Optimizing Query Rewriting in Ontology-Based Data Access

Floriana Di Pinto, Domenico Lembo, Maurizio Lenzerini, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo
DIAG, Sapienza Università di Roma
<lastname>@dis.uniroma1.it

ABSTRACT

In ontology-based data access (OBDA), an ontology is connected to autonomous, and generally pre-existing, data repositories through mappings, so as to provide a high-level, conceptual view over such data. User queries are posed over the ontology, and answers are computed by reasoning both on the ontology and the mappings. Query answering in OBDA systems is typically performed through a query rewriting approach which is divided into two steps: (i) the query is rewritten with respect to the ontology (ontology rewriting of the query); (ii) the query thus obtained is then reformulated over the database schema using the mapping assertions (mapping rewriting of the query). In this paper we present a new approach to the optimization of query rewriting in OBDA. The key ideas of our approach are the usage of inclusion between mapping views and the usage of perfect mappings, which allow us to drastically lower the combinatorial explosion due to mapping rewriting. These ideas are formalized in PerfectMap, an algorithm for OBDA query rewriting. We have experimented PerfectMap in a real-world OBDA scenario: our experimental results clearly show that, in such a scenario, the optimizations of PerfectMap are crucial to effectively perform query answering.

1. INTRODUCTION

While the amount of data managed by current information systems and the processes acting on such data continuously grow, turning these data into information, and governing both data and processes are still tremendously challenging tasks for even small organizations. The problem is complicated by the proliferation of both data sources and services that are relevant for the organization. Several factors combine to explain why such a proliferation constitutes a major problem with respect to the goal of carrying out effective data governance tasks. Firstly, the data sources are often created for serving specific applications, and they lack mechanisms for comparing, combining, and reconciling their content with the one of other data sources. Secondly, it is common practice to change a data source (e.g., a database) so as to adapt it both to specific application-dependent needs, or to new requirements, with little or no attention to keeping the documentation up-to-date. Finally, the data stored in different sources and the processes operating over them tend to be redundant, and mutually inconsistent, mainly because of the lack of central, coherent and unified coordination of data management tasks.

All the above observations show that a unified access to data and an effective governance of data are extremely difficult goals to achieve in modern information systems. Yet, both are crucial objectives for getting useful information out of the information system. This explains why organizations spend a great deal of time and money for the understanding, the governance, the curation, and the integration of data stored in different sources, and of the processes/services that operate on them, and why this problem is often cited as a key and costly Information Technology challenge faced by medium and large organizations today [1].

Ontology-based data access [16] (OBDA) is a promising direction for addressing the above challenges. OBDA is a new paradigm for accessing data, whose key idea is to resort to a three-level architecture, constituted by the ontology, the data sources, and the mapping between the two. The ontology is a formal description of the domain of interest, and is the heart of the whole system. The data sources are the repositories used in the organization by the various processes and the various applications. The mapping layer explicitly specifies the relationships between the domain concepts on the one hand and the data sources on the other hand. Notice that, the ontology and the corresponding mappings to the data sources provide a common ground for the documentation of all the data in the organization, with clear advantages for the governance and the management of the information system.

In this sense, OBDA can be seen as a form of information integration, where the usual global schema is replaced by the conceptual model of the domain of interest, formulated as an ontology expressed in a logic-based language. With this approach, the integrated view that an OBDA system provides to information consumers is not merely a data structure accommodating the various data at the sources, but a semantically rich description of the relevant concepts in the domain of interest, as well as the relationships (called "roles") between such concepts. As in information integration, one of the most important tasks that an OBDA system is required to carry out is to allow a client to query the data through the global unified view represented by the ontology. Such task is the focus of this paper. In particular, it is the responsibility of the OBDA system to compute the answer to the queries posed in terms of the ontology. Such queries are expressions that use the predicates defined in the ontology, and should be answered by reasoning on the ontology and on the mappings, and by accessing the appropriate data sources to collect the correct data to return to the client.

Query answering in OBDA is currently a hot research topic (see e.g., [5, 19, 4, 13, 2, 9]). One of the outcomes of this research is a

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EDBT/ICDT '13 March 18 - 22 2013, Genoa, Italy
Copyright 2013 ACM 978-1-4503-1597-5/13/03 ...$15.00.

detailed study of the complexity of query answering. In particular, the study of OBDA has focused on understanding which languages for the ontology and for the mappings allow query answering to be performed with reasonable computational complexity with respect to the size of the data. It is now well known that query answering can be performed efficiently only if (i) the ontology is expressed in a lightweight ontology language, and (ii) the mappings are of type GAV (Global-As-View) [6]. We remind the reader that, while in the LAV (Local-As-View) approach, every data source is described in terms of a view over the ontology, in the GAV approach, the data sources are mapped to the ontology by associating to each predicate in the ontology a query over the data sources [15]. Notably, it has been shown that, with lightweight ontology languages, and with GAV mappings, conjunctive queries are *first-order rewritable*, i.e., answering (unions of) conjunctive queries ((U)CQs) expressed over the ontology can be reduced to the evaluation of a suitable first-order query (called the perfect rewriting of the original query, and computed on the basis of the original query, the ontology, and the mapping) expressed over the data sources.

The problem. As we said before, many recent papers on OBDA have concentrated on understanding how reasoning on the ontology affects the process of query answering. For this reason, most of them refer to a simplified framework of OBDA, in which the data sources are constituted by ad-hoc data stores accommodating the instances of the concepts and the relationships of the ontology. Obviously, in such a simplified setting, mappings reduce to direct correspondences between the ontology predicates and the tables in the data stores, and do not pose any challenge to query answering.

In the present work, we address the full-fledged OBDA scenario, i.e., the situation where the data of the OBDA system are located in the sources of the organization, and a (complex) mapping is used to relate such data with the domain ontology. In order to simplify the exposition, we assume that all such data sources are wrapped into a single relational database, and the mappings are used to reconcile the view of the domain represented by the ontology with the data stored in such relational database. We observe that assuming that the data sources are represented as a single database is a realistic assumption, because, even if they are distributed and heterogeneous, they can be easily wrapped into a single relational store by means of a data federation tool. More precisely, here is the list of assumptions that we make in our work:

- the OBDA system comprises a single relational (SQL) data source;
- the mapping assertions are of type GAV, and therefore specify suitable correspondences between each predicate of the ontology and appropriate queries over the relational data source;
- the language used to express the ontology allows first-order rewritability (actually UCQ-rewritability, see later) of conjunctive queries.

As we already mentioned, under the above assumptions, the most effective approach to query answering in OBDA systems is through query rewriting. According to this approach, query answering is divided into two steps:

1. The original query is first rewritten with respect to the ontology into a new query over the ontology; we call this step the *ontology rewriting* of the query;
2. The query thus obtained is then reformulated over the source database schema using the mapping assertions; we call this step the *mapping rewriting* of the query.

While in recent years we have seen many approaches to the ontology rewriting step and its optimizations, (see, e.g., [5, 22, 23, 9]), very little has been done towards the optimization the mapping rewriting step. Note that mapping rewriting is also relevant in data integration. However, the literature on data integration has mainly focused on the LAV approach to mappings [17, 20, 12], where mapping rewriting is a form of view-based query rewriting, a well-known NP-complete problem. Differently, query answering under GAV mappings does not suffer from the intractability problem, and has been considered somehow trivial. Indeed, a naive approach to GAV mapping rewriting consists of grouping all SQL queries mapping the same ontology concept (or role) to the database into a single query, in such a way that such single query becomes a view constituted by the union of all original SQL queries. After this step, performing the mapping rewriting of an ontology query is trivial, since every concept and role can be simply rewritten in terms of the database view associated with it by the mappings. This method produces a compact final SQL query to be sent to the sources, i.e., an SQL query of size polynomial with respect to the input of the mapping rewriting step. Unfortunately, this approach is unfeasible in practice, since, even under relatively simple mappings and under empty ontologies, the final queries to be executed at the source database are too complex to be handled by current SQL engines. This is due to the fact that such engines are not able to optimize the execution of queries with complex nested expressions, such as the ones using views with complex unions.

The conclusion is that grouping all mappings relative to the same concept or role into a single SQL query is not a good idea. If, however, we keep such queries separate, i.e., we keep several mapping assertions for the same element of the ontology, and we target our rewriting towards (unions of) conjunctive queries, thus avoiding complex nested expressions in the final query, we have to face a different problem. Indeed, in this case, when processing the query atom corresponding to concept C, the mapping rewriting algorithm is forced to combine the various queries in the mappings in all possible ways into the final rewriting, and this may very well produce a final SQL query whose size is exponential with respect to the size of the initial query and the size of the mappings. For instance, if the query to be rewritten with respect to the mappings is a CQ with 10 atoms, and the predicate of each atom is mapped to 4 SQL queries, then the mapping rewriting step produces a rewritten query which is the union of 4^{10} SQL queries, i.e., over 1 million SQL disjuncts. That is, even if the ontology query is not very large, the size of the rewriting might be too large to be handled by any SQL engine.

In fact, according to our experiments in real world scenarios, the mapping rewriting phase is a bottleneck of query rewriting in OBDA, even under GAV mappings. In particular, computing the mapping rewriting of a CQ may be prohibitive if the mapping is even moderately complex. Note that this is very likely the case when the "cognitive distance" between the ontology and the database is significant, e.g., when the ontology is a domain conceptualization independent of the source database schema.

Our proposal. The starting point of our work is that the experiments we have carried out in various applications of OBDA showed that the above mentioned combinatorial explosion can be often avoided. Indeed, in many cases, several subqueries of the final union of SQL subqueries are actually redundant, i.e., contained into other subqueries. So, in principle it is possible to optimize the mapping rewriting phase by limiting the combinatorial explosion, in particular if we could exploit the knowledge on containment between the SQL subqueries used in the mapping. Note that checking containment of arbitrary SQL queries is an undecidable problem, and therefore it might seem unfeasible to base the optimization on

the ability to perform such check. Fortunately, there are two observations that make the idea applicable. Firstly, there are classes of (relatively simple) SQL queries for which the problem becomes actually decidable. Secondly, even sound and incomplete containment checking algorithms (i.e., algorithms that do not guarantee to discover all containments) make sense in this context. Indeed, discovering even a small number of containments between SQL subqueries can drastically lower the size of the final SQL query, thus making query answering over the OBDA system feasible.

Based on the above considerations, our experience in real world OBDA scenarios led us to propose the following optimizations:

1. We use intermediate predicates, called view predicates, to denote views in the mappings, and split the mapping rewriting phase into two steps, called *high-level* and *low-level* mapping rewriting, respectively. This allows us to limit the size of the final rewritten query on the one hand, and to exploit further optimizations of the rewritten query without reasoning about SQL expressions.

2. We add *view inclusions* to the OBDA specification, i.e., inclusion assertions between (projections of) the SQL queries used in mapping assertions, or, more precisely, between the corresponding view predicates. Based on such inclusions, we are able to eliminate conjunctive queries contained into other conjunctive queries of the rewritten query.

3. Although already very useful, the optimization step based on view inclusions is not sufficient in general to considerably lower the combinatorial explosion previously described, since it is applied only after generating the unoptimized mapping rewriting of the query. We therefore propose a further optimization of the whole query rewriting process, based on the use of so-called *perfect mapping assertions*. These are special assertions logically entailed by the OBDA specification, which allow for handling whole subqueries as single atoms both in the ontology rewriting and in the mapping rewriting process. We show that their usage leads to a drastic reduction of the combinatorial explosion of the mapping rewriting phase.

We combine all the above techniques in PerfectMap, an algorithm for computing the perfect rewriting of UCQs in OBDA systems. Notably, PerfectMap abstracts away from the specific language used for defining the ontology and the technique used for the ontology rewriting step, which is used as a black box. The only assumption that PerfectMap makes on such a technique is that it is able to produce a UCQ as ontology rewriting of the input query, which is a realistic assumption in current OBDA approaches [5, 9, 18, 3].

Evaluation. We have tested an implementation of the PerfectMap algorithm within the MASTRO [4] OBDA system, in which ontologies are specified in the lightweight Description Logics *DL-Lite*$_A$ [19]. In this implementation, PerfectMap makes use of the algorithm Presto [24] to realize the ontology rewriting step[1].

We have used this system in an OBDA project funded by the Italian Ministry of Economy and Finance. Within this project, we have experienced the limits of the previous techniques for OBDA query answering. Indeed, in this scenario, a large part of the queries corresponding to the real information needs of the administration could not be effectively processed by the system: within a 4-hour timeout, we could actually execute only 6 out of the 40 queries which were initially extracted by the reports used by the administration.

[1]In fact, Presto returns a non-recursive Datalog program, which in our implementation of PerfectMap is unfolded to obtain a UCQ.

An analysis of the problem highlighted that the queries produced by the mapping rewriting were highly redundant: in particular, such queries could be drastically simplified by exploiting the containment relationship between many SQL queries used in mapping assertions. We thus were able to specify a significant number of view inclusions; then, we experimented the PerfectMap-based version of MASTRO on this extended specification.

Our experimental results show that query answering based on the PerfectMap query rewriting technique outperforms the previous unoptimized technique (in particular, all the 40 relevant queries could be executed within a few seconds by the system). In the final part of the paper, we present a fragment of the ontology and the mapping used in our experimental setting, and a set of results about query answering. These results, summarized in Figure 3, clearly show the improvement in the performance due to the PerfectMap algorithm (and the different optimizations).

Structure of the paper. In the following, after some preliminaries given in Section 2, we define our notion of OBDA system specification, which includes the definition of inclusions between database views, and introduce the notion of OBDA perfect rewriting (Section 3). Then, in Section 4, we define an algorithm for computing OBDA perfect rewritings and discuss the limits of the above query rewriting technique. In Section 5, we introduce the notion of perfect mapping for an OBDA system specification, and then, in Section 6, we define the algorithm PerfectMap, which exploits perfect mappings to optimize the computation of OBDA perfect rewritings. In Section 7, we present a set of experimental results, conducted in a real-world OBDA scenario, which show how PerfectMap improves query answering over OBDA systems. We then discuss some related work in Section 8, and finally conclude the paper in Section 9.

2. PRELIMINARIES

In this section we discuss some preliminary notions on ontologies and relational databases.

In OBDA, we consider three pairwise disjoint alphabets: an alphabet $Pred$ of predicate symbols, an alphabet $Const$ of constant symbols, and an alphabet Var of variable symbols. The alphabet $Pred$ is partitioned into two alphabets: the ontology predicate alphabet Σ_O, and the database predicate alphabet Σ_R.

A *database instance*, or simply a database, is a set of ground atoms over the predicates in Σ_R and the constants in $Const$. To express queries over databases, we use SQL. Given an n-tuple of variables $\vec{x} = x_1, \ldots, x_n$, we use the notation $Q_{DB}(\vec{x})$ for an SQL query of arity n, where every x_i denotes the i-th attribute of the query[2]. When the specification of the attributes $\vec{x}$ is not necessary, we simply use Q_{DB} to denote SQL queries. Given an SQL query Q_{DB} and a database instance D, we denote by $Ans(Q_{DB}, D)$ the set of tuples computed by evaluating Q_{DB} over D.

An ontology is a conceptualization of a domain of interest expressed in terms of a formal language. Here, we consider logic-based languages, and, more specifically, Description Logics (DLs). Generally speaking, an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ expressed in a DL is formed by two distinct parts: the *TBox* $\mathcal{T}$, which comprises axioms specifying universal properties of the concepts and the roles that are relevant in the domain, and the *ABox* $\mathcal{A}$, which contains axioms about instances of concepts and roles. In OBDA, and in this paper, the only relevant component of the ontology is the TBox. Indeed, the information about the instances of concepts and roles is not pro-

[2]To simplify notation, we assume that query attribute names are obtained by renaming attributes in the target list of the SQL query with variable symbols.

vided by the ABox, but by the combination of the database and the mappings.

In the examples and experiments discussed in this paper we focus on ontologies specified in *DL-Lite*$_A$ [19], a member of the *DL-Lite* family. *DL-Lite*$_A$ allows for specifying *concepts*, representing sets of objects, *roles*, representing binary relations between objects, and *attributes*, representing binary relations between objects and values. The syntax of concept, role and attribute *expressions* in *DL-Lite*$_A$ is as follows:

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

In such rules, $A, P, U, T_1, \ldots, T_n$ belong to Σ_O. A, P, and U denote a *concept name*, a *role name*, and an *attribute name*, respectively. P^- denotes the *inverse of a role*. B and R are called *basic concept* and *basic role*, respectively. $\neg B$ (resp. $\neg Q$, $\neg U$) denotes the *negation* of a basic concept B (resp. basic role, or attribute). The concept $\exists Q$, also called *unqualified existential restriction*, denotes the *domain* of a role Q, i.e., the set of objects that Q relates to some object. Similarly, the concept $\delta(U)$ denotes the *domain* of an attribute U, i.e., the set of objects that U relates to some value. Conversely, $\rho(U)$ denotes the *range* of an attribute U, i.e., the set of values to which U relates some object. $T_1, \ldots, T_n$ are unbounded pairwise disjoint predefined *value-domains*.

A *DL-Lite*$_A$ TBox $\mathcal{T}$ is a finite set of assertions of the form

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

From left to right, assertions in the first row denote inclusions between concepts, roles, value-domains, and attributes, respectively. Assertions of the second row denote global functionality on roles and on attributes. Notice that in *DL-Lite*$_A$ TBoxes we further impose that roles and attributes occurring in functionality assertions cannot be specialized (i.e., they cannot occur in the right-hand side of inclusions).

The semantics of *DL-Lite*$_A$ is given in terms of first-order logic interpretations $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ over $\Sigma_O \cup Const$. $\Delta^\mathcal{I}$ is a non-empty domain such that $\Delta^\mathcal{I} = \Delta_V \cup \Delta_O^\mathcal{I}$, where $\Delta_O^\mathcal{I}$ is the domain used to interpret object constants in Γ_O, and Δ_V is the fixed domain (disjoint from $\Delta_O^\mathcal{I}$) used to interpret data values. $\cdot^\mathcal{I}$ is an interpretation function defined as follows:

$$\begin{array}{ll} A^\mathcal{I} \subseteq \Delta_O^\mathcal{I} & (\delta(U))^\mathcal{I} = \{\, o \mid \exists v.\, (o,v) \in U^\mathcal{I} \,\} \\ P^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_O^\mathcal{I} & (P^-)^\mathcal{I} = \{\, (o,o') \mid (o',o) \in P^\mathcal{I} \,\} \\ U^\mathcal{I} \subseteq \Delta_O^\mathcal{I} \times \Delta_V & (\exists Q)^\mathcal{I} = \{\, o \mid \exists o'.\, (o,o') \in Q^\mathcal{I} \,\} \\ (\neg B)^\mathcal{I} = \Delta_O^\mathcal{I} \setminus B^\mathcal{I} & (\rho(U))^\mathcal{I} = \{\, v \mid \exists o.\, (o,v) \in U^\mathcal{I} \,\} \\ (\neg Q)^\mathcal{I} = (\Delta_O^\mathcal{I} \times \Delta_O^\mathcal{I}) \setminus Q^\mathcal{I} & (\neg U)^\mathcal{I} = (\Delta_O^\mathcal{I} \times \Delta_V) \setminus U^\mathcal{I} \end{array}$$

Notice that each $\cdot^\mathcal{I}$ interprets the value domains $T_1, \ldots, T_n$ and each value constant c in the same way, i.e., such interpretations are fixed once and for all. An interpretation $\mathcal{I}$ satisfies a concept (resp., role) inclusion assertion $B \sqsubseteq C$ (resp., $Q \sqsubseteq R$) if $B^\mathcal{I} \subseteq C^\mathcal{I}$ (resp., $Q^\mathcal{I} \subseteq R^\mathcal{I}$). Furthermore, a role functionality assertion ($\mathsf{funct}\ Q$) is satisfied by $\mathcal{I}$ if, for each $o, o', o'' \in \Delta_O^\mathcal{I}$, we have that $(o, o') \in Q^\mathcal{I}$ and $(o, o'') \in Q^\mathcal{I}$ implies $o' = o''$. The semantics for attribute and value-domain inclusion assertions, and for functionality assertions over attributes can be defined analogously. As for the semantics of ABox assertions, we say that $\mathcal{I}$ satisfies the ABox assertions $A(a)$, $P(a, b)$ and $U(a, c)$ if $a^\mathcal{I} \in A^\mathcal{I}$, $(a^\mathcal{I}, b^\mathcal{I}) \in P^\mathcal{I}$ and $(a^\mathcal{I}, c^\mathcal{I}) \in U^\mathcal{I}$, respectively. Furthermore, *DL-Lite*$_A$ adopts the Unique Name Assumption (UNA), i.e., in every interpretation $\mathcal{I}$, and for every pair $c_1, c_2 \in Const$, if $c_1 \neq c_2$ then $c_1^\mathcal{I} \neq c_2^\mathcal{I}$.

The main mechanism for querying ontologies is through *unions of conjunctive queries (UCQs)*. Generally speaking, given a signature $\Sigma = \Sigma_P \cup Var \cup Const$, such that Σ_P is an alphabet of predicates (e.g., $\Sigma_P = \Sigma_O$), a *conjunctive query* q over Σ is a first-order query of the form $\{\vec{x} \mid \psi(\vec{x})\}$, where ψ is an expression of the form $\exists \vec{y}.\alpha_1(\vec{x}, \vec{y}) \wedge \ldots \wedge \alpha_n(\vec{x}, \vec{y})$, where $\vec{y}$ are variables from Var, $\vec{x}$ are terms from $Var \cup Const$, and each $\alpha_i(\vec{x}, \vec{y})$ is an atom with predicate from Σ_P. The number of variables in $\vec{x}$ is the arity of the query q.

A *subquery* q' of q is a CQ of the form $\{\vec{x}' \mid \exists \vec{y}'.\alpha'_1(\vec{x}', \vec{y}') \wedge \ldots \wedge \alpha'_m(\vec{x}', \vec{y}')\}$, where each atom $\alpha'_i(\vec{x}', \vec{y}')$ occurs also in q, each variable in $\vec{x}'$ occurs also in $\vec{x}$ and each variable in $\vec{y}'$ occurs also in $\vec{y}$. The difference between q and its subquery q', denoted $q - q'$, is the CQ $\{\vec{x}'' \mid \exists \vec{y}''.\phi(\vec{x}'', \vec{y}'')\}$, where ϕ is obtained deleting the atoms of q' from q, and $\vec{x}''$ (resp. $\vec{y}''$) contains those variables of $\vec{x}$ (resp. of $\vec{y}$) that occur in the remaining atoms. An UCQ over Σ is a set of CQs over Σ of the same arity.

The notion of evaluation of UCQs over a first-order interpretation is given in the standard way. More precisely, given an interpretation $\mathcal{I}$ for $\Sigma_P \cup Const$, a CQ $q(\vec{x})$ of arity n over Σ is interpreted in $\mathcal{I}$ as the set $q^\mathcal{I}$ of tuples $\vec{c} \in Const^n$ such that, when we substitute the variables $\vec{x}$ with the constants $\vec{c}$, the formula $\exists \vec{y}.\alpha_1(\vec{x}, \vec{y}) \wedge \ldots \wedge \alpha_n(\vec{x}, \vec{y})$ evaluates to true in $\mathcal{I}$. A UCQ Q over Σ is interpreted in $\mathcal{I}$ as the union $Q^\mathcal{I}$ of the interpretations of all CQs contained in Q.

Finally, given a CQ cq of the form $\{\vec{x} \mid \psi(\vec{x})\}$ and a tuple $\vec{t}$ of constants, we denote by $cq(\vec{t})$ the first-order sentence obtained from ψ by replacing the free variables $\vec{x}$ with $\vec{t}$.

3. OBDA SYSTEM SPECIFICATION

We extend the classical notion of OBDA specification by adding inclusions between the database views appearing in the mapping. Therefore, while traditional OBDA is based on three components, in our approach, an OBDA system is characterized by four components: the ontology, the source schema, the mapping between the source schema and the ontology, and a set of view inclusions. The purpose of this section is to present the formal definition of the syntax and the semantics of this new notion of OBDA specification.

As we said in the introduction, in ODBA specifications, we limit our attention to GAV mapping assertions. We first define the notion of mapping assertions in general. A *mapping assertion* m is an expression of the form

$$Q_{DB}(\vec{x}^1) \rightsquigarrow cq(\vec{x}^2)$$

where

- $Q_{DB}(\vec{x}^1)$ is an SQL query over the alphabet Σ_R – such query is called the *view* associated to m;
- $cq(\vec{x}^2)$ is a conjunctive query over the alphabet Σ_O;

A *GAV mapping assertion* is simply a mapping assertion in which every variable x occurring in $\vec{x}^2$ also occurs in $\vec{x}^1$. Note that this condition guarantees that the mapping is indeed of type GAV. In fact, a LAV mapping assertion differs from a GAV assertion because it contains only atomic queries on the left-hand side, and may contain variables in the right-hand side of the assertion that do not appear in the left-hand side (existential variables). Such variables are exactly those which are not allowed by the above condition.

In the following, we assume that no variable occurs more than once in $\vec{x}^1$ or $\vec{x}^2$. Also, when we talk about the arity of m, we simply mean the arity of $Q_{DB}(\vec{x}^1)$.

The intuitive meaning of a mapping assertion m of the above form is that all the tuples satisfying view Q_{DB} also satisfy the ontology query cq. Note that this implies that we are assuming that

the source database directly stores the constants denoting the instances of the concepts and the roles in the ontology. More sophisticated approaches to OBDA do not make such a simplified assumption. Rather, they are based on the idea that the objects denoting instances of concepts are not stored in the database, but are constructed through the mappings starting from the values of the data sources (cf.[19]). The whole approach presented in this paper can be in fact straightforwardly generalized to this situation. This has not been done here for ease of exposition.

For every mapping assertion m, we introduce a new predicate symbol v_m (called the *view name of* m), drawn from an alphabet which is pairwise disjoint with the alphabets $Pred$, $Const$, Var. The arity of the predicate symbol v_m is simply the arity of m.

We are now ready to define the syntax of an OBDA specification. As we said before, while traditionally an OBDA a specification is constituted by the ontolgy, the source schema and the mapping, here we introduce a new notion of OBDA specification, whose form is defined as follows.

DEFINITION 1. **(Syntax of OBDA specification)** *An* OBDA specification $\mathcal{B}$ *is a quadruple* $\langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$, *where*

- $\mathcal{T}$ *is a TBox, called the ontology of* $\mathcal{B}$,
- $\mathcal{S}$ *is a database schema, called the data source of* $\mathcal{B}$,
- $\mathcal{M}$ *is a set of mapping assertions between* $\mathcal{S}$ *and* $\mathcal{T}$, *called the* mapping *of* $\mathcal{B}$,
- $\mathcal{C}$ *is a set of* view inclusion dependencies, *called the view inclusions of* $\mathcal{B}$, *where each view inclusion dependency (or simply* view inclusion*) is an expression of the form*

$$v_1[i_1, \ldots, i_k] \subseteq v_2[j_1, \ldots, j_k]$$

with v_1 *and* v_2 *view names,* $i_1, \ldots, i_k$ *sequence of pairwise distinct integers ranging from 1 to the arity of* v_1, *and* $j_1, \ldots, j_k$ *sequence of pairwise distinct integers ranging from 1 to the arity of* v_2.

Let us now show an example of OBDA specification, which is a simplified version of the one used in the real world project mentioned in the introduction.

EXAMPLE 1. *The OBDA system used in this example is constituted by the following DL-Lite*$_A$ *ontology:*

$\mathit{PublicOrg} \sqsubseteq \mathit{Organization}$	$\mathit{PublicDep} \sqsubseteq \mathit{PublicOrg}$
$\exists \mathit{worksWith} \sqsubseteq \mathit{Organization}$	$\exists \mathit{worksWith}^- \sqsubseteq \mathit{Organization}$
(funct *name*)	(funct *address*)

The concepts in the ontology are Organization, PublicOrg, *denoting public organizations, and* PublicDep, *denoting public departments. The axioms in the first row state that public organizations are particular organizations, and public departments are particular public organizations. The role* worksWith *relates organizations that work together (second row). The attributes of the ontology are* name, address, prjName. *The axioms in the third row specify that* name *and* address *are functional.*

The following is the source schema of our OBDA example:

```
Dept_MinistryA(dep_id,dep_name)
Works_On(dep_id,proj_name)
Dept_MinistryB(dep_id,dep_addr)    Cooperate(dept1,dept2)
```

Table `Dept_MinistryA` *(*`Dept_MinistryB`*) stores data about departments belonging to MinistryA (MinistryB). Table* `Works_On` *stores data about projects carried out by departments, and* `Cooperate` *specifies pairs of departments which cooperate.*

The mapping between the sources and the ontology is expressed in terms of the following mapping assertions:

```
SELECT dep_id AS x, dep_name AS y FROM Dept_MinistryA
```
$\rightsquigarrow \{x, y \mid \mathsf{PublicDep}(x) \wedge \mathsf{name}(x, y)\}$

which relates table `Dept_MinistryA` *to the instances of concept* PublicDep *and their names;*

```
SELECT dep_id AS x, dep_addr AS y FROM Dept_MinistryB
```
$\rightsquigarrow \{x, y \mid \mathsf{PublicDep}(x) \wedge \mathsf{address}(x, y)\}$

which relates table `Dept_MinistryB` *to the instances of* PublicDep *and their addresses;*

```
SELECT w1.dep_id as x, w2.dep_id as y, w2.proj_name as z
FROM Works_On w1, Works_On w2, Dept_MinistryA d1,
Dept_MinistryA d2
WHERE d1.dep_id=w1.dep_id AND d2.dep_id=w2.dep_id
AND w1.proj=w2.proj AND w1.dep_id <> w2.dep_id
```
$\rightsquigarrow \{x, y, z \mid \mathsf{worksWith}(x, y) \wedge \mathsf{prjName}(x, z) \wedge \mathsf{prjName}(y, z)\}$

which specifies that departments from MinistryA that work on the same project actually work together, and therefore are mapped to worksWith, *together with the indication of the project they work for (attribute* prjName*);*

```
SELECT d1.dep_id as x, d2.dep_id as y
FROM Cooperate c, Dept_MinistryB d1, Dept_MinistryB d2
WHERE c.dept1=d1.dep_id AND c.dept2=d2.dep_id
```
$\rightsquigarrow \{x, y \mid \mathsf{worksWith}(x, y)\}$

which maps the notion of cooperation represented in table `Cooperate` *to the role* worksWith *in the ontology.*

Now, let v1, v2, v3, v4 *be the view names associated to the mapping assertions shown above, in the same order they have been presented. Note that* v1, v2, v4 *have arity 2, whereas* v3 *has arity 3. By taking into account their meaning, it is easy to see that the following view inclusions are part of the OBDA specification:*

$$v3[1] \subseteq v1[1], \quad v3[2] \subseteq v1[1], \quad v4[1] \subseteq v2[1], \quad v4[2] \subseteq v2[1]. \qquad \square$$

We now turn our attention to the semantics of OBDA specifications. We start with two preliminary notions, namely view inclusion satisfaction, and mapping assertion satisfaction. Let

$$m_1 : Q_{DB}(\vec{x}) \rightsquigarrow cq \qquad m_2 : Q'_{DB}(\vec{y}) \rightsquigarrow cq'$$

be two mapping assertions with associated view names v_{m_1}, v_{m_2}, respectively. Let D be a database instance for the schema $\mathcal{S}$. The inclusion $v_{m_1}[i_1, \ldots, i_k] \subseteq v_{m_2}[j_1, \ldots, j_k]$ is *satisfied* by D if the projection of $Ans(Q_{DB}, D)$ over the attributes $x_{i_1}, \ldots, x_{i_k}$ is contained in the projection of $Ans(Q'_{DB}, D)$ over the attributes $y_{j_1}, \ldots, y_{j_k}$.

Given a database schema $\mathcal{S}$ and a set of inclusion dependencies $\mathcal{C}$ between views over $\mathcal{S}$, a database instance D for $\mathcal{S}$ is called *legal for* $\mathcal{C}$ if D satisfies every inclusion in $\mathcal{C}$.

Let $\mathcal{I}$ be a first-order interpretation over $\Sigma_O \cup Const$, and let D be a database instance for Σ_R. A mapping assertion $Q_{DB}(x_1, \ldots, x_n) \rightsquigarrow cq(x_{i_1}, \ldots, x_{i_k})$, is *satisfied by* $\mathcal{I}$ *and* D if, for every n-tuple of constants $\langle c_1, \ldots, c_n\rangle \in Ans(Q_{DB}, D)$, we have that $\mathcal{I} \models cq(\vec{c}\,')$, where $\vec{c}\,' = \langle c_{i_1}, \ldots, c_{i_k}\rangle$.

Based on the above notions, we are now ready to specify the semantics of OBDA specifications. We do so by defining the notion of model of an OBDA specification.

DEFINITION 2. **(Semantics of OBDA specification)** *Let* $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$ *be an OBDA specification, and let* D *be a source*

database that is legal for $\mathcal{C}$. A model *for $\langle \mathcal{B}, D\rangle$ is a first-order interpretation $\mathcal{I}$ of $\Sigma_O \cup$ Const such that (i) $\mathcal{I}$ is a model for $\mathcal{T}$; (ii) every mapping assertion in $\mathcal{M}$ is satisfied by $\mathcal{I}$ and D.*

Note that, in general, several (even an infinite number of) models exist for a given OBDA specification $\mathcal{B}$, and a database instance D. In the following, we denote by $Models(\mathcal{B}, D)$ the set of models for $\langle \mathcal{B}, D\rangle$.

The main task to be carried out by an OBDA system is to answer queries expressed over the ontology. Given a query Q over the ontology, the answer to Q that the system should compute is the set of the so-called certain answers, where the *certain answers to Q over $\mathcal{B}$ and D*, denoted by *CertAns*$(Q, \mathcal{B}, D)$, are the tuples of constants $\vec{c}$ such that $\mathcal{I} \models Q(\vec{c})$ for each $\mathcal{I} \in Models(\mathcal{B}, D)$.

In our work, we concentrate on the rewriting approach to computing the certain answers. The basic notion underlying this approach is the one of OBDA perfect rewriting.

DEFINITION 3. *Let $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$ be an OBDA specification, and let Q be a query over $\mathcal{T}$. A query Q_{DB} over $\mathcal{S}$ is an* OBDA perfect rewriting *(or, simply, perfect rewriting) of Q under $\mathcal{B}$ if, for every source database D legal for $\mathcal{C}$, we have that CertAns$(Q, \mathcal{B}, D) =$ Ans(Q_{DB}, D).*

In other words, a perfect rewriting of a query Q expresed over the ontology is a query over the sources that, when evaluated over a legal source database D, returns exactly the certain answers to Q. In the rest of this paper, we assume that the language used to express the ontology of our OBDA specification is UCQ-rewritable, defined as follows.

DEFINITION 4. *Let $\langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$ be an OBDA specification, and let Q, Q' be two queries over $\mathcal{T}$. Q' is an* ontology rewriting *of Q under $\langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$, if for every database D of $\mathcal{S}$ we have that CertAns$(Q, \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle, D) =$ CertAns$(Q', \langle \emptyset, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle, D)$.*

Intuitively, an ontology rewriting of a query Q under $\langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$ is another query Q' which incorporates all the relevant properties of the ontology axioms, so that, by using Q', we can compute the certain answers of Q by ignoring the TBox $\mathcal{T}$. Based on the notion of ontology rewriting, we now formally define what it means for an ontology language to be UCQ-rewritable.

DEFINITION 5. *An ontology language $\mathcal{L}$ is said to be* UCQ-rewritable *if, for every OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$, with $\mathcal{T}$ expressed in $\mathcal{L}$, and for every UCQ Q over $\mathcal{T}$, one can effectively compute a UCQ over $\mathcal{T}$ which is an ontology rewriting of Q under $\mathcal{B}$.*

We observe that *DL-Lite$_A$* is UCQ-rewritable. Other examples of UCQ-rewritable languages can be found in [5, 9, 3].

We conclude this section with an observation on view inclusions. A natural question to ask is how such inclusions are determined in practice. One obvious method is manually specifying the view inclusions, based on an analysis of the source relations and the meaning of the views. This is what we did in Example 1. This approach might be very costly, and error prone. An alternative method is based on the idea of trying to automatically derive them with the help of a first-order theorem prover. Of course, since SQL query containment is in general undecidable, termination is not guaranteed. Nevertheless, we have experimented the above idea, and our experiments have shown that state-of-the-art theorem provers are effectively able to check containment between SQL/relational algebra expressions in practice. More specifically, given a set of mapping assertions (in particular, the mappings generated in the OBDA project described in Section 7), we have systematically checked containment between all possible projections of the views used in such mappings, by translating the above problem to unsatisfiability of a first-order sentence, and then solving this problem through the Vampire theorem prover [21]. Notably, Vampire was able to solve all the above reasoning problems in a few seconds.

4. USING VIEW INCLUSIONS FOR QUERY REWRITING IN OBDA

In this section we describe a new algorithm for the computation of perfect rewritings of UCQs in OBDA systems with GAV mappings. In the rest of the paper, if not otherwise specified, we implicitely refer to OBDA specifications $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$, where $\mathcal{T}$ is expressed in a UCQ-rewritable ontology language, and $\mathcal{M}$ is a GAV mapping.

We know that, for OBDA specifications of this type, the OBDA perfect rewriting of a UCQ Q over $\mathcal{T}$ under $\mathcal{B}$ can be obtained in two steps: (i) compute the ontology rewriting Q' of Q under $\mathcal{B}$ (by using only the ontology $\mathcal{T}$), where Q' is a UCQ over $\mathcal{T}$; (ii) compute the mapping rewriting of Q' under EO (by using the mapping $\mathcal{M}$), thus obtaining an SQL query over $\mathcal{S}$.

Compared to existing approaches, the algorithm we propose performs an important optimization in the mapping rewriting step, based on the view inclusions that are present in the OBDA specification. Essentially, the optimization exploits the knowledge about view inclusions in order to eliminate redundant queries from the rewriting. The benefit will be that the size of the UCQ representing the rewriting will be smaller than in the traditional approaches, and the evaluation time of the final SQL will be smaller too.

Hereafter, each mapping assertion m of the form $Q_{DB}(\vec{x}^1) \rightsquigarrow cq(\vec{x}^2)$ in the set $\mathcal{M}$ with associated view v_m will be split into two parts:

- one *low-level mapping assertion* of the form $Q_{DB}(\vec{x}) \rightsquigarrow \{\vec{x} \mid v_m(\vec{x})\}$, and
- a set of *high-level mapping assertions* of the form

  $$\{\vec{x} \mid v_m(\vec{x})\} \rightsquigarrow \{\vec{x}_1 \mid \alpha_1(\vec{x}_1)\} \; \ldots \; \{\vec{x} \mid v_m(\vec{x})\} \rightsquigarrow \{\vec{x}_n \mid \alpha_n(\vec{x}_n)\}$$

  where $\alpha_i(\vec{x}_i)$ is an atom of $cq(\vec{x}^2)$ and every variable in $\vec{x}_i$ occurs in $\vec{x}$, for $1 \leq i \leq n$.

We will denote with $\mathcal{M}_L$ (resp. $\mathcal{M}_H$) the set of low-level (resp. high-level) mapping assertions obtained from the mapping assertions in the set $\mathcal{M}$.

EXAMPLE 2. *Consider the first mapping assertion given in Example 1 associated with view v1. The low-level mapping assertion for this case is*

```
SELECT dep_id AS x, dep_name AS y
FROM Dept_MinistryA
```

$\rightsquigarrow \{x, y \mid \mathsf{v1}(x, y)\}$

while the high-level mapping assertions are:

$$\{x, y \mid \mathsf{v1}(x, y)\} \rightsquigarrow \{x \mid \mathsf{PublicDep}(x)\}$$
$$\{x, y \mid \mathsf{v1}(x, y)\} \rightsquigarrow \{x \mid \exists z.\mathsf{name}(x, z)\}. \qquad \square$$

We notice that the use of view predicates and high-level mappings, besides allowing the optimizations described in the following, may per se help to limit the size of queries produced in the mapping rewriting phase. Indeed, even by applying classical unfolding techniques (cf. step 2 of the algorithm *OBDA-Rewrite$_{ID}$* given below), different atoms of the query to unfold might be

rewritten into the same atom. For example, the query $\{x, y \mid \mathsf{PublicDep}(x) \wedge \mathsf{name}(x, y)\}$, posed over the OBDA system of Example 2, is unfolded according to the high-level mappings into the query $\{x, y \mid v1(x, y)\}$.

We are ready to present the algorithm, that we call *OBDA-Rewrite*$_{ID}$.

Algorithm *OBDA-Rewrite*$_{ID}(\mathcal{B}, Q)$
Input: An OBDA system specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C} \rangle$,
with $\mathcal{T}$ expressed in an UCQ-rewritable language $\mathcal{L}$;
a UCQ Q over $\mathcal{T}$
Output: An SQL query over $\mathcal{S}$
begin
Compute a UCQ Q_1 that is an ontology rewriting of Q under $\mathcal{B}$;
$Q_2 = \mathit{HighMappingRewrite}(Q_1, \mathcal{M}_H)$;
$Q_3 = \mathit{ID\text{-}Optimize}(Q_2, \mathcal{C})$;
$Q_{SQL} = \mathit{LowMappingRewrite}(Q_3, \mathcal{M}_L)$;
return Q_{SQL};
end

The algorithm is constituted by four steps, which we now discuss.

Step 1: a UCQ Q_1 over $\mathcal{T}$ is computed, which is the ontology rewriting of Q under $\mathcal{B}$. Algorithm *OBDA-Rewrite*$_{ID}$ does not make any assumption on the way in which such a rewriting is computed. Since we assume that the language $\mathcal{L}$ is UCQ-rewritable (cf. Section 2), we can rely on any method for this task. Examples of ontology rewriting algorithms for UCQ-rewritable languages are [5, 9, 18, 3].

Step 2: Q_1 is rewritten into query Q_2 using the high-level mappings $\mathcal{M}_H$ by means of the function *HighMappingRewrite*. This function performs what is called an unfolding of a query in the data integration jargon [15, 25, 11]. In particular, for every disjunct in Q_1, a new disjunct is introduced in Q_2, by substituting every atom of q_2 with a view predicate that the mapping $\mathcal{M}_H$ associates to α_i (cf. [19]). Note that Q_2 is a UCQ over the view names.

Step 3: Q_2 is *minimized* by the algorithm *ID-Optimize*, on the basis of the view inclusions in $\mathcal{C}$, thus obtaining a new query Q_3 over the view names. Algorithm *ID-Optimize* is discussed later in the section.

Step 4: Q_3 is rewritten by *LowMappingRewrite* into an SQL query over the source database using the mapping $\mathcal{M}_L$. This step is similar to the unfolding in step 2, with the difference that the view names are now substituted by the corresponding SQL queries.

We now describe in some detail the algorithm *ID-Optimize*. Intuitively, the algorithm proceeds as follows: it first minimizes every single CQ in the UCQ, and then performs a further optimization on the whole UCQ.

Algorithm *ID-Optimize*$(Q, \mathcal{C})$
Input: UCQ Q over the view names; set of inclusions $\mathcal{C}$
Output: UCQ Q' over the view names
begin
$Q' = Q$;
for each CQ $q \in Q'$ **do** $q = \mathit{DeleteRedundantAtoms}(q, \mathcal{C})$;
for each pair of distinct CQs q_1, q_2 of Q' **do**
if $\mathsf{contained}(q_1, q_2, \mathcal{C})$
then eliminate q_1 from Q';
return Q';
end

The function $\mathit{DeleteRedundantAtoms}(q, \mathcal{C})$ eliminates from the conjunctive query q the atoms that are redundant with respect to the set $\mathcal{C}$ of view inclusions, i.e., the atoms that are implied, under $\mathcal{C}$, by other atoms of q. The obtained query is obviously equivalent to q. For example, if q is the query $\{x \mid \exists y. v_1(x) \wedge v_2(x, y)\}$ and $v_2[2] \subseteq v_1[1]$ is an inclusion in $\mathcal{C}$, then $\mathit{DeleteRedundantAtoms}(q, \mathcal{C})$ minimizes q into the query $\{x \mid \exists y. v_2(x, y)\}$.

After this first minimization, for every pair of CQs q_1, q_2 in Q', the algorithm checks containment of q_1 in q_2 under the inclusion dependencies $\mathcal{C}$, through the procedure $\mathsf{contained}$. Formally, let Σ be the signature for q_1 and q_2, q_1 is contained in q_2 under $\mathcal{C}$ if, for every database instance D for Σ, $\mathit{Ans}(q_1, D) \subseteq \mathit{Ans}(q_2, D)$. This check is in fact non-trivial, and can be realized in various ways. In our implementation (cf. Section 6 and Section 7), $\mathsf{contained}(q_1, q_2, \mathcal{C})$ is realized by verifying whether q_1 is contained in *ID-rewrite*$(q_2, \mathcal{C})$, where *ID-rewrite* is a query rewriting algorithm given in [3]. This algorithm rewrites q_2 according to $\mathcal{C}$, i.e., it produces a UCQ containing all queries implied by q_2 under $\mathcal{C}$. For example, let q_2 be the query $\{x \mid \exists y. v_1(x) \wedge v_2(x, y)\}$ and $v_3[2] \subseteq v_1[1]$ an inclusion in $\mathcal{C}$, *ID-rewrite*$(q_2, \mathcal{C})$ returns the UCQ $\{q_2, \{x \mid \exists y. v_3(x) \wedge v_2(x, y)\}\}$. To check containment of q_1 in q_2 under the inclusions in $\mathcal{C}$ is then sufficient to check standard query containment of q_1 in any CQ returned by *ID-rewrite*$(q_2, \mathcal{C})$.

Let us now discuss an example of application of the algorithm *OBDA-Rewrite*$_{ID}$.

EXAMPLE 3. *Consider the OBDA system of Example 1, and the following conjunctive query Q:*

$$\{x, y \mid \mathsf{PublicOrg}(x) \wedge \mathsf{WorksWith}(x, y) \wedge \mathsf{PublicOrg}(y)\}$$

Assume that the first step of the algorithm OBDA-Rewrite$_{ID}$ *produces the following ontology rewriting (e.g., by the algorithm* $\mathsf{PerfectRef}$ *[5], or REQUIEM [18]):*

$$\{x, y \mid \mathsf{PublicOrg}(x) \wedge \mathsf{WorksWith}(x, y) \wedge \mathsf{PublicOrg}(y)\}$$
$$\{x, y \mid \mathsf{PublicOrg}(x) \wedge \mathsf{WorksWith}(x, y) \wedge \mathsf{PublicDep}(y)\}$$
$$\{x, y \mid \mathsf{PublicDep}(x) \wedge \mathsf{WorksWith}(x, y) \wedge \mathsf{PublicOrg}(y)\}$$
$$\{x, y \mid \mathsf{PublicDep}(x) \wedge \mathsf{WorksWith}(x, y) \wedge \mathsf{PublicDep}(y)\}$$

Then, HighMappingRewrite returns the UCQ given by the set of CQs below:

$$\{x, y \mid \exists u, v, z. \mathsf{v1}(x, u) \wedge \mathsf{v3}(x, y, v) \wedge \mathsf{v1}(y, z)\}$$
$$\{x, y \mid \exists u, v, z. \mathsf{v1}(x, u) \wedge \mathsf{v3}(x, y, v) \wedge \mathsf{v2}(y, z)\}$$
$$\{x, y \mid \exists u, v, z. \mathsf{v2}(x, u) \wedge \mathsf{v3}(x, y, v) \wedge \mathsf{v1}(y, z)\}$$
$$\{x, y \mid \exists u, v, z. \mathsf{v2}(x, u) \wedge \mathsf{v3}(x, y, v) \wedge \mathsf{v2}(y, z)\}$$
$$\{x, y \mid \exists u, z. \mathsf{v1}(x, u) \wedge \mathsf{v4}(x, y) \wedge \mathsf{v1}(y, z)\}$$
$$\{x, y \mid \exists u, z. \mathsf{v1}(x, u) \wedge \mathsf{v4}(x, y) \wedge \mathsf{v2}(y, z)\}$$
$$\{x, y \mid \exists u, z. \mathsf{v2}(x, u) \wedge \mathsf{v4}(x, y) \wedge \mathsf{v1}(y, z)\}$$
$$\{x, y \mid \exists u, z. \mathsf{v2}(x, u) \wedge \mathsf{v4}(x, y) \wedge \mathsf{v2}(y, z)\}$$

ID-Optimize then computes the two CQs $\{x, y \mid \exists z. \mathsf{v3}(x, y, z)\}$ *and* $\{x, y \mid \mathsf{v4}(x, y)\}$. *The final rewriting returned by OBDA-Rewrite*$_{ID}$ *is simply the union of the SQL queries associated to* $\mathsf{v3}$ *and* $\mathsf{v4}$. □

On the basis of the correctness of the unfolding steps in the algorithms, and of the function $\mathsf{contained}$, which in turn relies on the correctness of the *ID-rewrite* algorithm, showed in [3], one can easily prove the following theorem.

THEOREM 1. *Let* $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C} \rangle$ *be an OBDA system where* $\mathcal{T}$ *is specified in a UCQ-rewritable language, and let Q be a UCQ over* $\mathcal{T}$. *The algorithm OBDA-Rewrite*$_{ID}(\mathcal{B}, Q)$ *returns a perfect rewriting of Q under* $\mathcal{B}$.

The effectiveness of *OBDA-Rewrite*$_{ID}$ in optimizing query rewriting relies essentially on the fact that the query produced by *ID-Optimize* can be significantly smaller than the input query Q, even if only few inclusions are declared and/or used by the algorithm. As a consequence, the SQL query returned by *OBDA-Rewrite*$_{ID}$ can be evaluated much more efficiently than the query it would return without the *ID-Optimize* step. In Figure 3 of Section 7, we report some data about the experiments we carried out in using our OBDA approach. We anticipate some results

here: the table shows that the size of the rewritten query computed by using *ID-Optimize* (column ND under (ii)) is in general much smaller than the one computed without applying such optimization (column ND under (i)). The same applies to the time needed for evaluating the rewritten query at the sources (columns ET under (i) and (ii)). On the other hand, there is a price to pay for this optimization: the treatment of view inclusions may make the overall rewriting process longer with respect to the non-optimized version (see columns RT under (i) and (ii) in Figure 3). To manage this issue, in the next two sections we introduce the notion of perfect mapping, and we devise new optimizations based on such notion.

5. PERFECT MAPPINGS

The use of algorithm $\textit{OBDA-Rewrite}_{ID}$ in real word application shows that, while the treatment of view inclusions allows for significantly reducing the size of the rewriting produced, the combinatorial explosion due to the mapping rewriting step is not actually avoided. This is due to the fact that the optimization introduced by the use of view inclusions is applied only after the whole unoptimized UCQ, which represents the reformulation of the initial query over the view names, is produced. Note that, as a consequence, the optimization introduced based on view inclusions appears of no use if the unoptimized rewritten query has a very large size.

In the rest of the paper, we present an approach to address this problem. The main ingredient of such an approach is the notion of *perfect mapping assertion*, which is introduced in this section. The intuitive meaning of a perfect mapping assertion is that it explicitely states which is a perfect rewriting of a conjunctive query (the one "covered" by the perfect mapping) over the ontology. We will see that, by exploiting perfect mapping assertions, we are able to adopt a kind of semantic caching approach for speeding up the evaluation of new queries, in the case where they contain subqueries "covered" by perfect mapping assertions.

DEFINITION 6. *A* perfect mapping assertion *for an OBDA specification $\mathcal{B}$ is a mapping assertion $Q_{DB} \rightsquigarrow cq$ such that Q_{DB} is a perfect rewriting of cq under $\mathcal{B}$. A* perfect mapping for $\mathcal{B}$ *is a set of perfect mapping assertions for $\mathcal{B}$.*

We illustrate below an example of perfect mapping assertion.

EXAMPLE 4. *In the following perfect mapping assertion, the conjunctive query in the right-hand side is the query Q_1 of Example 3, and the SQL query in the left-hand side is the perfect rewriting of Q_1 shown in the same example.*

```
SELECT w1.dep_id as x,w2.dep_id as y
FROM Works_On w1, Works_On w2, Dept_MinistryA d1,
Dept_MinistryA d2
WHERE d1.dep_id=w1.dep_id AND d2.dep_id=w2.dep_id
AND w1.proj=w2.proj AND w1.dep_id<>w2.dep_id
UNION
SELECT d1.dep_id as x, d2.dep_id as y
FROM Cooperate c, Dept_MinistryB d1, Dept_MinistryB d2
WHERE c.dept1=d1.dep_id AND c.dept2=d2.dep_id
```

$\rightsquigarrow \{x, y \mid \textsf{PublicOrg}(x), \textsf{worksWith}(x, y), \textsf{PublicOrg}(y)\}$ □

Notice that perfect mappings are constituted by arbitrary (i.e., not necessarily GAV) mapping assertions, where existential variables may appear in the CQs over the ontology.

It is important to observe that, from the semantic viewpoint, the perfect mapping associated to an OBDA specification does not add any knowledge to the specification. This is formalized by the following theorem.

THEOREM 2. *Let $\mathcal{B}$ be an OBDA specification with arbitrary mappings, and let $\mathcal{M}^p$ be a set of perfect mappings for $\mathcal{B}$. Then, for every legal database instance D for $\mathcal{C}$, $Models(\langle \mathcal{T}, \mathcal{M} \cup \mathcal{M}^p, \mathcal{S}, \mathcal{C}\rangle, D) = Models(\mathcal{B}, D)$.*

PROOF. First, we show that every interpretation $\mathcal{I} \in Models(\mathcal{B}, D)$ satisfies every mapping assertion in $\mathcal{M}^p$. In fact, let $Q_{DB}(x_1, \ldots, x_n) \rightsquigarrow cq(x_{i_1}, \ldots, x_{i_k})$ be a mapping assertion in $\mathcal{M}^p$. Then, from Definition 6 it follows that, for every n-tuple of constants $\langle c_1, \ldots, c_n\rangle \in Ans(Q_{DB}, D)$, the tuple $\vec{c}\,' = \langle c_{i_1}, \ldots, c_{i_k}\rangle$ is a certain answer to cq over $\mathcal{B}$ and D, and therefore $\mathcal{I} \models cq(\vec{c}\,')$ for every $\mathcal{I} \in Models(\mathcal{B}, D)$, which implies that the above mapping assertion is satisfied in $\mathcal{I}$. This proves that $Models(\langle \mathcal{T}, \mathcal{M} \cup \mathcal{M}^p, \mathcal{S}, \mathcal{C}\rangle, D) \supseteq Models(\mathcal{B}, D)$. Conversely, $Models(\langle \mathcal{T}, \mathcal{M} \cup \mathcal{M}^p, \mathcal{S}, \mathcal{C}\rangle, D) \subseteq Models(\mathcal{B}, D)$ trivially follows from Definition 2. □

The above property shows that the OBDA specification $\langle \mathcal{T}, \mathcal{M} \cup \mathcal{M}^p, \mathcal{S}, \mathcal{C}\rangle$ is logically equivalent to the OBDA specification $\langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$. Nevertheless, in the following we prove that the availability of perfect mapping assertions may drastically improve the computation of perfect rewritings (and hence the overall query answering over OBDA systems).

Similarly to the case of view inclusions, one might wonder how, in practice, perfect mappings for an OBDA specification are derived. This issue will be discussed at the end of the next section.

6. THE ALGORITHM PERFECTMAP

In this section we present PerfectMap, a new technique for computing OBDA perfect rewritings. With respect to the algorithm $\textit{OBDA-Rewrite}_{ID}$ presented above, PerfectMap exploits the presence of perfect mappings to optimize the size of the OBDA perfect rewriting of queries.

We first reformulate the perfect mapping $\mathcal{M}^p$ into a high-level and a low-level mapping in a way analogous to the above algorithm $\textit{OBDA-Rewrite}_{ID}$. More precisely, a perfect mapping assertion m of the form $Q_{DB}(\vec{x}^1) \rightsquigarrow cq(\vec{x}^2)$ is associated with a fresh auxiliary view predicate v_m, and is reformulated into a high-level mapping assertion $\{\vec{x}^2 \mid \exists \vec{x}\,'.v_m(\vec{x}^2, \vec{x}\,')\} \rightsquigarrow cq_i(\vec{x}^2)$ (where $\vec{x}\,'$ represents the tuple of variables from $\vec{x}^1$ that do not occur in $\vec{x}^2$), and a low-level mapping assertion $Q_{DB}(\vec{x}^1) \rightsquigarrow \{\vec{x}^1 \mid v_m(\vec{x}^1)\}$. We denote with $\mathcal{M}^p_H$ (resp. $\mathcal{M}^p_L$) the set of high-level (resp. low-level) mapping assertions of the form above.

The PerfectMap algorithm is the following.

Algorithm PerfectMap $(\mathcal{B}, \mathcal{M}^p, Q)$
Input: OBDA specification $\mathcal{B} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S}, \mathcal{C}\rangle$
with $\mathcal{T}$ expressed in a UCQ-rewritable language $\mathcal{L}$
perfect mapping $\mathcal{M}^p$ for $\mathcal{B}$
UCQ Q
Output: An SQL query over $\mathcal{S}$
begin
$Q_1 = \textit{ReplaceSubqueryR}(Q, \mathcal{M}^p_H, \mathcal{T})$;
Compute a UCQ Q_2 that is an ontology rewriting of Q_1 under $\mathcal{B}$;
$Q_3 = \textit{ReplaceSubquery}(Q_2, \mathcal{M}^p_H)$;
$Q_4 = \textit{HighMappingRewrite}(Q_3, \mathcal{M}_H)$;
$Q_5 = \textit{ID-Optimize}(Q_4, \mathcal{C})$;
$Q_6 = \textit{LowMappingRewrite}(Q_5, \mathcal{M}_L \cup \mathcal{M}^p_L)$;
return Q_6;
end

PerfectMap is constituted of six steps: step 2, 4, 5, and 6 are actually identical to the four steps of the previous algorithm $\textit{OBDA-Rewrite}_{ID}$. Thus, the novelty of PerfectMap lies in the first and the third steps, i.e., in the algorithms *ReplaceSubqueryR* and

ReplaceSubquery. The purpose of these algorithms is to *stop* the rewriting of subqueries that correspond to queries appearing in the head of perfect mappings. All the atoms of one such subquery are replaced with a single atom whose predicate is the auxiliary view predicate in the body of the high-level perfect mapping assertion. This atom will not be further rewritten until step 5 of PerfectMap.

Notice that, even though PerfectMap essentially considers the algorithm adopted at step 2 as a black box, we need here to assume that such an algorithm is able to process UCQs with auxiliary view predicates: this is trivial for ontology languages allowing for n-ary predicates [7, 2], whereas it may require some adaptations for DL languages.

The algorithms *ReplaceSubqueryR* and *ReplaceSubquery* are very similar, however they are based on different conditions for the applicability of the above replacement to subqueries of the input query.

The algorithm *ReplaceSubqueryR* is based on the notion of restricted homomorphism. Given two CQs q_1, q_2, a *restricted homomorphism* from q_1 to q_2 is a function h on the variables occurring in q_1 such that: (i) every distinguished variable of q_1 is mapped to a distinguished variable of q_2 or to a constant; (ii) every existential variable of q_1 is mapped to an existential variable of q_2; (iii) $h(q_1) = q_2$, where $h(q_1)$ is the CQ obtained from q_1 by applying the function h to its variables.

In the following, we assume that a high-level perfect mapping assertion m_i is of the form $\{\vec{x} \mid \exists \vec{y}.v_{m_i}(\vec{x},\vec{y})\} \rightsquigarrow cq_i(\vec{x})$. Furthermore, we call *length* of m_i, denoted $length(m_i)$, the number of atoms in cq_i.

The algorithm *ReplaceSubqueryR* is as follows:

Algorithm *ReplaceSubqueryR* $(Q, \mathcal{M}_H^p)$
Input: UCQ Q, high-level perfect mapping $\mathcal{M}_H^p$
Output: UCQ Q'
begin
  Compute an ordering $\langle m_1, \ldots, m_n\rangle$ of the assertions in $\mathcal{M}_H^p$
  s.t. $length(m_i) \geq length(m_{i+1})$ for every $i \in \{1, \ldots, n-1\}$;
  $Q_0 = Q$;
  $i = 1$;
  while $i \leq n$ **do begin**
    $Q' = Q_0$;
    repeat
      if there exists a CQ $q \in Q'$ of the form
        $\{\vec{x} \mid \exists \vec{y}.\alpha_1(\vec{x},\vec{y}) \wedge \ldots \wedge \alpha_n(\vec{x},\vec{y})\}$
        and a subquery q' of q of the form
        $\{\vec{x}' \mid \exists \vec{y}'.\alpha'_1(\vec{x}',\vec{y}') \wedge \ldots \wedge \alpha'_m(\vec{x}',\vec{y}')\}$ such that:
        (i) every existential variable of q that occurs in q' does not
          occur in any atom of q that does not belong to q', and
        (ii) there exists a restricted homomorphism h from cq_i to q'
      then
        replace q in Q' with the CQ
        $\{\vec{x} \mid \exists \vec{y}.h(v_{m_i}(\vec{x})) \wedge \alpha_{j_1}(\vec{x},\vec{y}) \wedge \ldots \wedge \alpha_{j_k}(\vec{x},\vec{y})\}$
        (where $\alpha_{j_1}(\vec{x},\vec{y}), \ldots, \alpha_{j_k}(\vec{x},\vec{y})$ are the atoms of
        q different from $\alpha'_1(\vec{x}',\vec{y}'), \ldots, \alpha'_m(\vec{x}',\vec{y}')$);
    until $Q' = Q_0$;
    $i = i + 1$;
  end
  return Q';
end

Essentially, the algorithm looks for subqueries in each CQ in Q that unify with the head of high-level perfect mapping assertions, and replaces such subqueries with single atoms using the view names of these mapping assertions. Notice that this is a form of query rewriting using views, a well-known NP-complete problem [17, 10]. In fact, *ReplaceSubqueryR* faces this problem by applying the following greedy strategy: the high-level perfect mapping assertions are ordered by decreasing length, and are checked in this order against the queries of Q. So, at every step, a perfect mapping assertion of maximum length is selected among the ones that are applicable to some query in Q. Of course, such a greedy strategy does not produce in general an optimal solution (i.e., CQs of minimal length).

Moreover, the algorithm executes the above subquery replacement under a restricted notion of unification between the head of the high-level perfect mapping assertion and the subquery used in the algorithm. As described above, the algorithm only looks for subqueries that do not share existential variables with the rest of the query, and only applies restricted homomorphisms to unify the head of the mapping with the subquery. It can be shown that such restrictions constitute a sufficient condition for the correctness of such a replacement.

THEOREM 3. *Let $\mathcal{B}$ be an OBDA specification, let q_1, q_2 be Boolean CQs, let q' be a subquery of q_2 such that there exists a restricted homomorphism h from q_1 to q' and q' does not share existential variables with $q_2 - q'$. If Q_1, Q'' are UCQs such that Q_1 is an ontology rewriting of q_1 under $\mathcal{B}$ and Q'' is an ontology rewriting of $q_2 - q'$ under $\mathcal{B}$, then $h(Q_1) \wedge Q''$ is an ontology rewriting of q_2 under $\mathcal{B}$.*

Then, the algorithm *ReplaceSubquery* is a simplified version of *ReplaceSubqueryR*: the difference lies in the fact that *ReplaceSubquery* applies the standard, non-restricted notions of unification and homomorphism in the subquery replacement step, i.e., the algorithm does not impose any restriction neither on the existential variables occurring in the subexpression nor in the homomorphism. Therefore, the specification of *ReplaceSubquery* can be obtained from that of *ReplaceSubqueryR* by simply replacing point (i) and (ii) with the condition:

"there exists a homomorphism h from cq_i to q'"

From Theorem 1 and Theorem 3, correctness of PerfectMap easily follows.

THEOREM 4. *Let $\mathcal{B}$ be an OBDA specification, let $\mathcal{M}^p$ be a perfect mapping for $\mathcal{B}$, let Q be a UCQ, and let Q' be the query returned by* PerfectMap$(Q, \mathcal{B}, \mathcal{M}^p)$. *Then, Q' is an OBDA perfect rewriting of q under $\mathcal{B}$.*

Let us now intuitively explain the optimization introduced by PerfectMap. The above described subquery replacement allows for exploiting perfect mappings to minimize OBDA perfect rewritings, because in this way a subquery is directly mapped to the OBDA perfect rewriting represented in the perfect mapping. This allows for lowering the number of CQs over the view names produced at the end of step 2, but only if perfect mappings store OBDA perfect rewritings of queries that have been significantly optimized, through the *ID-Optimize* algorithm, with respect to their unoptimized version. Therefore, the presence and usage of view inclusions is crucial for making perfect mappings useful in PerfectMap. In other words, the key idea here is that the use of perfect mappings allows for drastically reducing the number of conjunctive queries generated at the end of step 2, if the perfect mappings are the result of a previous optimization due to the presence of view inclusions.

EXAMPLE 5. *The perfect mapping m shown in Example 4 can be exploited by* PerfectMap. *Consider for example the following query Q_2:*

$$\{z_1, z_2, z_3, z_4 \mid \mathit{PublicOrg}(z_1) \wedge \mathit{worksWith}(z_1, z_2) \wedge \mathit{PublicOrg}(z_2) \wedge \mathit{name}(z_1, z_3) \wedge \mathit{name}(z_2, z_4)\}$$

Let us consider the query $Q' = \{z_1, z_2 \mid \mathit{PublicOrg}(z_1) \wedge \mathit{worksWith}(z_1, z_2) \wedge \mathit{PublicOrg}(z_2)\}$, which is a subquery of Q_2.

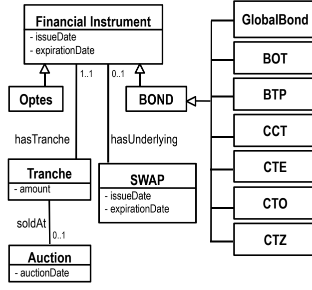

Figure 1: UML representation of the test ontology

Now, $h(x) = z_1$, $h(y) = z_2$ is a restricted homomorphism to Q' from the head of the perfect mapping m given in Example 4.

It is easy to see that Q' and m satisfy the conditions described in the algorithm ReplaceSubqueryR. Therefore, ReplaceSubqueryR replaces in Q_2 the atoms occurring in the query Q' with the atom $v_m(z_1, z_2)$, and returns the CQ

$$\{z_1, z_2, z_3, z_4 \mid \mathsf{v}_m(z_1, z_2) \wedge \mathit{name}(z_1, z_3) \wedge \mathit{name}(z_2, z_4)\}$$

where v_m is the auxiliary view predicate associated to m. This query is then passed to the algorithm HighMappingRewrite which in turn returns the CQ

$$\{z_1, z_2, z_3, z_4 \mid v_m(z_1, z_2) \wedge v1(z_1, z_3) \wedge v1(z_2, z_4)\}.$$

Notice that, without the replacement done at step 1 using the perfect mapping assertion m, at step 2 the algorithm PerfectMap *would rewrite the atoms of the subquery Q' according to the ontology. This actually would mean producing several queries, e.g., 4 CQs if such rewriting is performed as mentioned in Example 3. Then, HighMappingRewrite would return 8 CQs instead of one (cf. Example 3).*

It is easy then to obtain the perfect rewriting by applying step 5 and step 6 of the algorithm PerfectMap*.*

□

Finally, as anticipated in the previous section, we deal with the issue of how automatically derive perfect mapping assertions. The idea is to use PerfectMap for this purpose. Actually, perfect mapping assertions may be obtained by simply storing the perfect rewritings computed by the algorithm PerfectMap itself. In fact, one may think of a methodology that starts from relatively simple queries, that can be computed even in the absence of perfect mappings, and whose OBDA perfect rewriting is significantly optimized by the *ID-Optimize* algorithm. Once such an OBDA perfect rewriting Q_{DB} is produced by PerfectMap for a query q, then it is possible to add the mapping $Q_{DB} \rightsquigarrow q$ to the perfect mapping assertions. In other words, the set of perfect mappings can be simply considered as a *memory* of the previous optimized OBDA perfect rewritings computed by the system itself. In this sense, a nice property of PerfectMap is that its computation improves along the time, since PerfectMap exploits its previous results to compute new results.

7. IMPLEMENTATION AND EVALUATION

We have implemented the algorithm PerfectMap and integrated it into the MASTRO tool for OBDA [4]. The new release of MASTRO thus obtained has been used within a joint project between Sapienza Università di Roma and the Italian Ministry of Economy and Finance. The main objectives of the project have been: the design and specification in *DL-Lite$_A$* of an ontology on the domain of the Italian public debt; the realization of the mapping between the ontology and relational data sources that are part of the management accounting system currently in use at the ministry; the definition and execution of queries over the ontology aimed at extracting data of core interest for the users of the applications. In particular, the information returned by such queries relates to sales of bonds issued by the Italian government, maturities of bonds, monitoring of various financial products, etc., and are at the basis of various reports on the overall trend of the national public debt.

The overall ontology that we realized is over an alphabet containing 164 concept names, 47 role names, 86 attribute names, and is specified through around 1440 *DL-Lite$_A$* assertions. Moreover, we defined 300 mapping assertions involving around 60 relational tables managed by Microsoft SQLServer. This database contains 193 tables for an overall number of 3 million tuples and an overall size of 1.5 gigabytes. To capture the interdependencies between mapping assertions, due to the characteristics of the domain, we specified around 100 inclusion assertions on mapping views, and executed a tuning phase during which the system learned about 50 perfect mapping assertions. We tested a very high number of queries and produced through MASTRO several reports of interest for the ministry. We point out that around 80% of the queries we tested could be executed only thanks to the optimizations described in this paper, since without optimizations the system either ran out of memory or did not terminate within a 4-hour timeout. Furthermore, optimizations allowed for a drastic reduction of the query answering time for all queries of interest.

We now present some sample tests that summarize what we experimented in the practice. We carried out such tests over an excerpt of the ontology produced in the project, and considering only a subset of the mappings and of the inclusions. An UML repre-

$Q_{1,1} = \{x, y \mid \mathsf{SW}(x) \wedge \mathsf{issD}(x, y)\}$
$Q_{1,2} = \{x, y, z \mid \mathsf{SW}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{hasUnd}(x, z)\}$
$Q_{1,3} = \{x, y, z, k \mid \mathsf{SW}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{hasUnd}(x, z) \wedge \mathsf{issD}(z, k)\}$
$Q_{1,4} = \{x, y, z, k, w \mid \mathsf{SW}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{hasUnd}(x, z) \wedge \mathsf{issD}(z, k) \wedge \mathsf{expD}(z, w)\}$
$Q_{1,5} = \{x, y, z, k, w, t \mid \mathsf{SW}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{hasUnd}(x, z) \wedge \mathsf{issD}(z, k) \wedge \mathsf{expD}(z, w) \wedge \mathsf{hasTr}(z, t)\}$
$Q_{1,6} = \{x, y, z, k, w, t, a \mid \mathsf{SW}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{hasUnd}(x, z) \wedge \mathsf{issD}(z, k) \wedge \mathsf{expD}(z, w) \wedge \mathsf{hasTr}(z, t) \wedge \mathsf{amount}(t, a)\}$

$Q_{2,1} = \{x \mid \mathsf{Bond}(x)\}$
$Q_{2,2} = \{x, y \mid \mathsf{Bond}(x) \wedge \mathsf{issD}(x, y)\}$
$Q_{2,3} = \{x, y, z \mid \mathsf{Bond}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{expD}(x, z)\}$
$Q_{2,4} = \{x, y, z, t \mid \mathsf{Bond}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{expD}(x, z) \wedge \mathsf{hasTr}(x, t)\}$
$Q_{2,5} = \{x, y, z, t, a \mid \mathsf{Bond}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{expD}(x, z) \wedge \mathsf{hasTr}(x, t) \wedge \mathsf{amount}(t, a)\}$
$Q_{2,6} = \{x, y, z, t, a, k \mid \mathsf{Bond}(x) \wedge \mathsf{issD}(x, y) \wedge \mathsf{expD}(x, z) \wedge \mathsf{hasTr}(x, t) \wedge \mathsf{amount}(t, a) \wedge \mathsf{soldAt}(t, k) \wedge \mathsf{aucD}(k, y)\}$

$Q_{3,1} = \{x \mid \mathsf{Tr}(x)\}$
$Q_{3,2} = \{x, y \mid \mathsf{Tr}(x) \wedge \mathsf{hasTr}(y, x)\}$
$Q_{3,3} = \{x, y, a \mid \mathsf{Tr}(x) \wedge \mathsf{hasTr}(y, x) \wedge \mathsf{amount}(x, a)\}$
$Q_{3,4} = \{x, y, z \mid \mathsf{Tr}(x) \wedge \mathsf{hasTr}(y, x) \wedge \mathsf{Tr}(z) \wedge \mathsf{hasTr}(y, z)\}$
$Q_{3,5} = \{x, y, z, a \mid \mathsf{Tr}(x) \wedge \mathsf{hasTr}(y, x) \wedge \mathsf{amount}(x, a) \wedge \mathsf{Tr}(z) \wedge \mathsf{hasTr}(y, z) \wedge \mathsf{amount}(z, a)\}$
$Q_{3,6} = \{x, y, z, a, k \mid \mathsf{Tr}(x) \wedge \mathsf{hasTr}(y, x) \wedge \mathsf{amount}(x, a) \wedge \mathsf{Tr}(z) \wedge \mathsf{hasTr}(y, z) \wedge \mathsf{amount}(z, a) \wedge \mathsf{soldAt}(x, k) \wedge \mathsf{soldAt}(z, k)\}$

$Q_{4,1} = \{x, y \mid \mathsf{hasTr}(x, y)\}$
$Q_{4,2} = \{x, y, a \mid \mathsf{hasTr}(x, y) \wedge \mathsf{amount}(y, a)\}$
$Q_{4,3} = \{x, y, a, k \mid \mathsf{hasTr}(x, y) \wedge \mathsf{amount}(y, a) \wedge \mathsf{soldAt}(y, k)\}$
$Q_{4,4} = \{x, y, a, d, k \mid \mathsf{hasTr}(x, y) \wedge \mathsf{amount}(y, a) \wedge \mathsf{soldAt}(y, k) \wedge \mathsf{aucD}(k, d)\}$
$Q_{4,5} = \{x, y, a, d, k \mid \mathsf{BOT}(x) \wedge \mathsf{hasTr}(x, y) \wedge \mathsf{amount}(y, a) \wedge \mathsf{soldAt}(y, k) \wedge \mathsf{aucD}(k, d)\}$
$Q_{4,6} = \{x, y, a, d, k, t, r \mid \mathsf{BOT}(x) \wedge \mathsf{hasTr}(x, y) \wedge \mathsf{amount}(y, a) \wedge \mathsf{soldAt}(y, k) \wedge \mathsf{aucD}(k, d) \wedge \mathsf{issD}(x, t) \wedge \mathsf{expD}(x, r)\}$

Figure 2: Queries used in the experiments

	(i) No Opt.			(ii) Inclusion Opt.			(iii) Full Opt.		
	ND	RT	ET	ND	RT	ET	ND	RT	ET
$Q_{1,1}$	7	74	162	2	84	78	2	73	68
$Q_{1,2}$	7	71	122	2	80	62	2	68	70
$Q_{1,3}$	161	176	3644	2	250	94	2	111	82
$Q_{1,4}$	3703	9504	145885	2	70708	242	2	94	90
$Q_{1,5}$	44436	1267285	2397053		T/O		4	88	148
$Q_{1,6}$	*(12600000)*	T/O			T/O		4	126	226
$Q_{2,1}$	37	71	257	1	89	81	1	80	76
$Q_{2,2}$	851	210	14301	1	2698	91	1	80	75
$Q_{2,3}$	19573	21846	743797	1	1599547	154	1	89	92
$Q_{2,4}$	*(266400)*	T/O			T/O		2	78	217
$Q_{2,5}$	*(2664000)*	T/O			T/O		2	87	251
$Q_{2,6}$	*(213120000)*	T/O			T/O		2	302	538
$Q_{3,1}$	22	62	328	4	70	162	4	70	112
$Q_{3,2}$	12	64	236	2	66	115	2	72	92
$Q_{3,3}$	120	110	3403	2	194	171	2	69	105
$Q_{3,4}$	144	107	3587	4	242	382	4	70	321
$Q_{3,5}$	14400	94327	686085	4	1133332	501	4	985	950
$Q_{3,6}$	*(1440000)*	T/O			T/O		16	2766	3774
$Q_{4,1}$	12	57	234	2	66	110	2	74	114
$Q_{4,2}$	120	102	3315	2	168	161	2	77	170
$Q_{4,3}$	1200	1202	37044	4	6769	300	4	87	343
$Q_{4,4}$	9600	46646	537886	4	481921	757	4	119	503
$Q_{4,5}$	28800	410201	1537348		T/O		4	78	509
$Q_{4,6}$	*(17280000)*	T/O			T/O		4	22957	1157

Legenda: ND = Number of disjuncts of the rewriting; RT = Rewriting time; ET = Evaluation time; T/O = timeout

Figure 3: Experimental results

sentation of the portion of the ontology used in this test is given in Figure 1. Such a portion talks about Financial Instrument and the two main subclasses it has, i.e., Bond and Optes, where Bond indicates a certificate of debt issued by the Italian government or a by local public organization, and Optes consists of very short debt contracts (normally of few days), used mainly to raise cash. Each Financial Instrument has an issueDate and expirationDate. Bond has several subclasses of interest. Each of them corresponds to a particular type of certificates of debt. For example BOT and CTZ are zero coupon bond of different duration where interests are computed at a fixed rate, whereas BTP are deferred coupon bonds. A Financial Instrument can be issued in various tranches, each with a certain amount, and soldAt an Auction, in a certain auctionDate. A SWAP is a contract in which two parties agree to exchange periodic interest payments. The amounts exchanged depend on a financial instrument called underlying (cf. role hasUnderlying in Figure 1). Each SWAP has an issueDate and expirationDate.

Figure 2 shows 24 queries divided into four groups, each one containing queries of increasing difficulty (due to lack of space, the reported queries use abbreviations of predicate names). We executed such queries through MASTRO under three different modalities: (i) without optimizations, which amounts to simply execute the *OBDA-Rewrite*$_{ID}$ algorithm in the absence of view inclusions; (ii) with the optimization that uses only inclusions on mapping views, which means executing all steps of *OBDA-Rewrite*$_{ID}$ for every query; (iii) with the complete optimization technique described in this paper. More precisely, in modality (iii), we ran the algorithm PerfectMap for each query, and modified the extended OBDA system specification in input at each such execution, starting with an empty set of perfect mappings, and adding at each execution perfect mapping assertions learned at the end of the previous execution. We considered each group of query separately, i.e., the set $\mathcal{M}^p$ has been emptied after processing each group of queries.

The experiments have been carried out on a 2.6 GHz Intel Core i7 740M with 6GB RAM memory and under Windows7 operating system and Microsoft SQL Server DBMS. Figure 3 reports the results we have obtained under the three mentioned modalities (columns (i), (ii) and (iii)). All times are in milliseconds, and the timeout has been fixed at 2 hours. Timeout is always caused by the rewriting phase: for queries that led to system timeout we report an estimated number of the expected disjuncts of the rewriting. This is indicated in italic and between parenthesis in the ND column of mode (i). For each query q with n atoms, let k_i with $1 \leq i \leq n$ be the number of mappings that involve the i-th atom predicate of q, the above value is given by the product $k_1 * \ldots * k_n$. Analogous values for mode (ii) are omitted for space reasons.

As already discussed, the table shows that applying the optimization that exploit mapping inclusions produces rewritings of much smaller size, and requiring lower evaluation time, with respect to those produced without using such optimization (see columns ND and ET under (i) and (ii), respectively). This however can make the overall rewriting process longer, since it also takes into account the time needed for the execution of the *ID-Optimize* algorithm. For query $Q_{1,5}$ and query $Q_{4,5}$ this even led to a system timeout, whereas, by disabling the optimizations, we have been able to process such queries (see columns RT under (i) and (ii)).

To execute all queries considered in our experiments we therefore needed to use all optimizations presented in this paper (modality (iii) of the experiments). The power of such optimizations is clearly showed by the last two columns of Figure 3. Indeed, even though for very simple queries, e.g., $Q_{1,1}, Q_{1,2}, Q_{2,1}$, the computational overhead introduced by the use of perfect mappings is not useful, for complex queries the gain is really notable.

8. RELATED WORK

To the best of our knowledge, our work is the first one to investigate in detail the mapping rewriting step in OBDA and to propose optimizations for this task. Indeed, as already said, previous works on OBDA (see, e.g., [5, 24, 13, 2, 9]) have focused on the ontology rewriting step and its optimizations, and for this reason have considered a simplified OBDA framework, in which data are directly stored in the ontology ABox, rather than in external data sources connected to the ontology TBox through mappings. Notably, the optimization technique proposed in this paper couples with any ontology rewriting algorithm proposed in the literature, provided that the ontology rewriting is given in terms of a UCQ.

The mappings considered in the present paper have been in fact introduced in [19]. In that paper, however, no optimizations for the mapping rewriting step have been provided. In this respect, the introduction of view inclusion dependencies in the OBDA specification, and their use in the algorithm *OBDA-Rewrite*$_{ID}$, as well as the notion of perfect mappings, and the algorithm PerfectMap are original contributions of this paper.

A recent work [22, 23] that is very close to the present approach proposes the use of *ABox dependencies* to optimize conjunctive query answering in the DL *DL-Lite*. ABox dependencies are concept inclusions that are satisfied by the ABox, i.e., the extensional part of the ontology. While there is a semantic similarity between database view inclusions and ABox dependencies, it can be shown that view inclusions are more powerful than the ABox dependencies used in [22, 23], since they are expressed at a lower level of the OBDA system: therefore, in general view inclusions cannot be translated into corresponding ABox dependencies.

As already said, mapping rewriting is also relevant in data integration, where the notions of GAV and LAV mappings have been introduced [15, 25, 11]. Research in this field has mostly considered GAV mapping rewriting a trivial task, realizable essentially by unfolding query atoms with the corresponding mapping views. Conversely, LAV mapping rewriting, which amounts to a form of query answering using views, has been widely investigated, and

various interesting optimizations for it have been proposed (see, e.g., [17, 8, 20, 12]). We notice that all such techniques are specifically tailored to the treatment of non-distinguished variables occurring in the mapping views, and that in the absence of such variables, as in GAV mappings, they essentially collapse to unfolding. Therefore, their use does not provide gains in our OBDA framework.

An interesting exception among papers considering GAV mapping rewritings is [14], which addresses the problem of answering SPARQL queries posed over virtual (GAV) SPARQL views. Driven by motivations similar to ours, the authors provide some optimizations to reduce the size of the final rewriting, which is a union of conjunctive queries specified in SPARQL. In particular, their methods aim to minimize conjunctive queries in the rewriting and eliminate disjuncts that have an empty evaluation. To eliminate such disjuncts they exploit some heuristics that look at the underlying data. In this respect, the approach differs from ours, which is instead purely intensional, and therefore does not have to cope with updates at data sources. Furthermore, the technique in [14] does not make use of dependencies on mapping views, nor allow for re-using previously computed rewritings, as we do through perfect mappings, and can be therefore considered complementary to our approach, despite the different languages it considers.

9. CONCLUSIONS

In this paper we have presented an approach to the optimization of query answering in OBDA. The approach is based on two main ideas: (i) the use of inclusions on database views to optimize perfect rewritings; (ii) the usage of perfect mappings to reuse previous optimizations of subqueries in the computation of the perfect rewriting of a new query.

We plan to continue the present work along several directions. First, we would like to extend our constraint language on database views to other forms of constraints, in particular *disjointness* constraints. Then, it would be interesting to go beyond perfect mappings, and explore alternative ways of exploiting view inclusions (as well as other forms of constraints over the views) to improve ontology rewriting and the overall query rewriting process in OBDA. Finally, we would like to consider whether, and to what extent, our optimizations can be extended to OBDA contexts using different ontology specification languages, even non-UCQ-rewritable ones.

Acknowledgments. This research has been partially supported by the EU under FP7 project Optique – Scalable End-user Access to Big Data (grant n. FP7-318338).

10. REFERENCES

[1] P. A. Bernstein and L. Haas. Information integration in the enterprise. *Comm. of the ACM*, 51(9):72–79, 2008.

[2] A. Calì, G. Gottlob, and A. Pieris. New expressive languages for ontological query answering. In *Proc. of AAAI 2011*, pages 1541–1546, 2011.

[3] A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of IJCAI 2003*, pages 16–21, 2003.

[4] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The Mastro system for ontology-based data access. *Semantic Web J.*, 2(1):43–53, 2011.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Conceptual modeling for data integration. In A. T. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, editors, *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*, volume 5600 of *LNCS*, pages 173–197. Springer, 2009.

[7] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. *Artificial Intelligence*, 2012. To appear.

[8] O. M. Duschka, M. R. Genesereth, and A. Y. Levy. Recursive query plans for data integration. *J. of Logic Programming*, 43(1):49–73, 2000.

[9] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*, pages 2–13, 2011.

[10] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[11] A. Y. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *Proc. of VLDB 2006*, pages 9–16, 2006.

[12] G. Konstantinidis and J. L. Ambite. Scalable query rewriting: a graph-based approach. In *Proc. of ACM SIGMOD*, pages 97–108, 2011.

[13] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in *DL-Lite*. In *Proc. of KR 2010*, pages 247–257, 2010.

[14] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In *Proc. of WWW 2011*, pages 655–664, 2011.

[15] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.

[16] M. Lenzerini. Ontology-based data management. In *Proc. of CIKM 2011*, pages 5–6, 2011.

[17] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. of PODS'95*, pages 95–104, 1995.

[18] H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic*, 8(2):186–209, 2010.

[19] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[20] R. Pottinger and A. Y. Halevy. MiniCon: A scalable algorithm for answering queries using views. *VLDB Journal*, 10(2–3):182–198, 2001.

[21] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2–3):91–110, 2002.

[22] M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work in practice. In *Proc. of AMW 2011*, volume 749 of *CEUR*, `ceur-ws.org`, 2011.

[23] M. Rodriguez-Muro and D. Calvanese. High performance query answering over *DL-Lite* ontologies. In *Proc. of KR 2012*, pages 308–318, 2012.

[24] R. Rosati and A. Almatelli. Improving query answering over *DL-Lite* ontologies. In *Proc. of KR 2010*, pages 290–300, 2010.

[25] J. D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.