Querying Databases from Description Logics

Paolo Bresciani
IRST, I-38050 Trento Povo, TN, Italy
bresciani@irst.itc.it

Abstract

Two different aspects of data management are addressed by description logics (DL) and databases (DB): the semantic organization of data and powerful reasoning services (by DL) and their efficient management and access (by DB). It is recently emerging that experiences from both DL and DB should profitably cross-fertilize each other, and a great interest is rising about this topic.

In the present paper our technique, that allows uniform access – by means of a DL-based query language – to information distributed over knowledge bases and databases, is briefly reviewed. Our extended paradigm integrates the separately existing retrieving functions of description logics management systems (DLMS) and of database management systems (DBMS) in order to allow, via a query language grounded on a DL-based schema knowledge, uniformly formulating and answering queries, so that uniform retrieval from mixed knowledge/data bases is possible.

In particular, some new developments extending those presented in [Bresciani, 1994] are introduced. By means of them the mapping between DL *concepts* and DB views is not more limited to primitive concepts, but also to some non-primitively defined ones.

1 Introduction

The main difference between knowledge representation (KR) and database (DB) systems is that the latter are oriented to the efficient management of large amount of data while the former seek to give a more structured representation of the universe of discourse in which data are placed. More precisely, in a KR system the universe of discourse is described by means of a collection of terms – or *concepts* – that are placed into a taxonomy. The capability of *classifying* concepts to form taxonomies is given by an appropriate calculus, whose first goal is to provide a *subsumption* algorithm. Concept Languages together with appropriate subsumption calculi are called Description Logics (DL). Databases, instead, are suited to manage data efficiently, with little concern about their dimension, but their formalism for organizing them in a structured way is quite absent, as well as the capability to infer new information from the existing ones. Thus, two different aspects of data management are addressed by description logics management systems (DLMS) and by database management systems (DBMS): the semantic organization of data (by DLMS), and their efficient management and access (by DBMS).

The importance of KR has been regarded as fundamental for the construction of good Intelligent Information Systems for more than ten years (see, e.g., [Tou *et al.*, 1982]), but only recently the theoretical foundations of a DL approach to DB have been established [Buchheit *et al.*, 1994].

From another point of view, KR-based applications and, more generally, AI-based applications can be widely enhanced by AI/DB interfaces [Pastor *et al.*, 1992, McKay *et al.*, 1990].

In particular, for the task of implementing DL-based applications, several reasons can be argued in favor of the use of external DB:

- because, in realistic applications, knowledge bases (KB) not only can be complex, but can also involve a large number of individuals, that are difficult – when not impossible – to manage with the existing DLMS ABoxes, due to their lack of efficiency in dealing with large amounts data, often it is better to manage large portions of data by means of a DBMS;
- as [Borgida and Brachman, 1993] mentions, KB based on DL are often used in applications where they need access to large amounts of data stored in *already existing* databases;
- as observed in [Bresciani, 1994, Bresciani, 1992], the task of acquiring knowledge for a real knowledge based application often includes a great amount of raw data collecting; for this subtask instead of using an ABox often it is better to use databases.

In particular we faced these problems when we were developing a large natural language system prototype [Bresciani, 1992], whose domain and linguistic model were represented using LOOM [MacGregor, 1991]. A first implementation of the ideas here presented is currently used in an enhanced version of this prototype, capable of dealing with thousands of individuals.

In such applications it is very important that the database can be queried from the DLMS in a way completely transparent to the user. This call for a semantically well founded linking between the DL knowledge base and the database. This can be obtained by *coupling* DLMS and DBMS [Borgida and Brachman, 1993]: primitive concepts and relations in a KB are made to correspond respectively to unary and binary tables in a DB. In [Borgida and Brachman, 1993] two possible way to couple DLMS and DBMS are proposed:

- *loose coupling*, that requires a pre-loading of the data from the DB into the KB;
- *tight coupling*, that implements a *on demand* access to the DB;

but in the system there presented only the loose coupling paradigm is implemented [Devanbu, 1993, Borgida and Brachman, 1993].

Instead, our system is based on tight coupling, allowing the following advantages:

- complex compound conjunctive queries involving unary and binary predicates can be done;
- no memory space is wasted in the DLMS in order to keep descriptions of DBMS data;
- answers are given on the basis of the current state of the KB and the DB, without needing periodical updating of the KB with new or modified data from the DB.

Our technique [Bresciani, 1994] will be in the following briefly reviewed. This approach is here extended with the possibility of mapping a wider set of DL *concepts* into DB *views*: in this way less restrictions about the form of the KB are necessary.

2 TBox, ABox and DBox

The basic idea of our approach is to extend the traditional DL ABox with a *DBox*,[1] by which the standard TBox/ABox architecture is coupled with one or more, possibly heterogeneous and distributed, databases, so that the user can make queries to this extended system without any concern on which DB or the KB has to be accessed.

A mapping – called PM (see section 3) – between the TBox and the DBox is needed. Therefore, a *knowledge base* $\mathcal{KB} = \langle \mathcal{T}, \mathcal{W}, \mathcal{D}, PM \rangle$ [Bresciani, 1994] is formed by a *terminology* $\mathcal{T}$ and a *world description* $\mathcal{W}$ as usual [Nebel, 1990], plus a *data base* $\mathcal{D}$ and the mapping function PM. A uniform query answering function to $\mathcal{KB}$, based on the two distinct complete query answering functions (one for the ABox and one for the DBox), can be implemented. For the sake of simplicity, it will be assumed here that $\mathcal{D}$ is represented by means of a relational database, and queries to the DBox can be done in SQL.

3 Coupling

Coupling the terminology $\mathcal{T}$ with the data base $\mathcal{D}$ corresponds to associating some terms (concepts and roles) of $\mathcal{T}$ with tables or views in the DB. The coupling of $\mathcal{T}$ with $\mathcal{D}$ is performed in two steps. First, a partial mapping PM between primitively defined terms and the tables in the DB must be given. Giving a mapping of a primitively defined term into a DB-table corresponds to giving its extension in the DB. Let the terms for which PM is defined be called $\mathcal{D}$-terms. Then, using PM, also non-primitively defined concepts can be recursively mapped into views of the DB. If the (expanded) definition of a non-primitively defined concept contains both $\mathcal{D}$-terms and non-$\mathcal{D}$-terms, the view in which the concept is mapped does not contain all the instances of the concept. Therefore, non-primitively defined concepts with (expanded) definition containing both $\mathcal{D}$-terms and non-$\mathcal{D}$-terms cannot be completely managed in our system. Thus, the following constraints must be imposed on $\mathcal{KB}$:

1. Every table in $\mathcal{D}$ must correspond to one primitively defined term in $\mathcal{T}$, called $\mathcal{D}$-term; $\mathcal{D}$-terms cannot be used in the (expanded) definition of any primitively defined term in $\mathcal{T}$.
2. The (expanded) definitions of non-primitively defined concepts of $\mathcal{T}$ must contain only $\mathcal{D}$-terms or no $\mathcal{D}$-term at all.

The aim of the constraint 1 is to avoid any need of consistency checking in case of conflicts between defining and defined concepts. If, to ensure the avoidance of such conflicts, an exhaustive checking – that could involve also the extensional analysis of DBox-tables – were provided, this constraint could be released.

As mentioned, all the information needed to correctly drive the query mechanism is the association of $\mathcal{D}$-terms with the corresponding tables in the DB. Thus, defined the partial mapping:

$$PM : \mathcal{PT} \rightarrow DBtable$$

where $\mathcal{PT}$ is the set of primitive terms in $\mathcal{T}$, and *DBtable* is the set of tables in the DB, the views corresponding to non-primitive concepts can be built via a recursive partial mapping:

$$RM : \mathcal{T} \rightarrow DBtable \cup DBview$$

where *DBview* is the (virtual) set of views in the DB. RM maps DL-expressions into corresponding SQL-expressions.

In the following, to simplify the description, it is assumed that concepts are mapped into unary tables with one column called `lft`, and roles into binary tables with two columns called `lft` and `rgt`. As an example, assume that non-primitively defined concepts that contain $\mathcal{D}$-terms in their (expanded) definition are constrained to use the sub-language with the only `AND` and `SOME` operators; in this case RM can be defined as follows:[2]

[1] D for data.

[2] Note that `R` stands for a role name, i.e., for an atomic role in $\mathcal{T}$, while C and D stand for concept names or expressions. In general, the `TYPEWRITER` font will be used for atomic terms.

$RM((\texttt{AND}\ C\ D)) =$

```
SELECT DISTINCT  lft
FROM             RM(C), RM(D)
WHERE            RM(C).lft = RM(D).lft
```

if both $RM(C)$ and $RM(D)$ are defined;

$RM((\texttt{SOME R}\ D)) =$

```
SELECT DISTINCT  lft
FROM             RM(R)
WHERE            RM(R).rgt IN RM(D)
```

if both $RM(\texttt{R})$ and $RM(D)$ are defined;

$RM(\texttt{T}) = \quad PM(\texttt{T})$
if $PM(\texttt{T})$ is defined;

and

$RM(\texttt{T}) =$

```
SELECT DISTINCT  *
FROM             T_1
UNION
⋮
SELECT DISTINCT  *
FROM             T_n
```

if $M(\texttt{T}) = \{T_1, \ldots, T_n\}$, and $n > 0$.

Note that the last part of the above definition (see below for the definition of M) allows to take into account also all the tables and views corresponding to terms subsumed by T, whatever T is.

Of course RM could be extended to more general concepts, but in some cases the mapping would have to be carefully handled, due to the different semantics of DL and DB (see, e.g., the ALL and the NOT operators).

Note that, due to limitations of SQL in using subqueries, the SELECT used in the definition of RM are non exactly legal, due to the recursive application of RM. This problem can be easily overcome if a CREATE VIEW corresponds to each application of RM, and the names of the corresponding views are placed in lieu of the recursive applications of RM.[3]

The function:

$$M : \mathcal{T} \rightarrow 2^{DBtable \cup DBview}$$

used in the definition of RM returns the (possibly empty) set of tables/views necessary to retrieve all the instances (pairs) of a given concept (role) from the DB, that is:

$$M(\texttt{T}) = \{RM(x) \mid x \in subs(\texttt{T}) \wedge RM(x) \text{ is defined}\}$$

where $subs(\texttt{T})$ is the set of the terms classified under T in $\mathcal{T}$. Observe that RM and M are built starting from PM; this justifies the use of the only PM in the definition of $\mathcal{KB}$ given in section 2.

4 Query Answering

A *query* to $\mathcal{KB}$ is an expression:

$$\lambda\overline{x}.(P_1 \wedge \ldots \wedge P_n)$$

where $P_1, \ldots, P_n$ are predicates of the form $\texttt{C}(x)$ or $\texttt{R}(x, y)$, where C and R are concepts and roles in $\mathcal{T}$, respectively, and each of x and y appears in the tuple of variables $\overline{x} = \langle x_1, \ldots, x_m \rangle$ or is an individual constant in $\mathcal{W} \cup \mathcal{D}$. *Answering* a query in $\mathcal{KB}$ means finding a set $\{\overline{x}^1, \ldots, \overline{x}^m\}$ of tuples of instances such that, for each tuple $\overline{x}^i$, $\lambda\overline{x}.(P_1 \wedge \ldots \wedge P_n)[\overline{x}^i]$ holds – either explicitly or implicitly – in $\mathcal{KB}$. Let such tuples be called *answers* to the query and the set of all of them the *answer set*.

From the definition of answer to a query, it is obvious that, to avoid the generation of huge answer sets, free variables must not be used, that is, each variable appearing in $\overline{x}$ must appear also in the query body. Indeed, even stronger restrictions are adopted (see [Bresciani, 1994]).

To be answered, a query must be split into subqueries that can be answered by the two specialized query answering functions of the DLMS and the DBMS. To this end, a *marking* of all the possible atomic predicates, corresponding to the terms in $\mathcal{T}$, is needed; a term P is said to be:

- KB-marked iff $RM(P)$ is undefined;
- Mixed-marked otherwise.

These two markings reflect the fact that the instances (pairs) of P are all in $\mathcal{W}$, or part in $\mathcal{W}$ and part in $\mathcal{D}$, respectively. The case of queries in which the predicates are all KB-marked terms is trivial (it is enough to submit it to the DLMS answering function). The case of queries with also Mixed-marked predicates is more difficult.

Let a generic query be written as:

$$\lambda\overline{x}.(P_1^{KB} \wedge \ldots \wedge P_m^{KB} \wedge P_1^{M} \wedge \ldots \wedge P_n^{M})$$

where the P_i^{KB} correspond to the KB-marked terms, and the P_i^M to the Mixed-marked terms. The query can be split in the two sub-queries:

$$q^{KB} = \lambda\overline{x}_{KB}.(P_1^{KB} \wedge \ldots \wedge P_m^{KB}),$$

$$q^{M} = \lambda\overline{x}_{M}.(P_1^{M} \wedge \ldots \wedge P_n^{M}).$$

Because each predicate in q^M corresponds to a view in the DB – where the answers have to be searched in addition to those in the ABox – a translation of them into equivalent SQL queries can be provided. Of course, the views can easily be found via the recursive mapping RM. For each of the P_i^M in q^M the translation into an equivalent view is simply given by $RM(P_i^M)$. Thus, the SQL query corresponding to $q_i^M = \lambda\overline{y}.P_i^M$ – where $\overline{y}$ is the sub-tuple of $\overline{x}$ containing the only one or two variables used in P_i^M – is:

```
SELECT DISTINCT  select-body
FROM             RM(P_i^M)
WHERE            where-body
```

where the *select-body* contains $RM(P_i^M)$.lft, $RM(P_i^M)$.rgt, or both, according to the fact that P_i^M is of the kind $\texttt{C}(x)$ or $\texttt{R}(x, a)$, $\texttt{R}(a, y)$, or $\texttt{R}(x, y)$, respectively – with x and y variables, and a constant. The WHERE clause is present only in the case of $P_i^M = \texttt{R}(x, a)$ or $P_i^M = \texttt{R}(a, y)$; in this case the *where-body* is $RM(\texttt{R})$.lft $= a$ or $RM(\texttt{R})$.rgt $= a$, respectively.

[3] Of course, this requires a pre-compilation step of the DB with respect to the KB, but this is not a real overload of the presented query mechanism.

In this way n partial answer sets (one for each P_i^M) are obtained. Of course, the queries have to be submitted also to the DLMS, in case there are also $\mathcal{W}$-individuals satisfying them.

Now, it is, ideally, enough to get the intersection of all the partial answer sets obtained by processing the sub-queries of q^M and q^{KB}, but, due to the scope of the variables of the queries, this cannot be performed in a direct way: a *merging* of the results is needed. In fact, in each sub-query some of the variables in $\overline{x}$ may be unbound – that is, the proper tuple of variables $\overline{y}$ of the sub-query may be a sub-tuple of $\overline{x}$. Therefore, the corresponding answer set has to be *completed*, that is, each unbound variable in $\overline{x}$ must be made to correspond to each instance in $\mathcal{KB}$, for all the found answers, considering all the possible combinations. However, in this way huge answer sets would be generated.

To solve this problem a compact representation for the answer sets is needed. If $AS_{\overline{y}}$ is a generic partial answer set of a sub-query, and the variables of the original complete variable tuple $\overline{x}$ missing in $\overline{y}$ are $x_{p_1}, \ldots, x_{p_k}$, the completion of $AS_{\overline{y}}$ can be represented in a compact way as $AS_{\overline{x}} = \{\overline{I}^{\star} \mid \overline{I} \in AS_{\overline{y}}\}$, where each $\overline{I}^{\star}$ is equal to $\overline{I}$ except that it is lengthened by filling the k missing positions $p_1, \ldots, p_k$ with any marker, e.g., a star '$\star$', that stands for any individual in $\mathcal{KB}$. Using this representation it is possible to formulate an algorithm to efficiently cope with the *merging* of answers sets, as described in [Bresciani, 1994].

5 Conclusions

Our approach to deal with the task of integrating DLMS and DBMS, so that KB and DB can be uniformly queried from a DLMS, has been presented. With our technique, a third component – a DBox, allowing spreading extensional data among the ABox and databases – can be added to the traditional TBox/ABox architecture of DLMS. By means of the DBox it is possible to couple the DLMS with several, possibly distributed and heterogeneous, DBMS, and to use all the systems for uniformly answering queries to knowledge bases realized with this extended paradigm.

In our first implementation of the system[4] the DLMS is LOOM [MacGregor, 1991], and the database query language is SQL, but also other systems could be easily used.

At present our tool is used in a natural language dialogue system prototype [Bresciani, 1992], whose domain and linguistic knowledge is represented in a LOOM KB and, for some large amount of raw data, in an INGRES DB. Currently, our system support a more expressive query language than the one previously presented: existentially quantified conjunctions of atomic formulæ can also be used. The study of the use of even more complex query-languages is part of our future plans.

[4]Indeed, the answering algorithm has been implemented in a more sophisticated way than the one presented in section 4, including also optimizations for reducing the number of accesses to the DB (see [Bresciani, 1994]). The pre-compilation part of the method shown in section 3 – that allows dealing with non primitive concepts – is presently not yet fully implemented.

References

[Borgida and Brachman, 1993] Alex Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Proceeding of ACM SIGMOD '93*, 1993.

[Bresciani, 1992] Paolo Bresciani. Use of loom for domain representation in a natural language dialogue system. Technical Report 9203-01, IRST, Povo TN, March 1992. presented at LOOM Users Workshop, Los Angeles, March 23-24, 1992.

[Bresciani, 1994] Paolo Bresciani. Uniformly querying knowledge bases and data bases. In F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors, *Working Notes of the KI'94 Workshop: KRDB'94*, number D-94-11 in DFKI Documents, pages 58–62, Saarbrücken, Germany, September 1994.

[Buchheit *et al.*, 1994] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[Devanbu, 1993] Premkumar T. Devanbu. Translating description logics to information server queries. In *Proceedings of Second Conference on Information and Knowledge Management (CIKM '93)*, 1993.

[MacGregor, 1991] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[McKay *et al.*, 1990] Don P. McKay, Tim W. Finin, and Anthony O'Hare. The intelligent database interface. In *Proc. of AAAI-90*, pages 677–684, Boston, MA, 1990.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Heidelberg, New York, 1990.

[Pastor *et al.*, 1992] Jon A. Pastor, Donald P. McKay, and Timothy W. Finin. View-concepts: Knowledge-based access to databases. In *Proceedings of Second Conference on Information and Knowledge Management (CIKM '93)*, Baltimore, 1992.

[Tou *et al.*, 1982] F. Tou, M. Williams, R. Fikes, A. Henderson, and T. Malone. Rabbit: An intelligent database assistant. In *Proc. AAAI'82*, 1982.