16

Description Logics for Data Bases

Alex Borgida

Maurizio Lenzerini

Riccardo Rosati

Abstract

In contrast to the relatively complex information that can be expressed in DL ABoxes (which we might call knowledge/information), databases and other sources such as files, semi-structured data, and the World Wide Web provide rather simpler *data*, which must however be managed effectively. This chapter surveys the major classes of application of Description Logics and their reasoning facilities to the issues of data management, including: (i) expressing the conceptual domain model/ontology of the data source, (ii) integrating multiple data sources, and (iii) expressing and evaluating queries. In each case we utilize the standard properties of DLs, such as the ability to express ontologies at a level closer to that of human conceptualization (e.g., representing conceptual schemas), determining consistency of descriptions (e.g., determining if a query or the integration of some schemas is consistent), and automatically classifying descriptions that are definitions (e.g., queries are really definitions, so we can classify them and determine subsumption between them).

16.1 Introduction

According to [ElMasri and Navathe, 1994], a database is a coherent collection of related data, which have some "inherent meaning". Databases are similar to knowledge bases because they are usually used to maintain *models* of some domain of discourse (UofD). Of course, the purpose of such computer models is to support end-users in finding out things about the world, and therefore it is important to maintain an up-to-date and error-free model. The main difference between data and knowledge bases is that while the former concentrate on manipulating *large and persistent* models of relatively simple data, the latter provide more support for *inference*—finding answers about the model which had not been explicitly told to it—and involve fewer but more complex data.

Following the functional view of Knowledge Bases advocated by Levesque, we expect a number of operations that can be applied to the KB, such as `define`, `tell`, and **ask**. Each of these operations involves one or more languages, such as the schema/constraint language, the update language, the query language and the answer language. In an earlier paper surveying the application of DLs to data management [Borgida, 1995], it has been argued that DLs offer advantages for each of these languages, as well as the internal processing of queries.

We begin by providing a review of the important notions involving databases, their development and use, as preparation for examining the application of DLs in these tasks.

First, one needs to describe the UofD about which the database will be knowledgeable. This is a form of requirements specification, which is normally undertaken using some high-level language, because the requirements will have to be understandable both to end-users and implementors, so they can agree on the goals. In databases, the best known such language is the Entity-Relationship (ER) data model[1], but many other so-called *semantic modeling languages* have been proposed [Hull and King, 1987]. The ER data model will be described in considerable detail and precision in Section 16.2; for now, suffice it to say that it views the world as populated by entities, which are related to each other by n-ary relationships, and are described by attributes having atomic values. Note that a semantic model may be concerned with the universe of discourse as well as the data to be stored in the computer, and consists of mostly time-invariant generic information (e.g., "every department has exactly one manager") as opposed to specific facts (e.g., "Edna manages the shipping department".) The semantic model introduces the terms to be used in talking about the domain, and captures their meaning by their inter-relationships and constraints on them.

From this generic description of the UofD, the database designer develops a *logical schema*, describing the structure of data stored in the database, including the data types, interconnections, and constraints that must hold. Different data models are used for this purpose, but the *relational data model* has become the logical model of choice. While in the semantic modeling phase the emphasis was on a natural and direct mapping to the UofD, in this case the driving force is the existence of large software systems called Database Management Systems (DBMS), which support the management of the data in the model. For example, the relational data model views data as being stored in the form of tables/relations, with rows/tuples containing primitive data types (e.g., integers, strings). In this case, the schema contains, among others, the name of each table, with its columns and their datatype. For example, table Supplies may have columns for the material, the supplier, the recip-

[1] The term "data model" refers to a language or set of concepts for describing a class of databases.

ient, as well as the shipment date and the amount of material supplied. Relational DBMS require that each table be given a subset of attributes (called a "key") which uniquely identifies each tuple. DBMS may offer additional ways to capture integrity constraints—assertions distinguishing valid from invalid states of the data.

More recently, *Object Oriented DBMS* have been developed. These support the management of persistent objects with intrinsic identity, which can be related to (collections of) other objects, not just atomic values. Such OO-DBMS can be used, among others, for providing persistence for object-oriented languages. Object-oriented languages and databases also support the notion of "method"/procedure attached to a class, as well as implementation encapsulation, but these aspects will not be considered in this chapter.

The database is used of course to store facts about the (current) state of the world. Databases make the so-called "closed world assumption", which states that a fact is false unless it has been explicitly stated as true. This assumption works well with the restriction that the database represents only a very limited form of partial information. In particular, databases do not allow the representation of disjunctive information, and support only a very limited form of existential quantification: if there is no information about an attribute, it is given the *null* value.

In order to provide access to the data stored in databases, DBMS support a variety of *query languages*—languages for specifying declaratively what data is to be retrieved. For relational databases, SQL is the practical query language of choice. However, from the theoretical point of view, First Order Logic formulas with free variables are a much more elegant form, based on the observation that tables can be viewed as predicates. For example,

$$\exists m, d1, d2.\, \mathsf{supplies}('\texttt{intel}', r, m, d1) \wedge \mathsf{supplies}('\texttt{intel}', r, m, d2) \wedge (d2 \neq d1)$$

would be asking for recipients (values of the free variable r), who had received from '`intel`' shipments of the same material (m) on different dates ($d1, d2$).

Query languages of varying expressive power can be obtained by restricting or extending the above "standard". For example, the so-called "conjunctive" or "select-join-project" queries only allow formulas with existential quantifiers and conjunction, while Datalog is a query language that permits the use of intermediate tables derived using Horn rules, and thereby supports recursion [Ullman, 1988]. For example, if we want to describe when one company depends on another through a chain of suppliers, we could state the rules[1]

$$\begin{aligned} \mathsf{dependsOn}(x, y) &\leftarrow \mathsf{supplies}(x, y, m, d). \\ \mathsf{dependsOn}(x, y) &\leftarrow \mathsf{supplies}(x, z, m, d, a) \wedge \mathsf{dependsOn}(z, y, m_2, d_2, a_2). \end{aligned}$$

[1] Variables appearing only on the right hand side of "$\rightarrow$" are assumed to be existentially quantified.

In many DBMS, the result of a query is another structure of the kind found in the schema (e.g., relational queries return as answer tables). In some situations, either because a query is asked frequently or because we want to restrict the access of some users to a subset of the database, a query can be named, in which case it is called a *view*. If a view is *materialized*, then its value is stored rather than recomputed on demand, and it is kept correct after every update to the basic database.

The DBMS performs a number of hidden functions, insulating users from the considerable details of the *physical* level. For example, the DBMS places physically the incoming data onto storage media, and provides data structures and other information that permits efficient access of certain data at some later point of time. In particular, given a query, the DBMS attempts to optimize the time in which it is answered by looking at access structures available, statistical information and using the ability to reformulate queries into other, equivalent ones.

Over time, additional, more complex kinds of databases and DBMS have appeared. For example, *distributed databases* keep information at a variety of sites connected by networks (e.g., so that data might be closer to where it is used most frequently). Note however that the user is unaware of this detail, and perceives a single database. Heterogeneous and federated databases are collections of independent databases which choose to share information but are maintained autonomously. In the extreme, users may be interested in obtaining information from all kinds of sources, including non-databases such as files, etc. In such situations, a significant problem is relating the logical schemas at the various sites in order to provide a schema that can be presented to the user. The rest of the chapter is devoted to showing a variety of roles that DLs (and reasoning with them) can play in database management. In particular, in Section 16.2 we take a detailed look at their use in semantic/conceptual modeling. We then examine the possible uses of DLs in querying and query processing in Section 16.3, while in Section 16.4 we will consider the utility of DLs in providing integrated access to multiple information sources. We summarize the material in Section 16.5.

16.2 Data models and Description Logics

Recall that a "data model" is essentially a language or set of concepts for describing a class of certain kinds of databases. This section attempts to answer some questions about the relationship between data models and DLs:

What are some examples of such relationships? First, we will consider in detail the translation of Entity-Relationship models into knowledge bases expressed in the $\mathcal{DLR}$ description logic. In Section 16.2.5, we will consider

more cursorily several other data models, such as OODB and semistructured data.

How are relationships established? The answer is (i) formalizing the data model (ER in this case), (ii) choosing an appropriate DL ($\mathcal{DLR}$ in this case), (iii) defining a translation function from the former to the latter, and (iv) proving that this translation is "information preserving" (not done here, but detailed in [Calvanese *et al.*, 1999e]).

What benefits can be derived from having established relationships? Most significant is the use of automated DL reasoning services to support the development and maintenance of correct models (Section 16.2.4). In addition, since DLs are often more expressive, it is possible to suggest extensions to database data models that allow further information about the structure of the data to be captured (Section 16.2.3).

16.2.1 The Entity-Relationship model

In order to talk about the relationship between the Entity-Relationship (ER) model and DLs, it is necessary first to introduce the reader to the ER data model (see also Chapter 10). ER is the most widespread semantic data model, and it has become a standard, extensively used in the design phase of commercial applications. The ER Model was introduced in [Chen, 1976], with minor variants and extensions proposed over the years (e.g., [Teorey, 1989; Batini *et al.*, 1992; Thalheim, 1992; 1993]).

The basic elements of the ER Model are entities, relationships, and attributes. An *entity set* (or simply *entity*) denotes a set of objects, called its *instances*, that have common properties. Elementary properties are modeled through *attributes*, whose values belong to one of several predefined domains, such as Integer, String, or Boolean. Properties that are due to relations to other entities are modeled through the participation of the entity in relationships. A *relationship set* (or simply *relation*) denotes a set of tuples (also called its instances), each of which represents an association among a different combination of instances of the entities that participate in the relationship. Since each entity can participate in a relationship more than once (e.g., a company can be the recipient or sender in a "supply" relationship), the notion of *ER-role* is introduced to represent such a participation, and to which a distinguishing identifier within the relationship is assigned. The *arity* of a relationship is the number of its ER-roles. We assume that, for each relationship of arity n, the identifiers $1, \ldots, n$ are assigned to the roles of the relationship.

An entity B is said to be a specialization/IS-A of another entity A, if all the instances of B are also instances of A. Relationships can be similarly related by IS-A. This induces an inheritance of the attributes of an entity to its sub-entities,

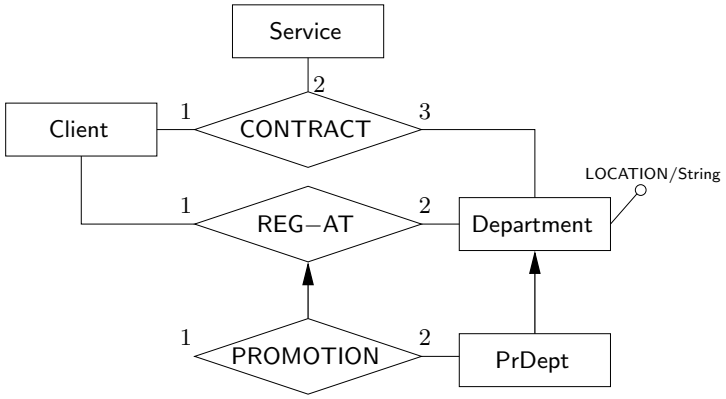

Fig. 16.1. Example of an ER schema.

and of the roles of a relationship to its sub-roles. The ER schema produced as a result of ER modeling is usually represented in a graphical notation, which is particularly useful for an easy visualization of the data dependencies. In the commonly accepted notation, entities are represented as boxes, whereas relationships are represented as diamonds. An attribute is shown as a circle attached to the entity for which it is defined. ER-roles are graphically depicted by connecting the relationship to the participating entities, and labeling the edges with the corresponding role identifier. An IS-A relation between two entities is denoted by an arrow from the more specific to the more general entity (analogously for IS-A relations between two relationships). *Cardinality constraints* can be attached to an ER-role in order to restrict the number of times each instance of an entity is allowed to participate via that ER-role in instances of the relationship.

Such constraints can be used to specify both existence dependencies and functionality of relations [Cosmadakis and Kanellakis, 1986]. They are often used only in a restricted form, where the minimum cardinality is either 0 or 1 and the maximum cardinality is either 1 or ∞. Cardinality constraints in the form considered here have been introduced already in [Abrial, 1974], and subsequently studied in [Grant and Minker, 1984; Lenzerini and Nobili, 1990; Ferg, 1991; Ye *et al.*, 1994; Thalheim, 1992; Calvanese and Lenzerini, 1994b].

An example of an ER schema is reported in Figure 16.1. Such a schema models information, handled by an enterprise, about contracts between customers and departments for services, and about registration of customers at departments. Some customers may be registered at "promotion departments".

For the purpose of relating the ER Model to DLs it is better to have a more formal description, which also abstracts out the most important common characteristics present in the different variants.

An *ER schema* $\mathcal{S}$ is constructed starting from pairwise disjoint sets of entity symbols, relationship symbols, ER-role symbols, attribute symbols, and domain symbols. Each domain symbol D has an associated predefined basic domain $D^{\mathcal{B}_\mathcal{D}}$, and we assume the basic domains to be pairwise disjoint. For each entity symbol, a set of attribute symbols is defined, and to each such attribute a unique domain symbol is associated. A relationship symbol of arity n has n associated ER-role symbols, each with an associated entity symbol, and defines a relationship between these entities. We assume that each ER-role symbol belongs to a unique relationship, thus determining also a unique entity. The cardinality constraints are represented by two functions $cmin_\mathcal{S}$, from ER-role symbols to nonnegative integers, and $cmax_\mathcal{S}$, from ER-role symbols to positive integers union the special symbol ∞. IS-A relations between entities and between relationships are modeled by means of a binary relation $\preceq_\mathcal{S}$. We do not need to make any special assumption on the form of $\preceq_\mathcal{S}$, such as acyclicity or injectivity.

The semantics of an ER schema can be given by specifying which database states are consistent with the information structure represented by the schema. Formally, a database state $\mathcal{B}$ corresponding to an ER schema $\mathcal{S}$ is constituted by a nonempty *finite* set $\Delta^\mathcal{B}$, assumed to be disjoint from all basic domains, and a function $\cdot^\mathcal{B}$ that maps

- every domain symbol D to the corresponding basic domain $D^{\mathcal{B}_\mathcal{D}}$,
- every entity E to a subset $E^\mathcal{B}$ of $\Delta^\mathcal{B}$,
- every attribute A to a set $A^\mathcal{B} \subseteq \Delta^\mathcal{B} \times \bigcup_{D \in \mathcal{D}_\mathcal{S}} D^{\mathcal{B}_\mathcal{D}}$, and
- every relationship R to a set $R^\mathcal{B}$ of labeled tuples over $\Delta^\mathcal{B}$.

A *labeled tuple* over a domain $\Delta^\mathcal{B}$ is a function from a set of ER-roles to $\Delta^\mathcal{B}$. The labeled tuple T that maps ER-role U_i to o_i, for $i \in \{1, \ldots, n\}$, is denoted $\langle U_1 \colon o_1, \ldots, U_n \colon o_n \rangle$. We also write $T[U_i]$ to denote o_i, and call it the *U_i-component of* T. The elements of $E^\mathcal{B}$, $A^\mathcal{B}$, and $R^\mathcal{B}$ are called *instances* of E, A, and R respectively.

A database state is considered acceptable if it satisfies all integrity constraints that are part of the schema. This is captured by the notion of legal database state. A database state $\mathcal{B}$ is *legal for* an ER schema $\mathcal{S}$, if it satisfies the following conditions:

- For each pair of entities E_1, E_2 with $E_1 \preceq_\mathcal{S} E_2$, it holds that $E_1^\mathcal{B} \subseteq E_2^\mathcal{B}$.
- For each pair of relationships R_1, R_2 with $R_1 \preceq_\mathcal{S} R_2$, it holds that $R_1^\mathcal{B} \subseteq R_2^\mathcal{B}$.
- For each entity E, if E has an attribute A with domain D, then for each instance $e \in E^\mathcal{B}$ there is exactly one element $a \in A^\mathcal{B}$ with e as first component, and the second component of a is an element of $D^{\mathcal{B}_\mathcal{D}}$.
- For each relationship R of arity n between entities $E_1, \ldots, E_n$, to which R is connected by means of ER-roles $U_1, \ldots, U_n$ respectively, all instances of R are of the form $\langle U_1 \colon e_1, \ldots, U_n \colon e_n \rangle$, where $e_i \in E_i^\mathcal{B}$, $i \in \{1, \ldots, n\}$.

- For each ER-role U of relationship R associated with entity E, and for each instance e of E, it holds that

$$cmin_{\mathcal{S}}(U) \leq |\{r \in R^{\mathcal{B}} \mid r[U] = e\}| \leq cmax_{\mathcal{S}}(U).$$

16.2.2 Transforming Entity-Relationship schemas into $\mathcal{DLR}$ knowledge bases

In order to represent ER Schemas in terms of Description Logics knowledge bases, we make use of the DL $\mathcal{DLR}$, which has been formally introduced in Chapter 5. We recall here the syntax of $\mathcal{DLR}$, which is a natural generalization of Description Logics towards n-ary relations: in particular, atomic relations, of given arity between 2 and n_{max}, belong to the basic elements of $\mathcal{DLR}$, and, besides concept expressions, arbitrary relation expressions can be formed, according to the following syntax:

$$\begin{array}{rcl} \mathbf{R} & := & \top_n \mid \mathbf{P} \mid (\$i/n{:}\, C) \mid \neg\mathbf{R} \mid \mathbf{R}_1 \sqcap \mathbf{R}_2 \\ C & := & \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid \exists[\$i]\mathbf{R} \mid \leqslant k\,[\$i]\mathbf{R} \end{array}$$

where $\mathbf{P}$ and $\mathbf{R}$ denote respectively atomic and arbitrary relations, i and j denote components of relations, i.e., integers between 1 and n_{max}, n denotes the arity of a relation, i.e., an integer between 2 and n_{max}, and k denotes a nonnegative integer. In what follows, we abbreviate $(\$i/n{:}\, C)$ with $(\$i{:}\, C)$ when n is clear from the context. Moreover, we use the following abbreviations:

$$\begin{array}{rcl} \forall[\$i]\mathbf{R} & \text{for} & \neg\exists[\$i]\neg\mathbf{R}, \\ \geqslant (k+1)\,[\$i]\mathbf{R} & \text{for} & \neg(\leqslant k\,[\$i]\mathbf{R}), \\ = k\,[\$i]\mathbf{R} & \text{for} & (\leqslant (k+1)\,[\$i]\mathbf{R}) \sqcap (\geqslant k\,[\$i]\mathbf{R}). \end{array}$$

In $\mathcal{DLR}$, n-ary relations are interpreted as sets of tuples of arity n, and the $\mathcal{DLR}$ constructs generalize those of traditional DLs. In particular, besides the Boolean constructs on concepts and relations, the construct $(\$i/n{:}\, C)$ denotes all tuples of arity n in which the i-the component is an instance of concept C, and thus represents a unary selection. The construct $\exists[\$i]\mathbf{R}$, denotes all objects that participate as i-th component in an tuple of relation $\mathbf{R}$, and thus represents a unary projection. Finally $\leqslant k\,[\$i]\mathbf{R}$ is a generalization of number restrictions to n-ary relations. We refer to Chapter 5, Section 5.7, for the formal semantics of the $\mathcal{DLR}$ constructs.

We now show that the semantics of the ER Model can be captured in $\mathcal{DLR}$ by defining a translation ϕ from ER schemas to $\mathcal{DLR}$ knowledge bases, and then establishing a correspondence between legal database states and models of the derived knowledge base. In the following, for each relationship R of arity n in $\mathcal{S}$, we denote with μ_R a mapping from the set of ER-roles associated with R to the integers $1, \ldots, n$.

The knowledge base $\phi(\mathcal{S})$ derived from an ER schema $\mathcal{S}$ is defined as follows:

- The set of atomic concepts of $\phi(\mathcal{S})$ consists of the set of entity and domain symbols in $\mathcal{S}$.[1]
- The set of atomic relations of $\phi(\mathcal{S})$ is obtained from the set of relationship and attribute symbols in $\mathcal{S}$. More specifically:
  - each symbol R in $\mathcal{S}$, denoting a relationship of arity n, is mapped into a symbol $\mathbf{P}_R$ in $\phi(\mathcal{S})$, denoting a relation of arity n.
  - each attribute symbol A in $\mathcal{S}$ is mapped into a symbol $\mathbf{P}_A$ in $\phi(\mathcal{S})$, denoting a relation of arity 2. Thus, each instance of the relation $\mathbf{P}_A$ is a tuple such that its first component corresponds to an entity, while the second component denotes an element of the concept corresponding to the attribute domain.
- The set of inclusion axioms of $\phi(\mathcal{S})$ consists of the following elements:
  - For each pair of entities E_1, E_2 such that $E_1 \preceq_{\mathcal{S}} E_2$, the inclusion axiom

$$E_1 \sqsubseteq E_2$$

  - For each pair of relationships R_1, R_2 such that $R_1 \preceq_{\mathcal{S}} R_2$, the inclusion axiom

$$\mathbf{P}_{R_1} \sqsubseteq \mathbf{P}_{R_2}$$

  - For each attribute A with domain D of an entity E, the inclusion axiom

$$E \sqsubseteq (\forall[\$1](\mathbf{P}_A \sqcap (\$2{:}\, D))) \sqcap = 1\,[\$1]\mathbf{P}_A$$

  - For each relationship R of arity n with ER-roles $U_1, \ldots, U_n$ in which each U_i is associated with the entity E_i, the inclusion axiom

$$\mathbf{P}_R \sqsubseteq (\$\mu_R(U_1){:}\, E_1) \sqcap \cdots \sqcap (\$\mu_R(U_n){:}\, E_n)$$

  - For each ER-role U of relationship R associated with entity E, with cardinality constraints $m = cmin_{\mathcal{S}}(U)$ and $n = cmax_{\mathcal{S}}(U)$,
    - if $m \neq 0$, the inclusion axiom

$$E \sqsubseteq \geqslant m\,[\$\mu_R(U)]\mathbf{P}_R$$

    - if $n \neq \infty$, the inclusion axiom

$$E \sqsubseteq \leqslant n\,[\$\mu_R(U)]\mathbf{P}_R$$

Based on the results presented in [Calvanese *et al.*, 1999e], the correctness of the translation presented above can be formally proved. More specifically, let $\mathcal{S}$ be an ER schema. Then, there is a one-to-one correspondence between legal database states of $\mathcal{S}$ and models of the $\mathcal{DLR}$ knowledge base $\phi(\mathcal{S})$. For example, an entity

[1] For the sake of simplicity, we model domains of ER schemas as concepts in $\mathcal{DLR}$.

E can be populated in a legal database state for $\mathcal{S}$ if and only if $\phi(\mathcal{S})$ admits a model in which E has a non-empty extension. This allows us to exploit reasoning techniques developed for the logic $\mathcal{DLR}$ in order to reason on ER schemas.

For example, by applying the translation presented above to the ER schema in Figure 16.1, presented earlier, we obtain the following $\mathcal{DLR}$ knowledge base:

$$\begin{aligned}
\mathsf{CONTRACT} &\sqsubseteq (\$1\colon \mathsf{Client}) \sqcap (\$2\colon \mathsf{Service}) \sqcap (\$3\colon \mathsf{Department}) \\
\mathsf{REG{-}AT} &\sqsubseteq (\$1\colon \mathsf{Client}) \sqcap (\$2\colon \mathsf{Department}) \\
\mathsf{PROMOTION} &\sqsubseteq \mathsf{REG{-}AT} \sqcap (\$2\colon \mathsf{PrDept}) \\
\mathsf{Department} &\sqsubseteq \forall[\$1](\mathsf{LOCATION} \sqcap (\$2\colon \mathsf{String})) \sqcap = 1\,[\$1]\mathsf{LOCATION} \\
\mathsf{PrDept} &\sqsubseteq \mathsf{Department}
\end{aligned}$$

16.2.3 Additions to the Entity-Relationship model

The ER Model does not provide several features which would prove useful in order to represent complex dependencies between data. On the other hand, the richness of constructs that is typical of Description Logics, and the correspondence between the two formalisms established in the previous section, makes it possible to add such constructs to the basic model and take them fully into account when reasoning on a schema. We provide several examples of useful additions to the basic ER Model that arise as a natural consequence of the correspondence with the Description Logic $\mathcal{DLR}$. We also consider a feature of the original ER Model that appears to force $\mathcal{DLR}$ itself to be extended.

- *Arbitrary Boolean constructs on entities.* The only direct relationship between entities that can be expressed in the basic ER Model is the IS-A relation. A common extension is by so called *generalization hierarchies* (see e.g., [Batini *et al.*, 1992]), which allow one to express that the extension of an entity should be the disjoint union of the extensions of other entities. Such construct can easily be translated by making use of union and negation of $\mathcal{DLR}$.
- *Refinement of properties along an IS-A hierarchy.* Another important extension that should be considered is the possibility to specify more complex forms of refinement of properties of entities along IS-A hierarchies, than the mere addition of attributes. This is already an essential feature of the more recent object-oriented models. In particular, cardinality constraints could be refined by restricting the range of values, and the participation in relationships can be restricted. One may require for specific instances of an entity that the objects they are related to via a certain relationship belong to a more specific entity than the one directly associated to the ER-role. Such forms of constraints can be naturally expressed in $\mathcal{DLR}$ by making use of universal quantification over relations.

- *Definitions of classes by means of complex properties.* In the ER Model (and more generally in Semantic Data Models) one can specify only necessary conditions that the instances of entities (or more generally classes) must satisfy. This means that in a database that conforms to the schema one cannot deduce that a certain object is an instance of an entity unless this fact is explicitly stated. When modeling a complex domain, however, in order to capture more precisely the intended semantics, one would like to be able to define classes of objects through necessary and sufficient conditions, or even to state just sufficient conditions for an object to be an instance of a class. The former correspond in fact to *views*, which are important parts of database schemas. By using the different types of axioms of $\mathcal{DLR}$, necessary and sufficient (and even just sufficient) conditions can be easily imposed and become part of the schema.
- *Key constraints.* Because of their utility in physical database design, even the original ER Model allowed the specification of key attributes/roles. Extending DLs with key constraints (roles which uniquely identify objects) has been the subject of several investigations [Borgida and Weddell, 1997]. In particular, Calvanese *et al.* [2000b] have shown that reasoning about $\mathcal{DLR}$ augmented by key constraints can be performed without increasing the worst-case computational complexity.
- *Temporal constraints.* Recent efforts in the Conceptual Modeling community have been devoted to properly capturing time-varying information, and several proposals of temporally enhanced Entity-Relationship (ER) exist. [Artale and Franconi, 1999; 2001; Artale *et al.*, 2001] provide a DL-based logical formalization of the various properties that characterize and extend different temporal ER models which are found in literature. In particular, [Artale *et al.*, 2001] define the DL $\mathcal{DLR}_{\mathcal{US}}$, an extension of $\mathcal{DLR}$ with temporal constructs, and study decidability and complexity of reasoning in such a logic.

16.2.4 Reasoning about Entity-Relationship schemas

Providing a formalization of the ER schema in terms of the logic $\mathcal{DLR}$ allows for supporting several forms of reasoning on the ER schema. Typical reasoning tasks at the conceptual level supporting the designer of an ER schema $\mathcal{S}$ (see [Calvanese *et al.*, 1998e]) include:

- *Entity satisfiability*, i.e., whether for every concept C, $\mathcal{S}$ admits a model in which it has a nonempty extension. If C must always have an empty extension then there is an inconsistency in its specification, or at the very least the concept is inappropriately named since it is a synonym for "`EmptyEntity`".

- *Relation satisfiability*, i.e., whether $\mathcal{S}$ admits a model in which a certain relation has a nonempty extension. (Similar to the above.)
- *Consistency of the ER schema*, i.e., whether $\mathcal{S}$ admits a finite model. Without this, there is no database that satisfies the schema, which indicates that the totality of the definitions is inconsistent or requires an infinite model, which is a clear sign of incorrectness. Ideally, the reasoning system could provide *explanations* [McGuinness and Borgida, 1995; Borgida *et al.*, 2000] for the source of inconsistencies, which could focus the search for modifications.
- *Redundancy of the ER schema.* Various forms of redundancy in the ER schema can be detected: e.g., if A, B are entities and both $A \sqsubseteq B$ and $B \sqsubseteq A$ hold, we can conclude that one of the entities is redundant.
- *Stronger constraints on relationship roles.* The concept and relationship specifications may combine to yield stronger cardinality or domain constraints than those explicitly specified by the designer. (The simplest example is when we permit (multiple) inheritance.)
- *Entity subsumption*, i.e., whether the extension of one concept B is a subset of the extension of another concept A in every model of $\mathcal{S}$. This property suggests that the designer check for the possible omission of an explicit IS-A relationship between B and A. Alternatively, if conceptually all B's are not supposed to be A's, then something is wrong in the rest of the schema, since it is forcing an undesired conclusion.
- *Relation subsumption*, i.e., whether the extension of one relation is a subset of the extension of another relation in every model of $\mathcal{S}$. (Similar to the above.)

Ideas such as the ones above have been been pursued, for example, within the DWQ European Project [Bouzeghoub *et al.*, 1999], where the DL system FaCT [Horrocks, 1998b] has been successfully used as reasoning tool supporting the analysis and the integration of diverse database conceptual schemas [Franconi and Ng, 2000].

16.2.5 Description Logics and other data models

Several other investigations have been carried out on the relationships between DLs and database models:

- [Bergamaschi and Nebel, 1994; Artale *et al.*, 1996a; Calvanese *et al.*, 1999e] provide formal models of object-oriented DBMSs using DLs.
- [Borgida *et al.*, 1989; Beck *et al.*, 1989; Bergamaschi and Sartori, 1992] introduce semantic data models based directly on DLs, which are different from ER and previous database semantic data models.

- More generally, class-based knowledge representation schemes, such as semantic networks, conceptual structures and frames [Lehmann, 1992; Sowa, 1991] have been considered as database models, or as ways to enrich the deductive capabilities of data models. These are related to DLs as suggested in Chapter 4.

A recent important development in the field of data management has been the need to represent data whose structure is less rigid and strict than that held in conventional databases. Such *semistructured data* are important in many application areas, such as web information systems, biological databases, and digital libraries. Semistructured data is neither raw text, nor strictly typed as in conventional database systems [Abiteboul, 1997]. In many recent formalisms, semistructured data is modeled by graphs with labeled edges, where the label keeps information on both the values and the schema of the data. Many authors have noticed that this model coincides with the ontology of DLs, where roles correspond to edges. In [Calvanese *et al.*, 1998c] it is shown that expressive DLs can not only capture semistructured data schemas, but can also add the ability to express several new kinds of constraints. The same kind of investigation has been carried out in [Calvanese *et al.*, 1999d] for the case of the XML language, which is currently a very popular formalism for semistructured data on the web (see Chapter 4, Section 4.3.3 for more details).

16.3 Description Logics and database querying

We have seen that descriptions can be used to present the schema of a database. For example, to emulate object-oriented databases, classes are equated with primitive concepts, while type restrictions on attributes are presented as necessary conditions that apply to these primitive classes in the form of role restrictions. In addition, certain integrity constraints can be expressed as rules of the form "if C then D", or axioms $C \sqsubseteq D$. On the other hand, since a concept description provides *necessary and sufficient* conditions for objects to satisfy it, it is natural to treat it as a query. So, in systems like CLASSIC [Borgida *et al.*, 1989] and CANDIDE [Beck *et al.*, 1989], we have a unification of two traditionally distinct languages: the data definition and data manipulation languages.

16.3.1 Description Logics as query languages

Once the query is viewed as a concept description, we can perform the standard operations on it. For example, the query description can be compared to the inconsistent description. If they are equivalent, this is almost surely a mistake on the part of the user—who would want to ask a query that never returns an object? The most likely reason for this is that the person asking the query is un-

familiar with the application domain. Since the query can be quite complex, and the schema quite large, a really helpful system would then assist the user in understanding the problem by isolating the specific parts of the query and of the schema that are responsible for the contradiction. Such a tool can be built on top of explanation facilities available for certain DLs [McGuinness and Borgida, 1995; Borgida *et al.*, 2000].

More generally, in situations where the query returns no individuals in the current database, it has been argued that the query is "not interesting", and should be generalized until a non-empty answer set is returned. As suggested by Anwar *et al.* [1992], this relaxation can be performed using the semi-lattice of descriptions provided by the subsumption relationship, which can guide the systematic weakening of terms in the query.

The query can be classified with respect to the concepts in the schema. This can be used to help users pose queries in an unfamiliar domain, as follows: if the answer set contains unwanted values, the immediate subsumers and subsumees of the query reveal other *potentially relevant* concepts, and through subsumption assertions in the schema, roles as well, which the user may want to restrict in stating the query. The result is a process of *query specification by iterative refinement* introduced by Tou *et al.* [1982].

Queries can also be classified with respect to each other into a subsumption hierarchy. In an environment where several people are asking exploratory questions about the data over a long period of time (e.g., data mining by humans), it is very useful to have the questions organized so that the results of *previous* related queries can be reviewed [Brachman *et al.*, 1992]. This prevents duplication of effort and, again, helps the user to pose queries that are more precise.

Unfortunately, in exchange for a more expressive description of the schema, DLs pay the price of a weaker than usual query language: queries can only return subsets of existing objects, rather than creating new objects (as in standard SQL databases); furthermore, the selection conditions are rather limited. In fact, it has been shown [Borgida, 1996] that even the most expressive DLs discussed in the literature until recently, could only express a variant of the "3-variable" subset of formulas of First Order Logic—i.e., formulas that only use 3 variables, although allowing numeric quantifiers, like "exists at least n".

Given the expressive limitations of DL concepts alone as queries, it is reasonable to consider extending standard queries (in Datalog) with DLs. Two different approaches have been pursued: In one, inspired by the work of Aït-Kaci and Nasr [1986] on LOGIN, and exemplified by the $\mathcal{AL}$-LOG language [Donini *et al.*, 1998b], descriptions are used essentially as *type constraints* on variables appearing in Horn clauses. In this case, a crucial condition is that concept and

role names form a disjoint set from the relations used in expressing rules. The second approach, exemplified by the CARIN language [Levy and Rousset, 1996; 1998], treats concepts and roles as ordinary unary and binary predicates that can also appear in query atoms. This is significant because it allows for the first time conjunctive queries to be expressed over DL databases/Aboxes.

A second important distinction is between recursive and non-recursive Datalog queries. For the non-recursive case (which covers a large portion of practically useful queries), it seems possible to combine some expressive decidable DLs with Datalog, while keeping query answering and even reasoning on queries decidable (see Section 16.4). For the recursive case, undecidability arises sooner, but some studies have identified suitable restrictions on the DL language and/or on the form of Datalog rules, for preserving decidability of query answering.

Consider first $\mathcal{AL}$-LOG. In the rule

$$\begin{aligned} \mathsf{happy}(x) \ \leftarrow \ & \mathsf{marriedTo}(x, y) \wedge \mathsf{employedBy}(y, z) \\ & \ \& \ \mathsf{Person}(x) \wedge \mathsf{Person}(y) \wedge \mathsf{StartUp}(z) \end{aligned}$$

the tests after the ampersand & are for concept membership, while those before it, are for n-ary relations, as in relational databases. The processing of such queries is complicated by the fact that the DL "type database" may contain disjunction or be otherwise incomplete. Instead of the standard answers, one gets a "conditional result", with a side condition c describing necessary DL constraints on the variables in the query. For example, for the above query one might get as answer

$$\mathsf{happy}(\mathsf{ANNA}) \ \text{ if } \ \mathsf{Person}(\mathsf{ANNA})$$

in a database containing

$$\mathsf{marriedTo}(\mathsf{ANNA}, \mathsf{JOE}), \ \mathsf{employedBy}(\mathsf{JOE}, \mathsf{IBM}), \ \mathsf{Person}(\mathsf{JOE}), \ \mathsf{StartUp}(\mathsf{IBM}).$$

Donini *et al.* [1998b] establish that answering queries in recursive $\mathcal{AL}$-LOG is decidable in the case when the DL used is $\mathcal{ALC}$. The framework of $\mathcal{AL}$-LOG is further extended in [Rosati, 1999] to the case of *disjunctive* Datalog, i.e., Datalog with negation as failure in rule bodies and disjunction in the head of rules.

The CARIN approach is more general, but this increase in expressive power comes at a price: for general Datalog rules, the query answering problem is now undecidable as soon as one allows $\forall R.C$ or $\leqslant n\, R$ as concept constructors. (These appear in most DLs.) However, if Datalog rules are restricted to avoid recursion, then query answering is decidable even for the $\mathcal{ALCNR}$ DL. Numerous other results circumscribing the cases when query processing is decidable may be found in [Levy and Rousset, 1998].

16.3.2 Query optimization

In the case when queries can be classified (as when they are descriptions or when the query implication problem is decidable), classification of queries has been proposed as a technique for *query processing and optimization.* In [Beck *et al.*, 1989], among others, queries are classified with respect to schema concepts; if the query concept Q is classified below concept C, then only instances of C need to be checked if they satisfy the full query. Of course, in this classification process one uses the axioms describing the schema of the database.

If the answers to previous queries are cached, then the query concepts can be left in the classification hierarchy, together with the other concepts in the schema. The result is a simple form of the query optimization technique known as "query answering using cached views": find the most specific views V that subsume the query Q; check only the individual instances of V (which, recall, are locally available) to see if they satisfy the query. Potentially, this could provide considerable savings, especially when gathering information from multiple sites, for example.

Buchheit *et al.* [1994b] elaborate on this by using a more powerful query language. In particular, in order to achieve the expressiveness of full FOL, expressing a query is viewed as a two phase process: as much of the query as possible is written in the "query DL" (yielding the so-called "structural part"), and the remainder of the query is written as a constraint in a first order logic notation (yielding the so-called "dirty part"). For example, the following query asks for students, whose advisor is the same as their committee chair, and the advisor is at least 5 years older:

QueryClass QueryStudent **isa** Student **with**
  derived
    $I1$: advisor: Prof
    $I2$: committee.(chair: Thing)
  where $I1 = I2$ **constraint** forall s/QueryStudent (s.age + 5 < s.advisor.age)

In this case, assuming that cached views only have structural conditions, the query is classified using only its own structural conditions. Thereafter, only the instances of the view are tested using both the structural and dirty parts of the query.

Finally, Bergamaschi *et al.* [1997] have investigated the use of DLs in optimizing query evaluation in object-oriented DBMS by eliminating redundant terms, among others. This is accomplished by first expanding the query as much as possible using the information in the schema; for example, subsumption is used to test when the antecedent of a rule can be applied to the query (subsumes it) so that its consequent can be added to it. By repeatedly applying this process, an expanded query is obtained. Then, all the query subterms that subsume the rest of the query (and are therefore redundant) are eliminated one by one. The result is a semantically equivalent description/query which may be more concise than the original one;

hence it may have fewer tests to evaluate. Furthermore, the new expanded query may be classified further down the pre-existing class/view hierarchy, providing more efficient query evaluation, using the query classification technique described earlier. These are forms of so-called "semantic query optimization".

An issue related to efficient processing of large numbers of individuals, is the situation where the user needs to query the conceptual model for DL instances, while the data is presented in a relational database, say. In other words, we need to obtain the proper ABox instances of the DL query (which involves concepts and roles) from the database. The main problem is that processing hundreds of thousands of individuals is not feasible with DL technology because in each case we try to perform complex inferences. However, most of the data in the database is very straightforward, and the corresponding individuals do not generate new inferences. The solution proposed in [Borgida and Brachman, 1993], is to associate with the *primitive* concepts (resp. roles) of the DL knowledge base unary (resp. binary) view tables defined over the DBMS. One can then translate automatically complex descriptions into complex SQL queries over these views. The important effect is that one gets the full benefit of DBMS optimization for the SQL query, and if only a few values satisfy the query, then only a few DL individuals need to be created. For example, for a primitive DL class Student, we might take the values appearing in the enrollee column of relational table Enrollment_R, and use this subset of the Person_R table to generate appropriate individuals in a special view Student_R, which has only one column. (The generation of unique identifiers for these individuals is in itself a research issue.) Similarly, for example, one would generate a two-column view visitor_R corresponding to the role visitor. Complex descriptions over Student and visitor are then translated algorithmically into SQL queries over the corresponding views. Additional optimizations turn out to be necessary to deal properly with multiple queries and functional roles [Borgida and Brachman, 1993].

16.4 Data integration

Integrating different data sources is one of the fundamental problems faced in the last decades by the database community [Batini *et al.*, 1986]. Generally speaking, the goal of a data integration system is to provide a uniform interface to various data sources [Levy, 2000], so as to enable users to focus on specifying what they want. As a result, the data integration system frees the users from tasks such as finding the relevant data sources, interacting with each source in isolation, and selecting, cleaning, and combining data from multiple sources.

The design of a data integration system is a very complex task, which comprises several different aspects. Our goal in this chapter is to discuss the use of DLs in two important aspects, namely:

- The specification of the content of the various data sources.
- The process of computing the answer to queries posed to the data integration system, based on the specification of the sources.

16.4.1 Specifying the content of data sources

The typical architecture of a data integration system allows one to explicitly model data and information needs—i.e., a specification of the data that the system provides to the user—at various levels:

- The *conceptual level* contains a conceptual representation of the sources and of the reconciled integrated data, together with an explicit declarative account of the relationships among their components.
- The *logical level* contains a representation of the sources in terms of a logical data model.

The conceptual level As we have seen before, the conceptual level contains a formal description of the concepts, the relationships between concepts, and the information requirements that the integration application has to deal with. The key feature of this level is that such a description is independent from any system consideration, and is oriented towards the goal of expressing the semantics of the application. In particular, we distinguish among the following elements:

- The *Enterprise Conceptual Schema* is a representation of the global concepts and relationships that are of interest to the application. It corresponds roughly to the notion of global conceptual schema in the traditional approaches to schema integration and to the notion of *world view*, as introduced in [Levy *et al.*, 1995; Kirk *et al.*, 1995].
- For an information source S, the *Source Conceptual Schema* of S is a conceptual representation of the data residing in S.
- The term *Domain Conceptual Schema* is used to denote the union of both the Enterprise Conceptual Schema and the various Source Conceptual Schemas, plus possible inter-schema relationships [Catarci and Lenzerini, 1993].

We have seen in Section 16.2 that DLs are very well suited for data modeling at the conceptual level, so it comes as no surprise that DLs have also been used in data integration projects to represent Source and Enterprise Conceptual Schemas [Catarci and Lenzerini, 1993; Arens *et al.*, 1993; 1996; Levy *et al.*, 1995; Goasdoue *et al.*, 2000]. In this section, following [Calvanese *et al.*, 1998e], we will continue to use the $\mathcal{DLR}$ DL for specifying these conceptual schemas.

As stated above, the Domain Conceptual Schema contains *inter-schema relationships*. In particular, since the sources are of interest in the system, integration does

not simply mean producing the Enterprise Conceptual Schema, but rather being able to establish the correct interdependencies both between the Source Conceptual Schemas and the Enterprise Conceptual Schema, and between the various Source Conceptual Schema.

To specify inter-schema relationships, we make use of the special kinds of assertions available in DL reasoning. In particular, following [Catarci and Lenzerini, 1993], one can use assertions of the following forms:

$$L_i \quad \sqsubseteq_{ext} \quad L_j$$
$$L_i \quad \sqsubseteq_{int} \quad L_j$$

where L_i and L_j are expressions of different schemas. In particular, L_i and L_j are either two relation expressions of the same arity, or two concept expressions. Intuitively, the first assertion states that L_i is extensionally included in L_j, which means that every object that satisfies the expression L_i in source i also satisfies the expression L_j in source j. For example, if the designer knows that the set of students stored in source 1 is a subset of those stored in source 2, then this knowledge is captured by the inter-schema assertion

$$\mathsf{Student}_1 \quad \sqsubseteq_{ext} \quad \mathsf{Student}_2$$

The second assertion states that the concept denoted by the expression L_i in source i is a subconcept of the one denoted by the expression L_j in source j, which means that every object in source i satisfying L_i also satisfies L_j in source j, provided that it does appear in source j. For example, if the designer knows that the concept of student in source 1 is a subconcept of person in source 2, then s/he can use the inter-schema assertion

$$\mathsf{Student}_1 \quad \sqsubseteq_{int} \quad \mathsf{Person}_2$$

It is worth noting that the possibility of reasoning about $\mathcal{DLR}$ schemas allows for sophisticated forms of reasoning on inter-schema assertions, e.g., for inferring those extensional relationships between concepts that are implied by the knowledge on the intensional interdependencies. More details about these forms of reasoning can be found in [Catarci and Lenzerini, 1993; Calvanese *et al.*, 1998e].

The logical level The logical level provides a description of the logical content of each source, called the *Source Schema*. Typically, a Source Schema is provided in terms of a set of relations using the relational logical model of data. So called *wrappers* can be used to hide how the source actually stores its data, the data model it adopts, etc., and presents the source as a set of relations.

The link between the logical representation of a source and the Domain Conceptual Schema can be specified in two different ways.

- According to the so-called *global-as-view approach*, a query over the source relations is associated to each concept in the Domain Conceptual Schema. Every such concept is thus seen as a view over the sources.
- In the alternative *local-as-view approach*, one associates with each source relation a query that describes its content in terms of the Domain Conceptual Schema. In other words, the logical content of a source relation is described in terms of a view over the Domain Conceptual Schema.

In [Levy, 2000], it is argued that the local-as-view approach has several advantages, and we will follow this approach in the rest of the chapter.

To describe the content of the sources through views, one needs a notion of query such as the union of conjunctive queries over the Domain Conceptual Schema. Specifically, a source relation is described in terms of a *query* of the form

$$q(\vec{\mathbf{x}}) \leftarrow conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$$

where:

- The *head* $q(\vec{\mathbf{x}})$ defines the schema of the relation in terms of a name, and the number of columns.
- The *body* describes the content of the relation in terms of the Domain Conceptual Schema.

In [Calvanese *et al.*, 2001c], $conj_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i)$ is a conjunction of *atoms*, and $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i$ are all the variables appearing in the conjunct (we use $\vec{\mathbf{x}}$ to denote a tuple of variables $x_1, \ldots, x_n$, for some n). Each atom is of the form $E(t)$, $R(\vec{\mathbf{t}})$, or $A(t, t')$, where $\vec{\mathbf{t}}$, t, and t' are variables in $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i$ or constants, and E, R, and A are respectively entities, relationships, and attributes appearing in the Domain Conceptual Schema.

The semantics of queries is as follows. Given a database that satisfies the Domain Conceptual Schema, a query q of arity n is interpreted as the set of n-tuples $(d_1, \ldots, d_n)$, with each d_i an object of the database, such that, when substituting each d_i for x_i, the formula

$$\exists \vec{\mathbf{y}}_1 . conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1) \vee \cdots \vee \exists \vec{\mathbf{y}}_m . conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m)$$

evaluates to true.

Analogously to the case of the conceptual level, it is interesting to perform several reasoning tasks on the DL representation of the sources, for example for inferring redundancies and/or inconsistencies among data stored in different sources. Since queries that include atoms from the Conceptual Schema are more expressive, new algorithms are required to answer the following problems:

- *Query containment.* Given two queries q_1 and q_2 (of the same arity n), check whether q_1 is *contained in* q_2, i.e., check if the set of tuples denoted by q_1 is

contained in the set of tuples denoted by q_2 in every database satisfying the Conceptual Schema. Papers that contain results relating to this question include [Levy and Rousset, 1998; Calvanese *et al.*, 1998a; Goasdoue and Rousset, 2000].

- *Query consistency.* Check if a query q over the Conceptual Schema is *consistent*, i.e., check if there exists a database satisfying the Conceptual Schema in which the set of tuples denoted by q is not empty.
- *Query disjointness.* Check whether two queries q_1 and q_2 (of the same arity) over the Conceptual Schema are *disjoint*, i.e., check if the intersection of the set of tuples denoted by q_1 and the set of tuples denoted by q_2 is empty, in every database satisfying the Conceptual Schema.

16.4.2 Query answering

The ultimate goal of a data integration system is to allow the user to pose queries over the global view, and to answer the queries by accessing the sources in a transparent way. The mechanism for answering queries differs depending on the approach adopted for specifying the sources. The possibility of reasoning about queries can provide useful support in both the global-as-view and the local-as-view approaches. As in the previous section, here we focus on the local-as-view approach, that is the one in which query answering is most complex.

In the local-as-view approach, relations at the sources are modeled as views over the virtual database represented by the Domain Conceptual Schema. Since the database is virtual, in order to answer a query Q formulated over the Domain Conceptual Schema, we can only use the source views. In other words, query processing cannot simply be done by looking at a set of relations, as in traditional databases, but requires reasoning on both the form of the query, and the content of the source views. This motivates the idea that query answering in data integration becomes the problem of *view-based query processing*. There are two approaches to view-based query processing, called *query rewriting* and *query answering*, respectively.

In the former approach, we are given a query Q and a set of view definitions, and the goal is to reformulate the query into an equivalent expression that refers only to the views available, and provides the answer to Q.

In the latter approach, besides Q and the view definitions, we also take into account the extensions of the views, and the goal is to compute the set of tuples that are implied by these extensions, i.e., the set of tuples t such that t satisfies Q in all the databases that are consistent with the views.

Notice the difference between the two approaches. In query rewriting, query processing is divided in two steps, where the first re-expresses the query in terms of a given query language over the alphabet of the view names, and the second step evaluates the rewriting over the view extensions. In query answering, we do

not pose any limit on query processing, and the only goal is to exploit all possible information, including view extensions, to compute the answer to the query.

View-based query processing has been extensively investigated by the database community [Levy, 2000]. Only recently has the problem been studied for the case where the Domain Conceptual Schema is expressed in DLs. For example, [Baader *et al.*, 2000] addresses the problem of rewriting queries that are concepts in terms of concepts in the conceptual schema. Query rewriting for to more general queries (e.g., ones involving conjunctions of atoms) has been studied in [Beeri *et al.*, 1997; Levy and Rousset, 1998; Goasdoue *et al.*, 2000; Calvanese *et al.*, 2001c], in some cases taking into consideration complex constraints expressed in DL as part of the Conceptual Schema. One issue that must be addressed here is that the original query Q may not be rewritable as an expression over the views because of limitations of the language for combining views. In this case, one must find heuristic best-effort approximations. Another issue is finding a minimum-cost rewriting (e.g., by eliminating unnecessary look-ups in some of the views).

Finally, we mention that Goasdoue *et al.* [2000] describe an implemented information integration system, which uses a combination of global-as-view and limited local-as-view approach applied to the $\mathcal{ALN}$ DL and non-recursive Horn rules.

Among the pioneering attempts at solving the query answering problem is the Information Manifold system [Levy *et al.*, 1996; 1995], which has detailed algorithms for query rewriting. In the context of heterogeneous databases, Mena *et al.* [2000] propose that each source has its own conceptual schema/ontology expressed in a DL, and these are inter-related by adding "hyponym" (subsumption) relationships between concepts in each. (This is reminiscent of the approach in [Catarci and Lenzerini, 1993].) One of the interesting features of this system is that it takes seriously the approximations resulting from the fact that some queries may not be expressible in terms of the combined ontologies. Among others, they study the notions of "precision" and "accuracy" of recall to quantify this approximation. A solution to the query answering approach is presented in [Calvanese *et al.*, 2000a], which, among others, illustrates the relationship between view-based query answering and ABox reasoning in DLs.

16.5 Conclusions

We have reviewed a number of ways in which DLs can be useful in the development and utilization of databases.

Probably the most successful applications are in areas where the conceptual model of the UofD is required. This includes the initial development stage, as well as access to heterogeneous data sources.

Concerning the initial conceptual modeling: First, DLs are powerful enough to

capture the domain semantics represented by various entity-relationship data models, as well as other data models introduced in the database literature. In fact, with most DLs, one can represent additional constraints. Second, because DLs have a clear semantics, the meaning of the DL model is unambiguous and precise. Third, not only can information be represented, but it can also be reasoned with: one can look for inconsistent class/entity definitions (ones that cannot have any individual instances) and more generally, one can check for the consistency of the entire model. Both of these are signs to the developer that there are modeling errors. Arguably, it is this third aspect, concerning reasoning with the model, that is the greatest advantage of DL models.

DL descriptions can be viewed as necessary and sufficient conditions, and hence as queries (or views!) for a database. DLs are somewhat less successful in this regard (at least in their pure form), because they have limited expressive power compared to the standard calculi known from relational databases, and because they cannot generate new objects—only select subsets of existing objects.

However, if one accepts a DL as a data model, then DL queries can be classified with respect to schema concepts and previous queries, supporting query by refinement and data exploration. The subsumption relationship can also be used for semantic query optimization.

Combining DLs with Datalog rules, or at least supporting conjunctive queries from concepts, is a promising way to obtain a more expressive query language. The evaluation of the resulting queries appears to be decidable with a wide range of DLs if the rules are not recursive. The addition of recursion appears to lead to undecidability relatively quickly. However, full recursion is not an necessity for practical applications, such as information integration, so further research in the possible combinations of DLs and Datalog restrictions is warranted.

The ability to represent the semantics of a UofD is also the reason why DLs are useful in situations where information is to be integrated from various sources, such as heterogeneous or federated databases. It is widely agreed that the integration needs to be achieved at the conceptual level. The DL can be used to define the ontology of each site, and then these ontologies are inter-related; alternatively, a global ontology is specified, and then the sites are described as views over it.