graph2Db utilizes the transformation and indexing algorithms we introduced in Sections 4.4.3 and 4.4.2.

  Methods:

  static Graph getTree(Graph g): implements the transformation algorithm.
  Public void indexTree(Graph g): implements the indexing algorithm.
  Public static void tree2Db(Graph g, String dbname): inserts the ontology tree g into the database dbname.
  Public String db2Xml(String dbname): extracts the complete ontology stored in database dbname and returns it in XML format.
  Public static void concept2Db(String XmlFile, String dbname): stores a newly acquired or generated concept in database dbname.

- **Class eos.graph.RuleChecker**

  Is able to interpret EOS **RULE**, **ACQUIRE** and **GENERATE** concepts..

  Methods:

  public static boolean check(eos.graph.Node node, eos.graph.Node rule): checks whether rule holds for the concept node (cf. Section 3.2.2.3).
  public void acquire(eos.graph.Node acquire): implements the knowledge acquisition procedure (cf. Section 3.2.3.2).
  public void generate(eos.graph.Node generate): implements the knowledge generation procedure (cf. Section 3.2.3.3).

- **Class eos.graph.executeQuery**

Is able to interpret EOS **QUERY** concepts.

Methods:

public void query(eos.graph.Node query, String database): implements the knowledge retrieval procedure (cf. 3.2.3.4). The user query is transformed into an SQL statement (cf. Section 4.4.4) and passed to the database.
Public eos.graph.Node rewriteQuery(eos.graph.Node query): performs semantic query rewriting (cf. Section 4.2.4 for examples on semantic query rewriting).

Supplementary to the eos.graph.executeQuery class, we also developed a graphical user interface for formulating ad-hoc queries. The user interface was implemented using the Javax Swing environment. It offers a query editor that may be used to browse the ontology that is stored in the internal database of the EOS prototype, and build an EOS **QUERY**. Optionally, the user interface also accepts **QUERY** concepts in XML syntax. Return values are, per default, displayed in XML syntax.

4.5.2 Java Classes in Package eos.wrapper

The Java package eos.wrapper implements a wrapper for university Web pages. The wrapper consists of a simple and fault tolerant parser. The corresponding ontology describes, on the one hand, the expected structure of the Web documents, e.g. regularities that can be observed on staff pages. On the other hand, it models the university domain that includes concepts like STUDENT, PROFESSOR, LECTURE, etc. The parser segments Web documents to any desired level, and interprets the parsed data according to the ontology concepts that relate to the information found. If the ontology contains no applicable pattern for some document portion, the parser will skip it and continue its execution with the next relevant document fragment. The wrapper is further equipped with methods to extract document links and navigate along them. The information identified within documents, e.g. the name of a person or chair, is subsequently molded into **ACQUIRE** concepts. After successfully parsing Web documents, the wrapper calls the acquisition procedure of the EOS prototype. The essential class of the eos.wrapper package is class eos.wrapper.Parser:

- **Class eos.wrapper.Parser**

  Provides the parser functionality for university Web pages. It implements the knowledge extraction procedure.

  Attributes:

  Along with attributes storing information specific to the standard Java ParserCallback functionality, eos.wrapper.Parser also includes:

  Vector concepts: the information found in Web pages translated into EOS concepts, i.e. particulars of the university domain.

  Methods:

  The parser methods provide heuristics for identifying patterns in natural language texts, and utilizing the structural patterns defined in the EOS ontology for the university domain. Methods in this context are of the form:

  public static boolean isOccurrence(String target, String universal): decides whether a text fragment target falls into a certain ontological category (universal).

In combination, the Java packages eos.wrapper and eos.graph are defining a simple implementation of the EOS framework.

Chapter 5 Conclusion

This thesis has introduced EOS, a new approach to knowledge representation and processing that is based on *Concept Theory*, a unicategorical formalism for defining *formal ontologies*. Ontologies, as they are used in the ontology engineering community today, are a means for knowledge sharing and reuse. We have shown that they are providing a formalized common understanding of a domain that can be communicated between people and heterogeneous, possibly widely spread, application systems [29]. Domain knowledge in current approaches is foremost identified with a static model on the entities of a particular application area. More advanced systems are providing inference engines that allow for deducing new domain facts. The EOS approach takes on a broader perspective and combines both, the static ontological domain model and the semantics of epistemological processes that operate on this domain model, into the formal framework of Concept Theory.

5.1 Current Scope of EOS

Concept Theory provides the formal foundation of EOS. Its development has been motivated mainly by two fields of research, Web technologies used for formalizing and representing knowledge that have become of vital importance in the Semantic Web community, and logic based approaches for processing knowledge. These considerations have lead to the specific notion of EOS *concepts* underlying Concept Theory. Such concepts are apt to describe domain entities as well as logic statements and processing semantics in knowledge acquisition, generation and retrieval. Generally, we differentiate between two categories of concepts, *universals* and *particulars*. Universals are representing abstract existents, while particulars are describing concrete domain facts that report on the current state of the domain as known to an EOS system.

In this thesis, EOS concepts have been first introduced from a purely ontological perspective, i.e. pertaining to the formalization of domain existents. An ontological concept unambiguously refers to one specific existent and describes its component structure. The ontological view on EOS concepts includes one special kind of concepts with predefined semantics, namely **ISA**. Occurrences of **ISA** define specialization and generalization among EOS concepts. Hence, they delineate the overall specialization hierarchy inherent in the domain model. Graphically speaking, the specialization hierarchy of an EOS ontology defines a leveled DAG, the ontology graph. **ISA** occurrences serve as the edges of the ontology graph, while the remaining set of concepts are the nodes of the graph.

The epistemological perspective of Concept Theory distinguishes more predefined semantics for special classes of concepts. Onto-epistemic **RULE** concepts, along with the logical EOS *conditions* **AND**, **OR** and **NOT** allow for defining constraints on concepts. Such conditions are also used in epistemological concepts, called EOS *laws*, like **ACQUIRE**, **GENERATE** and **QUERY** that are providing a standardized view on the corresponding epistemological procedures. These procedures are part of the EOS framework for knowledge processing systems. On its part, this framework defines a general architecture for EOS systems. An EOS system, finally, is a software tool that is able to interpret EOS concepts and to perform the epistemological procedures for knowledge acquisition, generation and retrieval. This indicates that the theoretical foundation provided by Concept Theory, is the basis of the behavior of an actual EOS system. How such an EOS system can be put to practice has further been elaborated by describing a concrete application scenario for EOS systems that involves classifying Web documents. In this context, practical aspects of realizing EOS knowledge processing have been addressed:

- A graphical notation for EOS ontologies
- A representation language for EOS ontologies based on a specific XML DTD
- Examples for modeling ontological and epistemological concepts in the document classification scenario
- A detailed discussion on semantic query rewriting based on these examples
- A solution for coupling ontology concepts and document markup in Web pages for extracting knowledge from foreign documents
- A relational database schema for storing EOS ontologies in a RDBS
- Indexing and querying techniques for efficient knowledge retrieval from such a RDBS

Considering the scope of the EOS approach as just sketched, this thesis covers a thorough guide to developing ontology-driven knowledge processing systems. EOS provides the theoretical foundation for harmonizing knowledge representation and processing, as well as practical solutions for concretizing and implementing the respective system components, while recurring to well-established standards, such as XML and RDBS technology. Thus, the current scope of EOS demonstrates an at the same time applicable and powerful way of employing formal ontologies with rich semantics. Further research may refine and extend this groundwork.

5.2 The Road Ahead

In this section we will cover interesting perspectives for future research on EOS. Based on the EOS framework as covered in the preceding chapters, we will motivate in Section 5.2.1 how the limitations of the closed world paradigm (cf. Section 3.2.3) can be lifted in EOS systems. Finally, Section 5.2.2 will present three examples for advanced knowledge processing scenarios.

5.2.1 Integration of Particulars under the Open World Paradigm

EOS as presented in this thesis has assumed the closed world paradigm. Under the closed world paradigm, an EOS system will only accept new particulars in knowledge acquisition that conform to its internal ontology. This means that particulars of an unknown component structure will be rejected by the system. An interesting research topic, here, is to adapt the EOS approach to also support the open world paradigm and admit structurally unfamiliar particulars by offering mechanisms to interpret them semantically. This pertains to the epistemological side of an EOS system, which is governed by EOS laws. Particularly, the open world paradigm affects knowledge acquisition and generation, as new particulars have to be read into the system (knowledge acquisition) and then be further processed according to the guidelines provided by **GENERATE** laws (knowledge generation). The EOS framework has been defined in such a way that only minor changes are necessary to allow for particular integration under the open world paradigm. We will shortly sketch a solution for this problem:

- We can ontologically define a concept UNKNOWN, a specialization of the root concept EXISTENT, that acts as an upper category for all concepts with unknown semantics. The knowledge extraction procedure that is providing new concepts will mark a particular it cannot interpret otherwise (as a whole or concerning one or more of its components) according to its EOS ontology, as an occurrence of UNKNOWN.
- The functionality of the knowledge acquisition procedure itself then needs to be changed in only one respect, namely in that it creates a new universal that is an occurrence of UNKNOWN and will act as an immediate parent concept for this particular, bearing the same component structure of the particular. For unknown components of the particular, the respective component of the new parent universal will be of type UNKNOWN, while components the extraction procedure could interpret are related to their ontological categories. In this way, particulars are structurally integrated into the ontology, yet without a clear classification in terms of their semantics.
- Finally, the knowledge generation procedure must include new functionality for occurrences of UNKNOWN. Ontologically, the occurrences of UNKNOWN build a bushy subtree of the ontology graph. The epistemologically interesting task, now, is to group "similar" particulars together by eliminating their immediate parents that are replaced by one single universal, and, analogously, to merge similar universals. This means that the knowledge generation procedure, next to its capacity to create concepts, must be able to delete concepts from the ontology graph (the obsolete universals and **ISA** occurrences). The circumstances that have to be fulfilled for merging ontology concepts can then be defined by **GENERATE** laws, i.e. they are part of the epistemological domain model. Contextual help for this must be available in the ontological part of the domain model. This may comprise, for example, information on which particulars were taken from the same document or Web site, which particulars coincided on the original Web page, etc. Naturally, the categories defining the specific information that is needed to support the formulation of **GENERATE** laws must be an integral part of the EOS ontology, i.e. it is on the part of domain experts to design a suitable domain model. The EOS approach simply provides the features to formulate such a model.

Thus, only minor changes to the epistemological procedures of EOS are necessary to allow for particular integration under the open world paradigm. However, this involves a sophisticated ontology for modeling advanced knowledge processing tasks. The following section will hint on some further aspects that tackle more sophisticated problems in knowledge processing.

5.2.2 Modeling Sophisticated Knowledge Processing Tasks

EOS laws are representing formalized descriptions on how to solve knowledge processing tasks on the basis of ontological knowledge. Some interesting problem areas that require a more sophisticated employment of EOS ontologies, and thus deserve a closer examination in further research projects, are:

- *Vagueness*: The query, '*Find the address of a philosopher living nearby',* contains an inexact, or *vague*, concept: NEARBY. The meaning of NEARBY depends on the context of the query, as there are different notions of closeness in the context of buildings, countries and persons. In such cases, the context needs to be established by respective ontology concepts.
- *Incomplete Knowledge*: When trying to answer a query like the one above, an EOS system may find that it lacks some information crucial for computing the result set. For example, the vague ontology concept NEARBY may not be defined for ADDRESS. This indicates *incomplete knowledge* which prevents the system from successfully calling the knowledge retrieval procedure. However, as the query interface of an EOS system can interpret the exact context of the query, it need not reject the query completely but can start a user interaction asking for the required information. The byproduct of this enhanced service to the end user is that the internal ontology is being enriched with new information about the domain model.
- *Uncertainty*: Not all concepts passed to an EOS system can be regarded as reliable information. For example, the ontology may include concepts provided by untrusted users (e.g. when treating incomplete knowledge), or the knowledge extraction procedure may be mistaken about the ontological categories it determined for concepts it extracted from some information source (e.g. because it is processing natural language texts, or non-native markup). Therefore, the semantics of such concepts are *uncertain*. An ontology universal IS_UNCERTAIN:=(1,(EXISTENT)) could then act as a relation that marks such concepts. Hence, the extraction procedure (or the user interface, respectively) has the possibility to indicate an uncertain concept C it is passing to the EOS system accordingly, i.e. it provides an additional concept C_IS_UNCERTAIN:=(1,(C)). **GENERATE** laws must then be formulated in a way that recognizes this (concepts generated from uncertain concepts may themselves be uncertain).

With these prospects for future research activities we conclude our treatment of the EOS approach to knowledge representation and processing. The different chapters of this thesis have reflected the scope and focus we chose for discussing our contribution to the relatively young but lively and rapidly emerging ontology engineering community. As such, EOS can be seen as one step in an ongoing joint research effort for a better understanding and practical employment of knowledge, the road ahead.

Bibliographic References

[1] S. Abiteboul, P. Buneman, D. Suciu, *Data on the Web. From Relations to Semistructured Data and XML*, Morgan Kaufmann Publishers, San Francisco, USA, 2000.

[2] Aristotle. *Metaphysics*, Book Z. Harvard University Press, Cambridge, MA.

[3] F. Baader et al. *Terminological knowledge representation: A proposal for a terminological logic*. Technical Memo TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1991.

[4] R. Baeza-Yates, B. Ribero-Neto. *Modern Information Retrieval*. Addison Wesley Longman, 1999.

[5] T. Berners-Lee, J. Hendler, O. Lassila. *The Semantic Web*. Scientific American, 2001.

[6] T. Berners-Lee, M. Fischetti, H. Francisco. *Weaving the web: The original design and ultimate destiny of the World Wide Web by its invento*r, 1999.

[7] J. Broekstra, C. Fluit, F. v. Harmlen. *The State of the Art on Representation and Query Languages for Semistructured Data*. Deliverable 8, EU-IST On-To-Knowledge IST-1999-10132, 2000.

[8] R. Brachman, J. Schmolze. *An overview of the KL-ONE knowledge representation system*. Cognitive Science, 9(2):171—216, 1985.

[9] T. Bray. *The Annotated XML Specification*. 1998, http://www.xml.com/axml/testaxml.htm.

[10] T. Buchheim. *Aristoteles*. Herder/Spektrum, Freiburg, Basel, Wien, 1999.

[11] M. Burnyeat, *The Theaetetus of Plato*. Hackett Publishing Company, Indianapolis, 1990.

[12] G. Cantor. *Gesammelte Abhandlungen mathematischen und philosophischen Inhalts*. Springer-Verlag, Berlin, 1932.

[13] V. K. Chaudhri et al. *Open knowledge base connectivity 2.0*. Technical Report KSL-98-06, Knowledge Systems Laboratory, Stanford, 1997.

[14] V. K. Chaudhri et al. OKBC: *A programmatic foundation for knowledge base interoperability*. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pp. 600-607. AAAI Press, 1998.

[15] P. P.-S. Chen. *The entity relationship model – toward a unified view of data*. ACM Press, New York, 1976.

[16] W. J. Clancey. *The Knowledge Level Reinterpreted: Modelling Socio-Technical Systems*. International Journal of Intelligent Systems, 8: 33-49, 1993.

[17] D. Connolly et al. *DAML+OIL (March 2001) Reference Description*. W3C Note 18, 2001, http://www.w3.org/TR/daml+oil-reference.

[18] T. Crane. *Universals*. In *Philosophy 1*, pp 204-213. Oxford University Press, New York, 1999.

[19] R. Davis et al. *What is a Knowledge Representation?*, AAAI, 1993.

[20] L. Dempsey et al. *Specification for resource description methods. Part 1. A review of metadata: a survey of current resource description formats*. DESIRE Project Deliverable, RE 1004, 1997.

[21] M. Dixon. *An Overview of Document Mining Technology*, 1997.

[22] Dublin Core Metadata Initiative, http://dublincore.org/.

[23] D. W. Embley et al. *Ontology-Based Extraction and Structuring of Information from Data-Rich Unstructured Documents*. 1998.

[24] D. W. Embley et al. *Ontology Suitability for Uncertain Extraction of Information from Multi-Record Web Documents*. 1999.

[25] B. Eversberg. *Rules for Formal Cataloging*. http://www.allegro-c.de/formate/rak-0e.htm.

[26] B. Eversberg. *On the Theory of Library Catalogs and Search Engines*. 2002, http://www.allegro-c.de/formate/tlcse.htm.

[27] B. Eversberg et al. *REUSE+, The Part → Whole Relationship in German and American Cataloging Data*, 1998.

[28] B. Eversberg et al. *REUSE, A Contribution to the Enhancement of International Bibliographic Compatibility*, http://www.oclc.cataloging/reuse_project/reuse_final_report.htm.

[29] D. Fensel. *Ontologies*: *A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, New York, 2001.

[30] D. Fensel, S. Decker, M. Erdmann, R. Studer. *Ontobroker: The Very High Idea*. In *Proceedings of the 11th International Flairs Conference (FLAIRS-98)*, Sanibal Island, Florida, USA, 131-135, Mai 1998.

[31] D. Fensel et al. *OIL in a nutshell*. In: *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000)*, R. Dieng et al. (eds.), Lecture Notes in Artificial Intelligence, LNAI, Springer-Verlag, October 2000.

[32] D. Fensel et al. *OnToKnowledge: Ontology-based Tools for Knowledge Management*. In *Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference*, Madrid, Spain, October 2000.

[33] D. Fensel et al. *On2broker: Semantic-Based Access to Information Sources at the WWW.*

[34] M. R. Genesereth. *Knowledge interchange format*. In J. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (KR'91)*. Morgan Kaufmann Publishers, San Francisco, California, 1991.

[35] M.R. Genesereth, R.E. Fikes. *Knowledge interchange format, version 3.0, reference manual*. Technical Report Logic-92-1, Computer Science Dept., Stanford University, 1992.

[36] A. Gomez-Perez, V. R. Benjamins. *Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods*. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden, August 1999.

[37] R. Gömpel, E. Niggemann. *RAK und MAB oder AACR und MARC?, Strategische Überlegungen zu einer – weil immer noch – aktuellen Diskussion.*

[38] A. C. Grayling. *Philosophy, a guide through the subject.* Oxford University Press, Oxford, 1999.

[39] T. R. Gruber. *A translation approach to portable ontologies*. Knowledge Acquisition, 5(2):199-220, 1993.

[40] T. R. Gruber. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. In Formal Ontology in Conceptual Analysis and Knowledge Representation, edited by N. Guarino and R. Poli. Kluwer Academic Publishers, 1993.

[41] N. Guarino. *Formal Ontology, Conceptual Analysis and Knowledge Representation*. International Journal of Human and Computer Studies, special issue on The Role of Formal Ontology in the Information Technology edited by N. Guarino and R. Poli, vol 43 no. 5/6, 1995.

[42] N. Guarino, P. Giaretta. *Ontologies and knowledge bases – towards a terminological clarification*. In N.J. Mars, editor, Towards Very Large Knowledge Bases – Knowledge Building and Knowledge Sharing, 1995, pp 25-32. IOS Press, Amsterdam, 1995.

[43] N. Guarino, C. Masolo, G. Vetere. *OntoSeek: content-based access to the web*. In *IEEE Intelligent Systems*, p. 70-80.

[44] H. Haddouti. *VD17, Cooperative Cataloging in a Scalable Digital Library System*. Ph.D. Thesis, Technische Universität München, 1999.

[45] H. Haddouti, Wolfgang Wohner, Rudolf Bayer. *Towards a Scalable System Architecture in Digital Libraries*. DEXA 1999: 852-861, Florence, Italy, 1999.

[46] A. S. Hornby, S. Wehmeier, ed. *Oxford Advanced Learner's Dictionary of Current English*, Oxford University Press, Oxford, 2000.

[47] I. Horrocks et al. *The Ontology Inference Layer OIL*, http://www.cs.vu.nl/~dieter/oil/Tr/oil.pdf.

[48] P. D. Karp, V. K. Chaudhri, J. Thomere. *XOL: An XML-based ontology exchange language. Version 0.3*, 1999.

[49] M. Kifer, G. Lausen, J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM, 42, 1995.

[50] R. Klemke. *Context Framework – an Open Approach to Enhance Organizational Memory Systems with Context Modelling Techniques*. In *Proceedings of the Third Int. Conf. On Practical Aspects of Knowledge Management (PAKM2000)*, Basel, Switzerland, 30-31 Oct. 2000.

[51] F. M. Lopez, *Overview Of Methodologies For Building Ontologies*. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden, August 1999.

[52] M. J. Loux, *Metaphysics, a contemporary introduction*. Routledge, New York, 1998.

[53] Luke, S., Heflin J. *SHOE 1.01. Proposed Specification*. SHOE Project. February 2000. http://www.cs.umd.edu/projects/plus/SHOE/spec1.01.htm.

[54] S. Luke, L. Spector, D. Rager, J. Hendler. *Ontology-based Web Agents*. In *Proceedings of First International ConferenceonAutonomous Agents*, 1997.

[55] M. Münnich. *REUSE or Rule Harmonization – just a project?*, ALA pre-conference, Washington D.C., 1998.

[56] B. Nebel. *Artificial intelligence: A computational perspective*. In G. Brewka, editor, *Principles of Knowledge Representation, Studies in Logic, Language and Information*. CSLI publications, Stanford, 1996.

[57] B. Omelayenko. *A Survey of Ontology Learning Approaches*,.

[58] B. Omelayenko et al. *Meta Data and UPML*, Deliverable D5, IBROW Project IST-1999-19005, 2000.

[59] M.-F. Plassard, ed. *Functional Requirements for Bibliographic Records*. Final Report, UBCIM Publications - New Series Vol 19, K.G. Saur, München, 1998.

[60] Project page "*Harmonization of Anglo-American Cataloguing Rules and Russian Cataloguing Rules*". http://webdoc.gwdg.de/ebook/aw/reuse/harmony

[61] Project page "*Project REUSE: Aligning International Cataloging Standards*". http://webdoc.gwdg.de/ebook/aw/reuse

[62] D. Raggett et al. *HTML 4.01 Specification*. W3C Recommendation, 1999, http://www.w3.org/TR/html4/

[63] Report on the project " *Harmonization of Anglo-American Cataloguing Rules and Russian Cataloguing Rules*". http://webdoc.gwdg.de/ebook/aw/reuse/harmony/harmony_report1.htm

[64] Report on the project "*Retrokonversion – Konversion von Zettelkatalogen in deutschen Hochschulbibliotheken. Methoden, Verfahren, Kosten*". Berlin 1993, Deutsches Bibliotheksinstitut (dbi-Materialien 128)

[65] Report on the project "*UseMARCON – User controlled generic MARC converter*". http://www.kb.nl/kb/sbo/bibinfra/usema-en.html

[66] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, Boston, 1998.

[67] B. Smith. *Ontology: Philosophical and Computational*, 2000, http://wings.buffalo.edu/philosophy/faculty/smith/articles/ontologies.html.

[68] B. C. Smith. *Reflection and Semantics in a Procedural Language*. PHD Thesis. MIT Laboratory for Computer Science. 1982.

[69] J. F. Sowa. *Knowledge Representation. Logical, Philosophical and Computational Foundations*. Brooks/Cole, Pacific Grove, USA, 2000.

[70] S. Sturgeon, M.G.F. Martin, A.C. Grayling. *Epistemology*. In *Philosophy 1*, pp 7-60. Oxford University Press, New York, 1999.

[71] S. E. Thomas. *Kooperation der Library of Congress mit deutschen Bibliotheken im Erschließungsbereich*. In *86. Deutscher Bibliothekartag in Erlangen 1996 – Ressourcen nutzen für neue Aufgaben*, pp. 266-272.

[72] M. Uschold, M. Grüninger. *Ontologies: Principles, Methods and Applications*, In *Knowledge Engineering Review*, Volume 11 Number 2, 1996.

[73] L. Vieille. *From Data Independence to Knowledge Independence: An on-going Story*. In *Proceedings of the 24th VLDB Conference,* New York, USA, 1998.

[74] S. Weibel, J. Miller, R. Daniel. *OCLC/NCSA metadata workshop report*. OCLC, 1995, http://www.oclc.org:5046/conferences/metadata/dublin_core_report.html.

[75] W. Wohner. *A Modest Proposal: Reasoning Beyond the Limits of Ontologies*. In Proceedings of IJCAI-01 Workshop on Ontologies and Information Sharing, Seattle, Washington, August 4 - 5, 2001.

[76] W. Wohner. *EOS: Making the Epistemic Impact of Ontologies in Knowledge Processing Explicit*. In Proceedings of KI-2001 Workshop on "Ontologies", Vienna, Austria, September 18, 2001.

[77] W3C. *Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000*. T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, editors, 2000. http://www.w3.org/TR/REC-xml.

[78] M. M. Yee, S. Shatford-Layne. *Improving Online Public Access Catalogs*. ALA, 1998.

[79] M. H. Zack. *Managing Codified Knowledge*. Sloan Management Review, Volume 40, Number 4, pp. 45-58, 1999.

Appendix: Indexes and Tables

Table of Figures

Table of Definitions

Table of Proofs