Conceptual Modeling for Data Integration

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2], Maurizio Lenzerini[2], and Riccardo Rosati[2]

[1] KRDB Research Centre
Free University of Bozen-Bolzano
calvanese@inf.unibz.it
[2] Dip. di Informatica e Sistemistica
SAPIENZA Università di Roma
lastname@dis.uniroma1.it

Abstract. The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing the user from the knowledge about where the data are, how they are stored, and how they can be accessed. One of the outcomes of the research work carried out on data integration in the last years is a clear architecture, comprising a global schema, the source schema, and the mapping between the source and the global schema. Although in many research works and commercial tools the global schema is simply a data structure integrating the data at the sources, we argue that the global schema should represent, instead, the conceptual model of the domain. However, to fully pursue such an approach, several challenging issues are to be addressed. The main goal of this paper is to analyze one of them, namely, how to express the conceptual model representing the global schema. We start our analysis with the case where such a schema is expressed in terms of a UML class diagram, and we end up with a proposal of a particular Description Logic, called *DL-Lite*$_{\mathcal{A},id}$. We show that the data integration framework based on such a logic has several interesting properties, including the fact that both reasoning at design time, and answering queries at run time can be done efficiently.

1 Introduction

The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing a client from the knowledge about where the data are, how they are stored, and how they can be accessed. The problem of designing effective data integration solutions has been addressed by several research and development projects in the last years. However, data integration is still one of the major challenges in Information Technology [5]. One of the reasons is that large amounts of heterogeneous data are nowadays available within an organization, but these data have been often collected and stored by different applications and systems. Therefore, on the one hand the need of accessing data by means of flexible and unified mechanisms is becoming more and more important, and, on the other hand, current commercial data integration tools have several drawbacks.

Starting from the late 90s, research in data integration has mostly focused on declarative approaches (as opposed to procedural ones) [32,26]. One of the outcomes of this

A.T. Borgida et al. (Eds.): Mylopoulos Festschrift, LNCS 5600, pp. 173–197, 2009.

research work is a clear architecture for (mediator-based[1]) data integration. According to this architecture [26], the main components of a data integration system are the global schema, the sources, and the mapping. Roughly speaking, the sources represent the repositories where the data are, the global schema represents the unified structure presented to the client, and the mapping relates the source data with the global schema. There are at least two different approaches to the design of the global schema. In the first approach, the global schema is expressed in terms of a database model (e.g., the relational model, or a semistructured data model), and represents a sort of unified data structure accommodating the various data at the sources. In the second approach the global schema provides a conceptual representation of the application domain [6], rather than a specification of a data structure. Thus, in this approach the distinction between the global schema and the data sources reflects the separation between the conceptual level (the one presented to the client), and the logical/physical level of the information system (the one stored in the sources), with the mapping acting as the reconciling structure between the two levels.

Although most of the research projects and the commercial data integration tools follow the first approach, we argue that designing the system in such a way that the global schema represents the conceptual model of the domain has several interesting advantages, both in the design and in the operation of the data integration system.

The first advantage is that a conceptual level in the architecture for data integration is the obvious means for pursuing a declarative approach to integration. As a consequence, all the advantages deriving from making various aspects of the system explicit are obtained, including the crucial fact that the conceptual level provides a system independent specification of the domain of interest to the client. By making the representation of the domain explicit we gain re-usability of the acquired knowledge, which is not achieved when the global schema is simply a unified description of the underlying data sources. This may also have consequences on the design of the user interface. Indeed, conceptual models are naturally expressed in a graphical form, and graphical tools that adequately present the overall information scenario are key factors in user interfaces.

A second advantage is the use of the mapping component of the system for explicitly specifying the relationships between the data sources and the domain concepts. The importance of this clearly emerges when looking at large organizations where the information about data is widespread into separate pieces of documentation that are often difficult to access and non necessarily conforming to common standards. The conceptual model built for data integration can thus provide a common ground for the documentation of the enterprise data stores and can be seen as a formal specification for mediator design. Obviously, the above advantages carry over in the maintenance phase of the data integration system (sources change, hence “design never ends”).

A third advantage has to do with incrementality and extensibility of the system. One criticism that is often raised to mediator-based data integration is that it requires merging and integrating the source data, and this merging process can be very costly. However, the conceptual approach to data integration does not impose to fully integrate the data sources at once. Rather, after building the domain model, one can incrementally add new sources or new elements therein, when they become available, or when needed,

[1] Other architectures, e.g. [4], are outside the scope of this paper.

thus amortizing the cost of integration. Therefore, the overall design can be regarded as the incremental process of understanding and representing the domain on the one hand, and the available data on the other hand.

We believe that all the advantages outlined above represent convincing arguments supporting the conceptual approach to data integration. However, to fully pursue such an approach, several challenging issues are to be addressed. The goal of this paper is to analyze one of them, namely, how to express the conceptual model representing the global schema.

We start our analysis with the case where the global schema of the data integration system is expressed in terms of a UML class diagram. Notably, we show that the expressive power of UML class diagrams is enough to get intractability of various tasks, including query answering. We then present a specific proposal of a logic-based language for expressing conceptual models. The language, called *DL-Lite*$_{\mathcal{A},id}$, is a tractable Description Logic, specifically defined for achieving tractability of the reasoning tasks that are relevant in data integration. The proposed data integration framework, based on such a logic, has several interesting properties, including the fact that both reasoning at design time, and answering queries at run time can be done efficiently. Also, we study possible extensions to the data integration framework based on *DL-Lite*$_{\mathcal{A},id}$, and show that our proposal basically represents an optimal compromise between expressive power and computational complexity.

The paper is organized as follows. In Section 2, we describe a general architecture for data integration, and the basic features of Description Logics, which are the logics we use to formally express conceptual models. In Section 3, we analyze the case where the global schema of the data integration system is expressed in terms of a UML class diagram. In Section 4, we illustrate the basic characteristics of the Description Logic *DL-Lite*$_{\mathcal{A},id}$, and in Section 5, we illustrate a specific proposal of data integration system based on such a logic. In Section 6, we study possible extensions to such data integration framework, whereas in Section 7, we conclude the paper with a discussion on related and future work.

2 The Data Integration Framework

The goal of this section is to describe a general architecture for data integration. We restrict our attention to data integration systems based on a so-called global schema, or mediated schema. In other words, we refer to data integration systems whose aim is providing the user with a representation of the domain of interest to the system, and connecting such a representation to the data residing at the sources. Thus, the global schema implicitly provides a reconciled view of all data, which can be queried by the user. One of the theses of this paper is that the global schema can be profitably expressed in terms of a conceptual model of the domain, and such a conceptual model can be formalized in specialized logics, called Description Logics. Therefore, another goal of this section is to illustrate the basic features of such logics.

2.1 Architecture for Data Integration

According to [26], we formalize a *data integration system* $\mathcal{J}$ in terms of a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{G}$ is the *global schema*, expressed in a language $\mathcal{L}_\mathcal{G}$ over an alphabet $\mathcal{A}_\mathcal{G}$. The alphabet comprises a symbol for each element of $\mathcal{G}$ (i.e., relation if $\mathcal{G}$ is relational, class if $\mathcal{G}$ is object-oriented, etc.).
- $\mathcal{S}$ is the *source schema*, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$. The alphabet $\mathcal{A}_\mathcal{S}$ includes a symbol for each element of the sources.
- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of *assertions* of the forms

$$q_\mathcal{S} \rightsquigarrow q_\mathcal{G},$$
$$q_\mathcal{G} \rightsquigarrow q_\mathcal{S},$$

  where $q_\mathcal{S}$ and $q_\mathcal{G}$ are two queries of the same arity, respectively over the source schema $\mathcal{S}$, and over the global schema $\mathcal{G}$. Queries $q_\mathcal{S}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_\mathcal{S}$, and queries $q_\mathcal{G}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_\mathcal{G}$. Intuitively, an assertion $q_\mathcal{S} \rightsquigarrow q_\mathcal{G}$ specifies that the concept represented by the query $q_\mathcal{S}$ over the sources corresponds to the concept in the global schema represented by the query $q_\mathcal{G}$ (similarly for an assertion of type $q_\mathcal{G} \rightsquigarrow q_\mathcal{S}$).

The global schema provides a description of the domain of interest, and not simply a unified representation of the source data. The source schema describes the structure of the sources, where the real data are. The assertions in the mapping establish the connection between the elements of the global schema and those of the source schema.

The semantics of a data integration system is based on the notion of interpretation in logic. Indeed, in this paper we assume that $\mathcal{G}$ is formalized as a logical theory, and therefore, given a source database D (i.e., a database for $\mathcal{S}$), the semantics of the whole system coincides with the set of interpretations that satisfy all the assertions of $\mathcal{G}$ (i.e., they are logical models of $\mathcal{G}$) and all the assertions of $\mathcal{M}$ with respect to D. Such a set of interpretations, denoted $sem_D(\mathcal{J})$, is called the set of models of $\mathcal{J}$ relative to D.

There are two basic tasks concerning a data integration system that we consider in this paper. The first task is relevant in the design phase of the system, and concerns the possibility of reasoning over the global schema: given $\mathcal{G}$ and a logical assertion α, check whether α holds in every model of $\mathcal{G}$. The second task is query answering, which is crucial during the run-time of the system. Queries to $\mathcal{J}$ are posed in terms of the global schema $\mathcal{G}$, and are expressed in a query language $\mathcal{L}_\mathcal{Q}$ over the alphabet $\mathcal{A}_\mathcal{G}$. A query is intended to provide the specification of which extensional information to extract from the domain of interest in the data integration system. More precisely, given a source database D, the answer $q^{\mathcal{J},D}$ to a query q in $\mathcal{J}$ with respect to D is the set of tuples t of objects such that $t \in q^\mathcal{B}$ (i.e., t is an answer to q over $\mathcal{B}$) for *every* model $\mathcal{B}$ of $\mathcal{J}$ relative to D. The set $q^{\mathcal{J},D}$ is called the set of *certain answers* to q in $\mathcal{J}$ with respect to D. Note that, from the point of view of logic, finding certain answers is a logical implication problem: check whether the fact that t satisfies the query logically follows from the information on the sources and on the mapping.

The above definition of data integration system is general enough to capture virtually all approaches in the literature. Obviously, the nature of a specific approach depends on the characteristics of the mapping, and on the expressive power of the various schema and query languages. For example, the language $\mathcal{L}_\mathcal{G}$ may be very simple (basically allowing for the definition of a set of relations), or may allow for various forms of

integrity constraints to be expressed over the symbols of $\mathcal{A}_\mathcal{G}$. Analogously, the type (e.g., relational, semistructured, etc.) and the expressive power of $\mathcal{L}_\mathcal{S}$ varies from one approach to another.

2.2 Description Logics

Description Logics [2] (DLs) were introduced in the early 80s in the attempt to provide a formal ground to Semantic Networks and Frames. Since then, they have evolved into knowledge representation languages that are able to capture virtually all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases. One of the distinguishing features of the work on these logics is the detailed computational complexity analysis both of the associated reasoning algorithms, and of the logical implication problem that the algorithms are supposed to solve. By virtue of this analysis, most of these logics have optimal reasoning algorithms, and practical systems implementing such algorithms are now used in several projects. In DLs, the domain of interest is modeled by means of *concepts* and *roles* (i.e., binary relationships), which denote classes of objects and binary relations between classes of objects, respectively. Concepts and roles can be denoted using expressions of a specified languages, and the various DLs differ in the expressive power of such a language. The DLs considered in this paper are subsets of a DL called $\mathcal{ALCQIb}_{id}$. $\mathcal{ALCQIb}_{id}$ is an expressive DL that extends the basic DL language $\mathcal{AL}$ (attributive language) with negation of arbitrary concepts (indicated by the letter $\mathcal{C}$), qualified number restrictions (indicated by the letter $\mathcal{Q}$), inverse of roles (indicated by the letter $\mathcal{I}$), boolean combinations of roles (indicated by the letter b), and identification assertions (indicated by the subscript *id*). More in detail, concepts and roles in $\mathcal{ALCQIb}_{id}$ are formed according to the following syntax:

$$\begin{aligned} C, C' &\longrightarrow A \mid \neg C \mid C \sqcap C' \mid C \sqcup C' \mid \\ &\qquad \forall R.C \mid \exists R.C \mid \geqslant n\, R.C \mid \leqslant n\, R.C \\ R, R' &\longrightarrow P \mid P^- \mid R \cap R' \mid R \cup R' \mid R \setminus R' \end{aligned}$$

where A denotes an *atomic concept*, P an *atomic role*, P^- the *inverse* of an atomic role, C, C' arbitrary *concepts*, and R, R' arbitrary *roles*. Furthermore, $\neg C$ denotes concept negation, $C \sqcap C'$ concept intersection, $C \sqcup C'$ concept union, $\forall R.C$ value restriction, $\exists R.C$ qualified existential quantification on roles, and $\geqslant n\, R.C$ and $\leqslant n\, R.C$ so-called number restrictions. We use $\top$, denoting the top concept, as an abbreviation for $A \sqcup \neg A$, for some concept A. An arbitrary role can be an atomic role or its inverse, or a role obtained combining roles through set theoretic operators, i.e., intersection ("$\cap$"), union ("$\cup$"), and difference ("$\setminus$"). W.l.o.g., we assume difference applied only to atomic roles and their inverses.

As an example, consider the atomic concepts *Man* and *Woman*, and the atomic roles *HAS-HUSBAND*, representing the relationship between a woman and the man with whom she is married, and *HAS-CHILD*, representing the parent-child relationship. Then, intuitively, the inverse of *HAS-HUSBAND*, i.e., *HAS-HUSBAND*$^-$, represents the relationship between a man and his wife. Also, *Man* $\sqcup$ *Woman* is a concept representing people (considered the union of men and women), whereas the concept $\exists$*HAS-CHILD*.*Woman* represents those having a daughter, and the concept

$\geqslant 2\,\mathit{HAS\text{-}CHILD}.\mathit{Woman} \sqcap \leqslant 4\,\mathit{HAS\text{-}CHILD}.\top$ represents those having at least two daughters and at most four children.

Like in any DL, an $\mathcal{ALCQIb}_{id}$ *knowledge base* (KB) is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$, the *TBox*, is a finite set of *intensional assertions*, and $\mathcal{A}$, the *ABox*, is a finite set of *extensional* (or, *membership*) *assertions*.

The TBox may contain intensional assertions of two types, namely inclusion assertions, and local identification assertions (see [13] for the meaning of "local" in this context).

- An *inclusion assertion* has the form $C \sqsubseteq C'$, with C and C' arbitrary $\mathcal{ALCQIb}_{id}$ concepts, or the form $R \sqsubseteq R'$, with R and R' arbitrary $\mathcal{ALCQIb}_{id}$ roles. Intuitively, an inclusion assertion states that, in every model of $\mathcal{T}$, each instance of the left-hand side expression is also an instance of the right-hand side expression. For example, the inclusions $\mathit{Woman} \sqsubseteq \leqslant 1\,\mathit{HAS\text{-}HUSBAND}.\top$ and $\exists \mathit{HAS\text{-}HUSBAND}^{-}.\top \sqsubseteq \mathit{Man}$ respectively specifies that women may have at most one husband and that husbands are men.
- A *local identification assertion* (or, simply, identification assertion or identification constraint – IdC) makes use of the notion of path. A *path* is an expression built according to the following syntax,

$$\pi \longrightarrow S \mid D? \mid \pi \circ \pi \tag{1}$$

where S denotes an atomic role or the inverse of an atomic role, and $\pi_1 \circ \pi_2$ denotes the composition of the paths π_1 and π_2. Finally, D denotes a concept, and the expression $D?$ is called a *test relation*, which represents the identity relation on instances of D. Test relations are used in all those cases in which one wants to impose that a path involves instances of a certain concept. For example, $\mathit{HAS\text{-}CHILD} \circ \mathit{Woman}?$ is the path connecting someone with his/her daughters.

A path π denotes a complex property for the instances of concepts: given an object o, every object that is reachable from o by means of π is called a π-*filler* for o. Note that for a certain o there may be several distinct π-fillers, or no π-fillers at all.

If π is a path, the length of π, denoted $\mathit{length}(\pi)$, is 0 if π has the form $D?$, is 1 if π has the form S, and is $\mathit{length}(\pi_1) + \mathit{length}(\pi_2)$ if π has the form $\pi_1 \circ \pi_2$. With the notion of path in place, we are ready for the definition of local identification assertion, which is an assertion of the form

$$(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$$

where C is an arbitrary concept, $n \geq 1$, and $\pi_1, \ldots, \pi_n$ (called the *components* of the identifier) are paths such that $\mathit{length}(\pi_i) \geq 1$ for all $i \in \{1, \ldots, n\}$, and $\mathit{length}(\pi_i) = 1$ for at least one $i \in \{1, \ldots, n\}$. Intuitively, such a constraint asserts that for any two different instances o, o' of C, there is at least one π_i such that o and o' differ in the set of their π_i-fillers. The term "local" emphasizes that at least one of the paths refers to a local property of C.

For example, the identification assertion $(\mathsf{id}\ \mathit{Woman}\ \mathit{HAS\text{-}HUSBAND})$ says that a woman is identified by her husband, i.e., there are not two different women with

$$
\begin{array}{rl}
A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} & \qquad P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\
\neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} & \qquad (P^{-})^{\mathcal{I}} = \{(o, o') \mid (o', o) \in P^{\mathcal{I}}\} \\
(C \sqcap C')^{\mathcal{I}} = C^{\mathcal{I}} \cap C'^{\mathcal{I}} & \qquad (R \cap R')^{\mathcal{I}} = R^{\mathcal{I}} \cap R'^{\mathcal{I}} \\
(C \sqcup C')^{\mathcal{I}} = C^{\mathcal{I}} \cap C'^{\mathcal{I}} & \qquad (R \cup R')^{\mathcal{I}} = R^{\mathcal{I}} \cup R'^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} = \{\, o \mid \forall o'.(o, o') \in R^{\mathcal{I}} \supset o' \in C^{\mathcal{I}} \,\} & \qquad (R \setminus R')^{\mathcal{I}} = R^{\mathcal{I}} \setminus R'^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} = \{\, o \mid \exists o'.(o, o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}} \,\} & \\
(\geqslant n\, R.C)^{\mathcal{I}} = \{\, o \mid |\{o' \in C^{\mathcal{I}} \mid (o, o') \in R^{\mathcal{I}}\}| \geq n \,\} & \\
(\leqslant n\, R.C)^{\mathcal{I}} = \{\, o \mid |\{o' \in C^{\mathcal{I}} \mid (o, o') \in R^{\mathcal{I}}\}| \leq n \,\} &
\end{array}
$$

Fig. 1. Interpretation of $\mathcal{ALCQI}b_{id}$ concepts and roles

the same husband, whereas the identification assertion (id *Man HAS-CHILD*) says that a man is identified by his children, i.e., there are not two men with a child in common. We can also say that there are not two men with the same daughters by means of the identification (id *Man HAS-CHILD* $\circ$ *Woman*?).

The ABox consists of a set of extensional assertions, which are used to state the instances of concepts and roles. Each such assertion has the form $A(a)$, $P(a, b)$, $a = b$, or $a \neq b$, with A and P respectively an atomic concept and an atomic role occurring in $\mathcal{T}$, and a, b constants.

We now turn to the semantics of $\mathcal{ALCQI}b_{id}$, which is given in terms of interpretations. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty interpretation domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$, which assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$ is such a way that the conditions specified in Figure 1 are satisfied. The semantics of an $\mathcal{ALCQI}b_{id}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is the set of *models* of $\mathcal{K}$, i.e., the set of interpretations satisfying all assertions in $\mathcal{T}$ and $\mathcal{A}$. It remains to specify when an interpretation satisfies an assertion.

An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $C \sqsubseteq C'$ (resp., $R \sqsubseteq R'$), if $C^{\mathcal{I}} \subseteq C'^{\mathcal{I}}$ (resp., $R^{\mathcal{I}} \subseteq R'^{\mathcal{I}}$).

In order to define the semantics of IdCs, we first define the semantics of paths, and then specify the conditions for an interpretation to satisfy an IdC. The extension $\pi^{\mathcal{I}}$ of a path π in an interpretation $\mathcal{I}$ is defined as follows:

- if $\pi = S$, then $\pi^{\mathcal{I}} = S^{\mathcal{I}}$,
- if $\pi = D?$, then $\pi^{\mathcal{I}} = \{\, (o, o) \mid o \in D^{\mathcal{I}} \,\}$,
- if $\pi = \pi_1 \circ \pi_2$, then $\pi^{\mathcal{I}} = \pi_1^{\mathcal{I}} \circ \pi_2^{\mathcal{I}}$, where $\circ$ denotes the composition operator on relations.

As a notation, we write $\pi^{\mathcal{I}}(o)$ to denote the set of π-fillers for o in $\mathcal{I}$, i.e., $\pi^{\mathcal{I}}(o) = \{o' \mid (o, o') \in \pi^{\mathcal{I}}\}$. Then, an interpretation $\mathcal{I}$ satisfies the IdC (id $C\ \pi_1, \ldots, \pi_n$) if for all $o, o' \in C^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) \cap \pi_1^{\mathcal{I}}(o') \neq \emptyset \wedge \cdots \wedge \pi_n^{\mathcal{I}}(o) \cap \pi_n^{\mathcal{I}}(o') \neq \emptyset$ implies $o = o'$. Observe that this definition is coherent with the intuitive reading of IdCs discussed above, in particular by sanctioning that two different instances o, o' of C differ in the set of their π_i-fillers when such sets are disjoint.

Finally, to specify the semantics of $\mathcal{ALCQI}b_{id}$ ABox assertions, we extend the interpretation function to constants, by assigning to each constant a an object $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

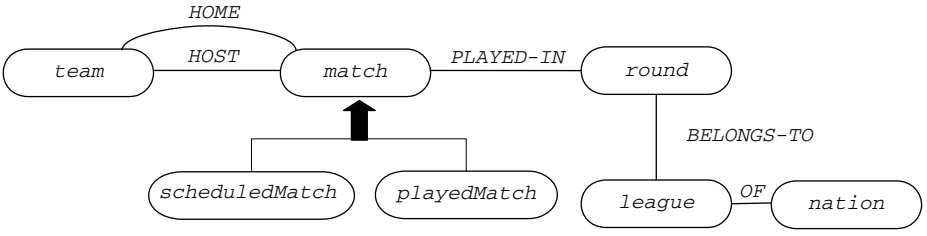


Fig. 2. Diagrammatic representation of the football leagues example

INCLUSION ASSERTIONS	
$league \sqsubseteq \exists OF.\top$	$\exists HOST.\top \sqsubseteq match$
$\exists OF.\top \sqsubseteq league$	$\exists HOST^-.\top \sqsubseteq team$
$\exists OF^-.\top \sqsubseteq nation$	$match \sqsubseteq \exists HOST.\top$
$round \sqsubseteq \exists BELONGS\text{-}TO.\top$	$playedMatch \sqsubseteq match$
$\exists BELONGS\text{-}TO.\top \sqsubseteq round$	$scheduledMatch \sqsubseteq match$
$\exists BELONGS\text{-}TO^-.\top \sqsubseteq league$	$playedMatch \sqsubseteq \neg scheduledMatch$
$match \sqsubseteq \exists PLAYED\text{-}IN.\top$	$match \sqsubseteq playedMatch \sqcup scheduledMatch$
$\exists PLAYED\text{-}IN.\top \sqsubseteq match$	$\top \sqsubseteq \leqslant 1\ OF.\top$
$\exists PLAYED\text{-}IN^-.\top \sqsubseteq round$	$\top \sqsubseteq \leqslant 1\ BELONGS\text{-}TO.\top$
$match \sqsubseteq \exists HOME.\top$	$\top \sqsubseteq \leqslant 1\ PLAYED\text{-}IN.\top$
$\exists HOME.\top \sqsubseteq match$	$\top \sqsubseteq \leqslant 1\ HOME.\top$
$\exists HOME^-.\top \sqsubseteq team$	$\top \sqsubseteq \leqslant 1\ HOST.\top$
IDENTIFICATION ASSERTIONS	
(id *match* *HOME*, *PLAYED-IN*)	
(id *match* *HOST*, *PLAYED-IN*)	

Fig. 3. The TBox in $\mathcal{ALCQI}b_{id}$ for the football leagues example

An interpretation $\mathcal{I}$ satisfies a membership assertion $A(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$, a membership assertion $P(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$, an assertion of the form $a = b$ if $a^{\mathcal{I}} = b^{\mathcal{I}}$, and an assertion of the form $a \neq b$ if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

We also note that, as in many DLs, reasoning in $\mathcal{ALCQI}b_{id}$, i.e., checking whether an assertion holds in every model of a KB, is decidable in deterministic exponential time (see [2]).

We conclude this section with an example in which we present an $\mathcal{ALCQI}b_{id}$ TBox modeling the annual national football[2] championships in Europe, where the championship for a specific nation is called *league* (e.g., the Spanish Liga). A league is structured in terms of a set of *rounds*. Every round contains a set of *matches*, each one characterized by one *home team* and one *host team*. We distinguish between scheduled matches, i.e., matches that have still to be played, and played matches. Obviously, a match falls in exactly one of these two categories.

In Figure 2, we show a schematic representation of (part of) the ontology for the football leagues domain. In this figure, the black arrow represents a partition of one concept into a set of sub-concepts. The TBox assertions in $\mathcal{ALCQI}b_{id}$ capturing the above aspects are shown in Figure 3. In particular, the identification constraints model the fact that a team is the home team of at most one match per round, and the host team of at most one match per round.

[2] Football is called "soccer" in the United States.

3 UML Class Diagram as Global Schema

In this section, we discuss the case where the global schema of a data integration system is a UML class diagram.

Since we concentrate on class diagrams from the data integration perspective, we do not deal with those features that are more relevant for the software engineering perspective, such as operations (methods) associated to classes, or public, protected, and private qualifiers for methods and attributes. Also, for sake of brevity and to smooth the presentation we make some simplifying assumptions that could all be lifted without changing the results presented here (we refer to [3] for further details). In particular, we will not deal explicitly with associations of arity greater than 2, and we will only deal with the following multiplicities: $0..*$ (unconstrained), functional participation $0..1$, mandatory participation $1..*$, and one-to-one correspondence $1..1$. These multiplicities are particularly important since they convey meaningful semantic aspects in modeling, and thus are the most commonly used ones.

Our goal is twofold. One the one hand, we aim at showing how class diagrams can be expressed in DLs. On the other hand, we aim at understanding which is the complexity of the two tasks we are interested in for a data integration system, when the global schema is a UML class diagram. We will show that the formalization in DLs helps us in deriving complexity results for both tasks.

3.1 Representing UML Class Diagrams in DLs

A *class* in a UML class diagram denotes a *sets of objects* with common features. The specification of a class contains its *name* and its *attributes*, each denoted by a name (possibly followed by the *multiplicity*, between square brackets) and with an associated *type*, which indicates the domain of the attribute values.

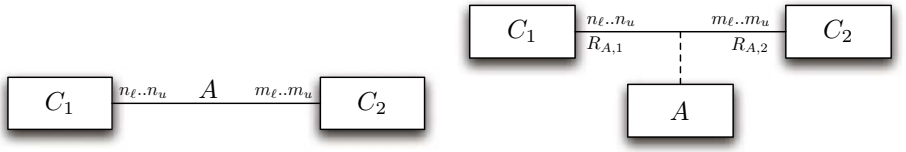
A UML class is represented by a DL concept. This follows naturally from the fact that both UML classes and DL concepts denote *sets of objects*.

A UML *attribute* a of type T for a class C associates to each instance of C, zero, one, or more instances of a class T. An optional *multiplicity* $[i..j]$ for a specifies that a associates to each instance of C, at least i and most j instances of T. When the multiplicity for an attribute is missing, $[1..1]$ is assumed, i.e., the attribute is *mandatory* and *single-valued*.

To formalize attributes, we have to think of an attribute a of type T for a class C as a binary relation between instances of C and instances of T. We capture such a binary relation by means of a DL role a_c. To specify the type of the attribute we use the DL assertions

$$\exists a_C \sqsubseteq C, \qquad\qquad \exists a_C^- \sqsubseteq T.$$

Such assertions specify precisely that, for each instance (c, v) of the role a_C, the object c is an instance of C, and the value v is an instance of T. Notice that in DL, the type T is represented as a concept, although containing values. Note that the attribute name a is not necessarily unique in the whole diagram, and hence two different classes, say C and C' could both have attribute a, possibly of different types. This situation is correctly captured in the DL formalization, where the attribute is contextualized to each class with a distinguished role, i.e., a_C and $a_{C'}$.



(a) Without association class (b) With association class

Fig. 4. Association in UML

To specify that the attribute is mandatory (i.e., multiplicity $[1..*]$), we add the assertion

$$C \sqsubseteq \exists a_C,$$

which specifies that each instance of C participates necessarily at least once to the role a_C. To specify that the attribute is single-valued (i.e., multiplicity $[0..1]$), we add the assertion

$$(\mathsf{funct}\ a_C),$$

which is an abbreviation for $\top \sqsubseteq \leqslant 1\, a_C.\top$. Finally, if the attribute is both mandatory and single-valued (i.e., multiplicity $[1..1]$), we use both assertions together:

$$C \sqsubseteq \exists a_C, \qquad (\mathsf{funct}\ a_C).$$

An *association* in UML is a relation between the instances of two (or more) classes. An association often has a related *association class* that describes properties of the association, such as attributes, operations, etc. A binary association A between the instances of two classes C_1 and C_2 is graphically rendered as in Figure 4(a), where the *multiplicity* $m_\ell..m_u$ specifies that each instance of class C_1 can participate at least m_ℓ times and at most m_u times to association A. The multiplicity $n_\ell..n_u$ has an analogous meaning for class C_2.

An association A between classes C_1 and C_2 is formalized in DL by means of a role A on which we enforce the assertions

$$\exists A \sqsubseteq C_1, \qquad \exists A^- \sqsubseteq C_2.$$

To express the multiplicity $m_\ell..m_u$ on the participation of instances of C_2 for each given instance of C_1, we use the assertion $C_1 \sqsubseteq \exists A$, if $m_\ell = 1$, and $(\mathsf{funct}\ A)$, if $m_u = 1$. We can use similar assertions for the multiplicity $n_\ell..n_u$ on the participation of instances of C_1 for each given instance of C_2, i.e., $C_2 \sqsubseteq \exists A^-$, if $n_\ell = 1$, and $(\mathsf{funct}\ A^-)$, if $n_u = 1$.

Next we focus on an *association* with a related *association class*, as shown in Figure 4(b), where the class A is the association class related to the association, and $R_{A,1}$ and $R_{A,2}$, if present, are the *role names* of C_1 and C_2 respectively, i.e., they specify the role that each class plays within the association A.

We formalize in DL an association A with an association class, by reifying it into a DL concept A and introducing two DL roles $R_{A,1}$, $R_{A,2}$, one for each role of A, which

intuitively connect an object representing an instance of the association to the instances of C_1 and C_2, respectively, that participate to the association[3]. Then, we enforce that each instance of A participates exactly once both to $R_{A,1}$ and to $R_{A,2}$, by means of the assertions

$$A \sqsubseteq \exists R_{A,1}, \qquad (\mathsf{funct}\ R_{A,1}), \qquad A \sqsubseteq \exists R_{A,2}, \qquad (\mathsf{funct}\ R_{A,2}).$$

To represent that the association A is between classes C_1 and C_2, we use the assertions

$$\exists R_{A,1} \sqsubseteq A, \qquad \exists R_{A,1}^- \sqsubseteq C_1, \qquad \exists R_{A,2} \sqsubseteq A, \qquad \exists R_{A,2}^- \sqsubseteq C_2.$$

Finally, we use the assertion

$$(\mathsf{id}\ A\ R_{A,1}, R_{A,2})$$

to specify that each instance of the concept A represents a *distinct* tuple in $C_1 \times C_2$.[4]

We can easily represent in DL multiplicities on an association with association class, by imposing suitable assertions on the inverses of the DL roles modeling the roles of the association. For example, to say that there is a one-to-one participation of instances of C_1 in the association (with related association class) A, we assert

$$C_1 \sqsubseteq \exists R_{A,1}^-, \qquad (\mathsf{funct}\ R_{A,1}^-).$$

In UML, one can use *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that in general do not hold for the parent class.

Generalization is naturally supported in DLs. If a UML class C_2 generalizes a class C_1, we can express this by the DL assertion

$$C_1 \sqsubseteq C_2.$$

Inheritance between DL concepts works exactly as inheritance between UML classes. This is an obvious consequence of the semantics of $\sqsubseteq$, which is based on subsetting. As a consequence, in the formalization, each attribute of C_2 and each association involving C_2 is correctly inherited by C_1. Observe that the formalization in DL also captures directly inheritance among association classes, which are treated exactly as all other classes, and multiple inheritance between classes (including association classes).

Moreover in UML, one can group several generalizations into a class hierarchy, as shown in Figure 5. Such a hierarchy is captured in DL by a set of inclusion assertions, one between each child class and the parent class, i.e.,

$$C_i \sqsubseteq C, \qquad \text{for each } i \in \{1, \ldots, n\}.$$

Often, when defining generalizations between classes, we need to add additional assertions among the involved classes. For example, for the class hierarchy in Figure 5, an

[3] If the roles of the association are not available, we may use an arbitrary DL role name.

[4] Notice that such an approach can immediately be used to represent an association of any arity: it suffices to repeat the above for every component.

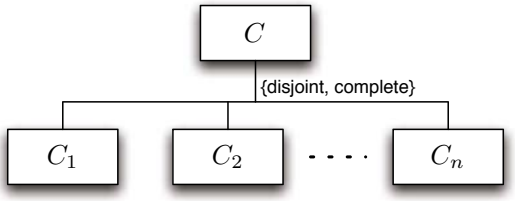


Fig. 5. A class hierarchy in UML

assertion may express that $C_1, \ldots, C_n$ are *mutually disjoint*. In DL, such a relationship can be expressed by the assertions

$$C_i \sqsubseteq \neg C_j, \qquad \text{for each } i, j \in \{1, \ldots, n\} \text{ with } i \neq j.$$

Moreover, we may want to express that a generalization hierarchy is *complete*, i.e., that the subclasses $C_1, \ldots, C_n$ are a *covering* of the superclass C. We can represent such a situation in DL by including the additional assertion

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_n.$$

Such an assertion models a form of *disjunctive information*: each instance of C is either an instance of C_1, or an instance of C_2, ... or an instance of C_n.

Similarly to generalization between classes, UML allows one to state *subset assertions* between associations. A subset assertion between two associations A and A' can be modeled in DL by means of the role inclusion assertion $A \sqsubseteq A'$, involving the two roles A and A' representing the associations. When the two associations A and A' are represented by means of association classes, we need to use the concept inclusion assertion $A \sqsubseteq A'$, together with the role inclusion assertions between corresponding roles of A and A'.

3.2 Reasoning and Query Answering

The fact that UML class diagrams can be captured by DLs enables the possibility of performing sound and complete reasoning to do formal verification at design time and query answering at runtime. Hence, one can exploit such ability to get support during the design phase of the global schema, and to take the information in the global schema fully into account during query answering.

It was shown in [3] that, unfortunately, reasoning (in particular checking the consistency of the diagram, a task to which other typical reasoning tasks of interest reduce) is EXPTIME-hard. What this result tells us is that, if the global schema is expressed in UML, then the support at design time for a data integration system may be impossible if the schema has a reasonable size.

Turning to query answering, the situation is even worse. The results in [11] imply that answering conjunctive queries in the presence of a UML class diagram formed by a single generalization with covering assertion is coNP-hard in the size of the instances

of classes and associations. Hence, query answering over even moderately large data sets is again infeasible in practice. It is not difficult to see that this implies that, in a data integration system where the global schema is expressed as a UML diagram, answering conjunctive queries is coNP-hard with respect to the size of the source data.

Actually, as we will see in the next section, the culprit of such a high complexity is mainly the ability of expressing covering assertions, which induces reasoning by cases. Once we disallow covering and suitably restrict the simultaneous use of subset constraints between associations and multiplicities, not only the sources of exponential complexity disappear, but actually query answering becomes reducible to standard SQL evaluation over a relational database.

4 A Tractable DL: *DL-Lite*$_{\mathcal{A},id}$

We have seen that in a data integration system where the global schema is expressed as a UML class diagram, reasoning is too complex. Thus, a natural question arising at this point is: which is the right language to express the global schema of a data integration system?

In this section, we present *DL-Lite*$_{\mathcal{A},id}$, a DL of the *DL-Lite* family [12,11], enriched with identification constraints (idCs) [12], and show that it is very well suited for conceptual modeling in data integration, in particular for its ability of balancing expressive power with efficiency of reasoning, i.e., query answering, which can be managed through relational database technology.

DL-Lite$_{\mathcal{A},id}$ is essentially a subset of $\mathcal{ALCQI}b_{id}$, but, contrary to the DL presented in Section 2, it distinguishes concepts from *value-domains*, which denote sets of (data) values, and roles from *attributes*, which denote binary relations between objects and values. Concepts, roles, attributes, and value-domains in this DL are formed according to the following syntax[5]:

$$
\begin{array}{lcl}
B & \longrightarrow & A \mid \exists Q \mid \delta(U) \\
C & \longrightarrow & B \mid \neg B \\
Q & \longrightarrow & P \mid P^- \\
R & \longrightarrow & Q \mid \neg Q
\end{array}
\qquad\qquad
\begin{array}{lcl}
E & \longrightarrow & \rho(U) \\
F & \longrightarrow & \top_D \mid T_1 \mid \cdots \mid T_n \\
V & \longrightarrow & U \mid \neg U
\end{array}
$$

In such rules, A, P, and P^- respectively denote an *atomic concept*, an *atomic role*, and the *inverse of an atomic role*, Q and R respectively denote a *basic role* and an *arbitrary role*, whereas B denotes a *basic concept*, C an *arbitrary concept*, U an *atomic attribute*, V an *arbitrary attribute*, E a *basic value-domain*, and F an *arbitrary value-domain*. Furthermore, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values; $\rho(U)$ denotes the *range* of U, i.e., the set of values that U relates to objects; $\top_D$ is the universal value-domain; $T_1, \ldots, T_n$ are n pairwise disjoint unbounded value-domains, corresponding to RDF data types, such as `xsd:string`, `xsd:integer`, etc.

[5] The results mentioned in this paper apply also to *DL-Lite*$_{\mathcal{A},id}$ extended with role attributes (cf. [9]), which are not considered here for the sake of simplicity.

In a *DL-Lite*$_{\mathcal{A},id}$ TBox, assertions have the forms

$$\begin{array}{cccc} B \sqsubseteq C & Q \sqsubseteq R & E \sqsubseteq F & U \sqsubseteq V \\ (\mathsf{funct}\ Q) & (\mathsf{funct}\ U) & (\mathsf{id}\ C\ \pi_1, \ldots, \pi_n) & \end{array}$$

The assertions above, from left to right, respectively denote inclusions between concepts, roles, value-domains, and attributes, (global) functionality on roles and on attributes, and identification constraints[6]. Notice that paths occurring in *DL-Lite*$_{\mathcal{A},id}$ identification assertions may involve also attributes and value-domains, which are instead not among the constructs present in $\mathcal{ALCQI}b_{id}$. More precisely, the symbol S in equation (1) now can be also an attribute or the inverse of an attribute, and the symbol D in (1) now can be also a basic or an arbitrary value-domain.

As for the ABox, beside assertions of the form $A(a)$, $P(a, b)$, with A an atomic concept, P and atomic role, and a, b constants, in *DL-Lite*$_{\mathcal{A},id}$ we may also have assertions of the form $U(a, v)$, where U is an atomic attribute, a a constant, and v a value. Notice however that assertions of the form $a = b$ or $a \neq b$ are not allowed.

We are now ready to define what a *DL-Lite*$_{\mathcal{A},id}$ KB is.

Definition 1. *A DL-Lite*$_{\mathcal{A},id}$ *KB* $\mathcal{K}$ *is a pair* $\langle \mathcal{T}, \mathcal{A} \rangle$*, where* $\mathcal{T}$ *is a DL-Lite*$_{\mathcal{A},id}$ *TBox,* $\mathcal{A}$ *is a DL-Lite*$_{\mathcal{A},id}$ *ABox, and the following conditions are satisfied:*

(1) for each atomic role P*, if either* $(\mathsf{funct}\ P)$ *or* $(\mathsf{funct}\ P^-)$ *occur in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $Q \sqsubseteq P$ *or* $Q \sqsubseteq P^-$*, where* Q *is a basic role;*
(2) for each atomic attribute U*, if* $(\mathsf{funct}\ U)$ *occurs in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $U' \sqsubseteq U$*, where* U' *is an atomic attribute;*
(3) all concepts identified in $\mathcal{T}$ *are basic concepts, i.e., in each IdC* $(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$ *of* $\mathcal{T}$*, the concept* C *is of the form* A*,* $\exists Q$*, or* $\delta(U)$*;*
(4) all concepts or value-domains appearing in the test relations in $\mathcal{T}$ *are of the form* A*,* $\exists Q$*,* $\delta(U)$*,* $\rho(U)$*,* $\top_D$*,* T_1*, . . ., or* T_n*;*
(5) for each IdC α *in* $\mathcal{T}$*, every role or attribute that occurs (in either direct or inverse direction) in a path of* α *is not specialized in* $\mathcal{T}'$*, i.e., it does not appear in the right-hand side of assertions of the form* $Q \sqsubseteq Q'$ *or* $U \sqsubseteq U'$*.*

Intuitively, the conditions stated at points (1-2) (resp., (5)) say that, in *DL-Lite*$_{\mathcal{A},id}$ TBoxes, roles and attributes occurring in functionality assertions (resp., in paths of IdCs) cannot be specialized. All the above conditions are crucial for the tractability of reasoning in our logic.

The semantics of a *DL-Lite*$_{\mathcal{A},id}$ TBox is standard, except that it adopts the *unique name assumption*: for every interpretation $\mathcal{I}$, and distinct constants a, b, we have that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Moreover, it takes into account the distinction between objects and values by partitioning the interpretation domain in two sets, containing objects and values, respectively. Note that the adoption of the unique name assumption in *DL-Lite*$_{\mathcal{A},id}$ makes it meaningless to use ABox assertions of the form $a = b$ and $a \neq b$, which instead occur in $\mathcal{ALCQI}b_{id}$ knowledge bases. Indeed, assertions of the first form cannot be satisfied by *DL-Lite*$_{\mathcal{A},id}$ interpretations, thus immediately making the knowledge base inconsistent, whereas assertions of the second form are always satisfied and are therefore implicit.

[6] We remind the reader that the identification constraints referred to in this paper are local.

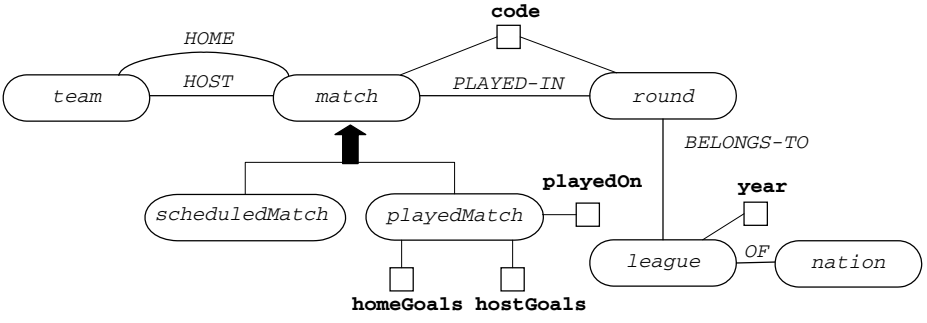


Fig. 6. Diagrammatic representation of the football leagues ontology

We finally recall a notable result given in [13], characterizing the complexity of query answering of UCQs over *DL-Lite*$_{\mathcal{A},id}$ knowledge bases. We remind the reader that AC_0 is the complexity class that corresponds to the complexity in the size of the data of evaluating a first-order (i.e., SQL) query over a relational database (see, e.g., [1]).

Theorem 1 ([13]). *Answering UCQs in DL-Lite*$_{\mathcal{A},id}$ *can be done in* AC_0 *with respect to the size of ABox.*

The above result is proved by showing that it is possible to reduce the query answering problem to the evaluation of a FOL query, directly translatable to SQL, over the database corresponding to the ABox assertions, thus exploiting standard commercial relational database technology.

Let us consider again the example on football leagues introduced in Section 2, and model it as a *DL-Lite*$_{\mathcal{A},id}$ TBox. By virtue of the characteristics of *DL-Lite*$_{\mathcal{A},id}$ we can now explicitly consider also attributes of concepts. In particular, we assume that when a scheduled match takes place, it is played in a specific date, and that for every match that has been played, the number of goals scored by the home team and by the host team are given. Note that different matches scheduled for the same round can be played in different dates. Also, we want to distinguish football championships on the basis of the nation and the year in which a championship takes place (e.g., the 2008 Spanish Liga). Finally, we assume that both matches and rounds have codes. In Figure 6, we show a schematic representation of (part of) the new ontology for the football leagues domain, whereas in Figure 7 the TBox assertions in *DL-Lite*$_{\mathcal{A},id}$ capturing the above aspects are shown. Note that, beside the new assertions involving attributes, Figure 7 lists all assertions given in Figure 3[7], which provide the $\mathcal{ALCQI}b_{id}$ TBox modeling of the football ontology, with the exception of the assertion $match \sqsubseteq scheduledMatch \sqcup playedMatch$. This is actually the price to pay to maintain reasoning tractable in *DL-Lite*$_{\mathcal{A},id}$, and in particular conjunctive query answering in AC_0. Indeed, the above assertion expresses the covering of the concept *match* with the concepts *scheduledMatch* and *playedMatch*, but as said in Section 3, the presence of covering assertions makes query answering coNP-hard in the size of the ABox.

[7] We have used $\exists R$ instead of $\exists R.\top$, and inclusions of the form $\top \sqsubseteq \leqslant 1\, R.\top$ are expressed as functional assertions of the form $(\mathsf{funct}\ R)$.

INCLUSION ASSERTIONS	
$\mathit{league} \sqsubseteq \exists \mathit{OF}$	$\mathit{playedMatch} \sqsubseteq \mathit{match}$
$\exists \mathit{OF} \sqsubseteq \mathit{league}$	$\mathit{scheduledMatch} \sqsubseteq \mathit{match}$
$\exists \mathit{OF}^- \sqsubseteq \mathit{nation}$	$\mathit{playedMatch} \sqsubseteq \neg \mathit{scheduledMatch}$
$\mathit{round} \sqsubseteq \exists \mathit{BELONGS\text{-}TO}$	
$\exists \mathit{BELONGS\text{-}TO} \sqsubseteq \mathit{round}$	$\mathit{league} \sqsubseteq \delta(\mathbf{year})$
$\exists \mathit{BELONGS\text{-}TO}^- \sqsubseteq \mathit{league}$	$\mathit{match} \sqsubseteq \delta(\mathbf{code})$
$\mathit{match} \sqsubseteq \exists \mathit{PLAYED\text{-}IN}$	$\mathit{round} \sqsubseteq \delta(\mathbf{code})$
$\exists \mathit{PLAYED\text{-}IN} \sqsubseteq \mathit{match}$	$\mathit{playedMatch} \sqsubseteq \delta(\mathbf{playedOn})$
$\exists \mathit{PLAYED\text{-}IN}^- \sqsubseteq \mathit{round}$	$\mathit{playedMatch} \sqsubseteq \delta(\mathbf{homeGoals})$
$\mathit{match} \sqsubseteq \exists \mathit{HOME}$	$\mathit{playedMatch} \sqsubseteq \delta(\mathbf{hostGoals})$
$\exists \mathit{HOME} \sqsubseteq \mathit{match}$	$\rho(\mathbf{playedOn}) \sqsubseteq$ `xsd:date`
$\exists \mathit{HOME}^- \sqsubseteq \mathit{team}$	$\rho(\mathbf{homeGoals}) \sqsubseteq$ `xsd:nonNegativeInteger`
$\mathit{match} \sqsubseteq \exists \mathit{HOST}$	$\rho(\mathbf{hostGoals}) \sqsubseteq$ `xsd:nonNegativeInteger`
$\exists \mathit{HOST} \sqsubseteq \mathit{match}$	$\rho(\mathbf{code}) \sqsubseteq$ `xsd:positiveInteger`
$\exists \mathit{HOST}^- \sqsubseteq \mathit{team}$	$\rho(\mathbf{year}) \sqsubseteq$ `xsd:positiveInteger`
FUNCTIONAL ASSERTIONS	
(funct *OF*)	(funct **year**)
(funct *BELONGS-TO*)	(funct **code**)
(funct *PLAYED-IN*)	(funct **playedOn**)
(funct *HOME*)	(funct **homeGoals**)
(funct *HOST*)	(funct **hostGoals**)
IDENTIFICATION CONSTRAINTS	
1. (id *league OF*, **year**)	6. (id *playedMatch* **playedOn**, *HOST*)
2. (id *round BELONGS-TO*, **code**)	7. (id *playedMatch* **playedOn**, *HOME*)
3. (id *match PLAYED-IN*, **code**)	8. (id *league* **year**, $\mathit{BELONGS\text{-}TO}^- \circ \mathit{PLAYED\text{-}IN}^- \circ \mathit{HOME}$)
4. (id *match HOME*, *PLAYED-IN*)	9. (id *league* **year**, $\mathit{BELONGS\text{-}TO}^- \circ \mathit{PLAYED\text{-}IN}^- \circ \mathit{HOST}$)
5. (id *match HOST*, *PLAYED-IN*)	10. (id *match HOME*, *HOST*, $\mathit{PLAYED\text{-}IN} \circ \mathit{BELONGS\text{-}TO} \circ \mathbf{year}$)

Fig. 7. The TBox in *DL-Lite*$_{\mathcal{A},id}$ for the football leagues example

The identification constraints given in Figure 7 model the following aspects:

1. No nation has two leagues in the same year.
2. Within a league, the code associated to a round is unique.
3. Every match is identified by its code within its round.
4. A team is the home team of at most one match per round.
5. As above for the host team.
6. No home team participates in different played matches in the same date
7. As above for the host team.
8. No home team plays in different leagues in the same year.
9. As above for the host team.
10. No pair (home team, host team) plays different matches in the same year.

5 Data Integration with *DL-Lite*$_{\mathcal{A},id}$

In this section, we illustrate a specific proposal of data integration system based on *DL-Lite*$_{\mathcal{A},id}$, by describing the three components of the system. The choice for the languages used in the three components is tailored towards the goal of an optimal trade-off between expressive power and complexity. After the description of the three components, we briefly illustrate the algorithm for answering queries in our approach, and we discuss its computational complexity.

5.1 The Global Schema

As said before, one of the basic characteristics of the approach to data integration advocated in this paper is that the global schema represents the conceptual model of the domain of interest, rather than a mere description of a unified view of the source data. We have also discussed the advantages of expressing the conceptual model in terms of a DL. Finally, we have seen in the previous section that *DL-Lite*$_{\mathcal{A},id}$ represents a valuable choice as a formalism for expressing the global schema of a data integration system. Therefore, in our approach to data integration, the global schema is expressed in terms of a *DL-Lite*$_{\mathcal{A},id}$ TBox. It is interesting to observe that most of the properties of class diagrams discussed in Section 3 can indeed be expressed in *DL-Lite*$_{\mathcal{A},id}$. The only modeling construct that cannot be fully represented is covering. In other words, generalizations expressible in *DL-Lite*$_{\mathcal{A},id}$ are not complete. Also, functional associations cannot be specialized.

5.2 The Source Schema

If $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is a data integration system in our approach, $\mathcal{S}$ is assumed to be a flat relational database schema, representing the schemas of all the data sources. Actually, this is not a limitation of the system, since the source schema can always be thought of as the schema managed by a relational data federation tool. Specifically, we assume that a data federation tool is in charge of interacting with the data sources, presenting them as a single relational database schema. Such a schema is obtained by wrapping physical sources, possibly heterogeneous, and not necessarily in relational format. Furthermore, the data federation tool is in charge of answering queries formulated over the source schema, by suitably transforming such queries, forwarding them to the right sources, and finally combining the single results into the overall answer. In other words, the data federation tool makes the whole system independent from the physical nature of the sources, by providing a logical representation of them (physical independence), whereas the other components of the system make all the logical aspects transparent to the user, by maintaining the conceptual global schema separate from the logical federated schema, and connecting them via suitable mappings (logical independence).

5.3 The Mapping

The mappings in our approach establish the relationship between the source schema and the global schema, thus specifying how data stored at the sources are linked to the instances of the concepts and the roles in the global schema. More specifically, we follow the *GAV (global-as-view)* approach for specifying mappings, which requires to describe the meaning of every element of the global schema by associating to it a view over the sources. The dual approach, called LAV (local-as-view), would require the sources to be defined as views over the global schema.

Moreover, our mapping specification language takes suitably into account the *impedance mismatch* problem, i.e., the mismatch between the way in which data is (and can be) represented in a data source, and the way in which the corresponding information is rendered through the global schema.

The mapping assertions keep data value constants separate from object identifiers, and construct identifiers as (logic) terms over data values. More precisely, object identifiers in our approach are *terms* of the form $f(d_1, \ldots, d_n)$, called *object terms*, where f is a function symbol of arity $n > 0$, and $d_1, \ldots, d_n$ are data values stored at the sources. Note that this idea traces back to the work done in deductive object-oriented databases [24].

We detail below the above ideas. The mapping component is specified through a set of mapping assertions, each of the form

$$\Phi(\boldsymbol{v}) \rightsquigarrow G(\boldsymbol{w})$$

where

- $\Phi(\boldsymbol{v})$, called the *body* of the mapping, is a first-order logic (FOL) query of arity $n > 0$, with distinguished variables $\boldsymbol{v}$, over the source schema $\mathcal{S}$ (we will write such query in the SQL syntax), and
- $G(\boldsymbol{w})$, called the *head*, is an atom where G can be an atomic concept, an atomic role, or an atomic attribute occurring in the global schema $\mathcal{G}$, and $\boldsymbol{w}$ is a sequence of terms.

We define three different types of mapping assertions:

- *Concept mapping assertions*, in which the head is a unary atom of the form $A(f(\boldsymbol{v}))$, where A is an atomic concept and f is a function symbol of arity n;
- *Role mapping assertions*, in which the head is a binary atom of the form $P(f_1(\boldsymbol{v}'), f_2(\boldsymbol{v}''))$, where P is an atomic role, f_1 and f_2 are function symbols of arity $n_1, n_2 > 0$, and $\boldsymbol{v}'$ and $\boldsymbol{v}''$ are sequences of variables appearing in $\boldsymbol{v}$;
- *Attribute mapping assertions*, in which the head is a binary atom of the form $U(f(\boldsymbol{v}'), v'' : T_i)$, where U is an atomic attribute, f is a function symbol of arity $n' > 0$, $\boldsymbol{v}'$ is a sequence of variables appearing in $\boldsymbol{v}$, v'' is a variable appearing in $\boldsymbol{v}$, and T_i is an RDF data type.

In words, such mapping assertions are used to map source relations (and the tuples they store), to concepts, roles, and attributes of the ontology (and the objects and the values that constitute their instances), respectively. Note that an attribute mapping also specifies the type of values retrieved from the source database, in order to guarantee coherency of the system.

We conclude this section with an example of mapping assertions, referring again to the football domain. Suppose that the source schema contains the relational table `TABLE(mcode,league,round,home,host)`, where a tuple (m, l, r, h_1, h_2) with $l > 0$ represents a match with code m of league l and round r, and with home team h_1 and host team h_2. If we want to map the tuples from the table `TABLE` to the global schema shown in Figure 7, the mapping assertions might be as shown in Figure 8. M_1 is a concept mapping assertion that selects from `TABLE` the code and the round of matches (only for the appropriate tuples), and then uses such data to build instances of the concept *match*, using the function symbol $\mathbf{m}$. M_2 is an attribute mapping assertion that is used to "populate" the attribute $\mathbf{code}$ for the objects that are instances of *match*. Finally, M_3 is a role mapping assertion relating `TABLE` to the atomic role *PLAYED-IN*,

M_1: `SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0` $\rightsquigarrow$ *match*(**m**(`T.mcode, T.round, T.league`))

M_2: `SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0` $\rightsquigarrow$ **code**(**m**(`T.mcode, T.round, T.league`), `T.mcode : xsd:string`)

M_3: `SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0` $\rightsquigarrow$ *PLAYED-IN*(**m**(`T.mcode, T.round, T.league`), **r**(`T.round, T.league`))

Fig. 8. Example of mapping assertions

where instances of *round* are denoted by means of the function symbol **r**. We notice that in the mapping assertion M_2, the mapping designer had to specify a correct *DL-Lite*$_{\mathcal{A},id}$ data type for the values extracted from the source.

We point out that, during query answering, the body of each mapping assertion is never really evaluated in order to extract values from the sources to build instances of the global schema, but rather it is used to unfold queries posed over the global schema, rewriting them into queries posed over the source schema. We discuss this aspect next.

5.4 Query Answering

We sketch here our query answering technique (more details can be found in [30,10]). Consider a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and a database D for $\mathcal{S}$, and assume that $\mathcal{J}$ is satisfiable with respect to D, i.e., $sem_D(\mathcal{J}) \neq \emptyset$ (cf. Section 2.1).

We start with the following observation. Suppose we evaluate (over D) the queries in the left-hand sides of the mapping assertions, and we materialize accordingly the corresponding assertions in the right-hand sides. This would lead to a set of ground assertions, denoted by $\mathcal{A}^{\mathcal{M},D}$, that can be considered as a *DL-Lite*$_{\mathcal{A},id}$ ABox. It can be shown that query answering over $\mathcal{J}$ and D can be reduced to query answering over the *DL-Lite*$_{\mathcal{A},id}$ knowledge base constituted by the TBox $\mathcal{G}$ and the ABox $\mathcal{A}^{\mathcal{M},D}$. However, due to the materialization of $\mathcal{A}^{\mathcal{M},D}$, the query answering algorithm resulting from this approach would be polynomial in the size of D. On the contrary, our idea is to avoid any ABox materialization, but rather answer Q by reformulating it into a new query that can be afterwards evaluated directly over the database D. The resulting query answering algorithm is much more efficient than the one sketched above, and is constituted by four steps, which are called *rewriting*, *filtering*, *unfolding*, and *evaluation*, and are described in the following.

Rewriting. Given a UCQ Q over a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a source database D for $\mathcal{J}$, the rewriting step computes a new UCQ Q_1 over $\mathcal{J}$, where the assertions of $\mathcal{G}$ are compiled in. In computing the rewriting, only inclusion assertions of the form $B_1 \sqsubseteq B_2$, $Q_1 \sqsubseteq Q_2$, and $U_1 \sqsubseteq U_2$ are taken into account, where B_i, Q_i, and U_i, with $i \in \{1, 2\}$, are a basic concept, a basic role, and an atomic attribute, respectively. Intuitively, the query Q is rewritten according to the knowledge specified in $\mathcal{G}$ that is relevant for answering Q, in such a way that the rewritten query Q_1 is such that $Q_1^{\langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D} = Q^{\mathcal{J}, D}$, i.e., the rewriting allows to get rid of $\mathcal{G}$.

We refer to [30,12] for a formal description of the query rewriting algorithm and for a proof of its soundness and completeness. We only notice here that the rewriting procedure does not depend on the source database D, runs in polynomial time in the size of $\mathcal{G}$, and returns a query Q_1 whose size is at most exponential in the size of Q.

Filtering. Let Q_1 be the UCQ produced by the rewriting step above. In the filtering step we take care of a particular problem that the disjuncts, i.e., conjunctive queries, in Q_1 might have. Specifically, a conjunctive query cq is called *ill-typed* if it has at least one join variable x appearing in two incompatible positions in cq, i.e., such that the TBox $\mathcal{G}$ of our data integration system logically implies that x is both of type T_i, and of type T_j, with $T_i \neq T_j$ (remember that in *DL-Lite*$_{\mathcal{A},id}$ data types are pairwise disjoint). The purpose of the filtering step is to remove from the UCQ Q_1 all the ill-typed conjunctive queries. Intuitively, such a step is needed because the query Q_1 has to be unfolded and then evaluated over the source database D (cf. the next two steps of the query answering algorithm, described below). These last two steps, performed for an ill-typed conjunctive query might produce incorrect results.

Unfolding. Given the UCQ Q_2 over $\mathcal{J}$ computed by the filtering step, the unfolding step computes, by using logic programming techniques, an SQL query Q_3 over the source schema $\mathcal{S}$, that possibly returns object terms. It can be shown [30] that Q_3 is such that $Q_3^D = Q_2^{\langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D}$, i.e., unfolding allows us to get rid of $\mathcal{M}$. Moreover, the unfolding procedure does not depend on D, runs in polynomial time in the size of $\mathcal{M}$, and returns a query whose size is polynomial in the size of Q_2.

Evaluation. The evaluation step consists in simply delegating the evaluation of the SQL query Q_3, produced by the unfolding step, to the data federation tool managing the data sources. Formally, such a tool returns the set Q_3^D, i.e., the set of tuples obtained from the evaluation of Q_3 over D.

5.5 Correctness and Complexity of Query Answering

It can be shown that the query answering procedure described above correctly computes the certain answers to UCQs. Based on the computational properties of such an algorithm, we can then characterize the complexity of our query answering method.

Theorem 2. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, and D a source database for $\mathcal{J}$. Answering a UCQ over $\mathcal{J}$ with respect to D can be reduced to the evaluation of an SQL query over D, and can be done in* AC_0 *in the size of D.*

In other words, the above theorem says that UCQs in our approach are FO-rewritable.

Finally, we remark that, as we said at the beginning of this section, we have assumed that the data integration system $\mathcal{J}$ is consistent with respect to the database D, i.e., $sem_D(\mathcal{J})$ is non-empty. Notably, it can be shown that all the machinery we have devised for query answering can also be used for checking consistency of $\mathcal{J}$ with respect to D. Therefore, checking consistency can also be reduced to sending appropriate SQL queries to the source database [30,13].

6 Extending the Data Integration Framework

We now analyze the possibility of extending the data integration setting presented above without affecting the complexity of query answering. In particular, we investigate possible extensions for the language for expressing the global schema, the language for expressing the mappings, and the language for expressing the source schema.

We start by dealing with extending the global schema language. There are two possible ways of extending *DL-Lite*$_{\mathcal{A},id}$. The first one corresponds to a proper language extension, i.e., adding new DL constructs to *DL-Lite*$_{\mathcal{A},id}$, while the second one consists of changing/strengthening the semantics of the formalism.

Concerning language extensions, the results in [11] show that it is not possible to add any of the usual DL constructs to *DL-Lite*$_{\mathcal{A},id}$ while keeping the data complexity of query answering within AC_0. This means that *DL-Lite*$_{\mathcal{A},id}$ is essentially the most expressive DL allowing for data integration systems where query answering is FO-rewritable.

Concerning the possibility of strengthening the semantics, we briefly analyze the consequences of removing the *unique name assumption* (UNA), i.e., the assumption that, in every interpretation of a data integration system, two distinct object terms and two distinct value constants denote two different domain elements. Unfortunately, this leads query answering out of LOGSPACE, and therefore, this leads to loosing FO-rewritability of queries.

Theorem 3 ([10]). *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ data integration system extended by removing the UNA, and D a database for $\mathcal{S}$. Computing the certain answers to a query constituted by a single atom in $\mathcal{J}$ with respect to D is* NLOGSPACE*-hard in the size of D.*

Next we consider extensions to the mapping language. The possibility of extending the language used to express the mapping has been analyzed in [10], which considers the so-called GLAV mappings, i.e., assertions that relate conjunctive queries over the sources to conjunctive queries over the global schema. Such assertions are therefore an extension of both GAV and LAV mappings. Unfortunately, even with LAV mappings only, it has been shown that instance checking and query answering are no more in LOGSPACE with respect to data complexity. Thus, with LAV mappings, we again loose FO-rewritability of UCQs.

Theorem 4 ([10]). *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ data integration system extended with LAV mapping assertions, and D a database for $\mathcal{S}$. Computing the certain*

answers to a query constituted by a single atom in $\mathcal{J}$ with respect to D is NLOGSPACE*-hard in the size of D.*

Finally, we consider the possibility of handling source schemas beyond the relational model. The data integration architecture referred to in this paper assumes to deal with relational sources, managed by a relational data federation tool. It is not hard to see, however, that all the results mentioned here apply also if we consider federation tools that provide a representation of the data at the sources according to a different data model (e.g., XML). Obviously, depending on the specific data model adopted by the data federation tool, we have to resort to a suitable query language for expressing the source queries appearing in the mapping assertions. To adhere to the framework adopted in this paper, the only constraint imposed on the query language (that is trivially satisfied by virtually all query languages used in practice) is that it is able to extract tuples of values from the sources.

7 Discussion and Conclusions

Starting from the late 90s, research in data integration has mostly focused on declarative approaches (as opposed to procedural ones) [32,26], such as the one advocated here. The GAV approach for specifying mappings has been proposed e.g., in [15,20,31,22], while the LAV approach is at the basis of the work in [25,18,14].

Although in the present work we make use of GAV mappings, the presence of constraints expressed in a rich ontology language in the global schema, makes query answering in our setting more similar to what is carried out in LAV data integration systems rather than in GAV systems. Indeed, while in general GAV systems have been realized as (simple) hierarchies of wrappers and mediators, query answering in LAV can be considered a form of reasoning in the presence of incomplete information, and thus significantly more complex. Early systems based on this approach, like Information Manifold [28,29], or INFOMASTER [21,19], have implemented algorithms [28] for rewriting queries using views (where the views are the ones specified through the CQs in the mappings). The relationship between LAV and GAV data integration systems is explored in [7], where it is indeed shown that a LAV system can be converted into a GAV one by introducing suitable inclusion dependencies in the global schema. If no functionality assertions are present in the global schema, such inclusion dependencies can then be dealt with in a way similar to what is done here for concept and role inclusions in *DL-Lite*$_{\mathcal{A},id}$. Note that this is no longer possible, instead, in the presence of functionality assertions.

The approach illustrated in this paper has been implemented in a prototype system called MASTRO-I (see [13]). We conclude the paper by mentioning some aspects that are important for the problem of semantic data integration, but that have not been addressed yet in the development of the system.

A first important point is handling inconsistencies in the data, possibly using a declarative, rather than an ad-hoc procedural approach. An interesting proposal is the one of the INFOMIX system [27] for the integration of heterogeneous data sources (e.g., relational, XML, HTML) accessed through a relational global schema with powerful forms of integrity constraints. The query answering technique proposed in such a system is

based on query rewriting in Datalog enriched with negation and disjunction, under stable model semantics [8,23].

A second interesting issue for further work is looking at "write-also" data integration tools. Indeed, while the techniques presented in this paper provide support for answering queries posed to the data integration system, it could be of interest to also deal with updates expressed on the global schema (e.g., according to the approach described in [16,17]). The most challenging issue to be addressed in this context is to design mechanisms for correctly reformulating an update expressed over the ontology into a series of insert and delete operations on the data sources.

Acknowledgements. This research has been partially supported by the IP project OntoRule (ONTOlogies meet Business RULEs ONtologiES), funded by the EC under ICT Call 3 FP7-ICT-2008-3, contract number FP7-231875, by project DASIbench (Data and Service Integration workbench), funded by IBM through a Faculty Award grant, and by the MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading (1995)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
3. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1–2), 70–118 (2005)
4. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: A vision. In: Proc. of the 5th Int. Workshop on the Web and Databases, WebDB 2002 (2002)
5. Bernstein, P.A., Haas, L.: Informaton integration in the enterprise. Communications of the ACM 51(9), 72–79 (2008)
6. Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.): On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer, Heidelberg (1984)
7. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: On the expressive power of data integration systems. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 338–350. Springer, Heidelberg (2002)
8. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), pp. 16–21 (2003)
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Linking data to ontologies: The description logic *DL-Lite*$_A$. In: Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED 2006). CEUR Electronic Workshop Proceedings, vol. 216 (2006), `http://ceur-ws.org/`
10. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R., Ruzzi, M.: Data integration through *DL-Lite*$_A$ ontologies. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 26–47. Springer, Heidelberg (2008)

11. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 260–270 (2006)
12. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
13. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Path-based identification constraints in description logics. In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008), pp. 231–241 (2008)
14. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Data integration in data warehousing. Int. J. of Cooperative Information Systems 10(3), 237–271 (2001)
15. Carey, M.J., Haas, L.M., Schwarz, P.M., Arya, M., Cody, W.F., Fagin, R., Flickner, M., Luniewski, A., Niblack, W., Petkovic, D., Thomas, J., Williams, J.H., Wimmers, E.L.: Towards heterogeneous multimedia information systems: The Garlic approach. In: Proc. of the 5th Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM 1995), pp. 124–131. IEEE Computer Society Press, Los Alamitos
16. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the update of description logic ontologies at the instance level. In: Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006), pp. 1271–1276 (2006)
17. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the approximation of instance level update and erasure in description logics. In: Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007), pp. 403–408 (2007)
18. Duschka, O.M., Genesereth, M.R.: Answering recursive queries using views. In: Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 1997), pp. 109–116 (1997)
19. Duschka, O.M., Genesereth, M.R., Levy, A.Y.: Recursive query plans for data integration. J. of Logic Programming 43(1), 49–73 (2000)
20. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.D., Vassalos, V., Widom, J.: The TSIMMIS approach to mediation: Data models and languages. J. of Intelligent Information Systems 8(2), 117–132 (1997)
21. Genereseth, M.R., Keller, A.M., Duschka, O.M.: Infomaster: An information integration system. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 539–542 (1997)
22. Goh, C.H., Bressan, S., Madnick, S.E., Siegel, M.D.: Context interchange: New features and formalisms for the intelligent integration of information. ACM Trans. on Information Systems 17(3), 270–293 (1999)
23. Grieco, L., Lembo, D., Ruzzi, M., Rosati, R.: Consistent query answering under key and exclusion dependencies: Algorithms and experiments. In: Proc. of the 14th Int. Conf. on Information and Knowledge Management (CIKM 2005), pp. 792–799 (2005)
24. Hull, R.: A survey of theoretical research on typed complex database objects. In: Paredaens, J. (ed.) Databases, pp. 193–256. Academic Press, London (1988)
25. Kirk, T., Levy, A.Y., Sagiv, Y., Srivastava, D.: The Information Manifold. In: Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments, pp. 85–91 (1995)
26. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), pp. 233–246 (2002)
27. Leone, N., Eiter, T., Faber, W., Fink, M., Gottlob, G., Greco, G., Kalka, E., Ianni, G., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Rosati, R., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX system for advanced integration of incomplete and inconsistent data. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 915–917 (2005)

28. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogenous information sources using source descriptions. In: Proc. of the 22nd Int. Conf. on Very Large Data Bases, VLDB 1996 (1996)
29. Levy, A.Y., Srivastava, D., Kirk, T.: Data model and query evaluation in global information systems. J. of Intelligent Information Systems 5, 121–143 (1995)
30. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 133–173. Springer, Heidelberg (2008)
31. Tomasic, A., Raschid, L., Valduriez, P.: Scaling access to heterogeneous data sources with DISCO. IEEE Trans. on Knowledge and Data Engineering 10(5), 808–823 (1998)
32. Ullman, J.D.: Information integration using logical views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1996)