MASTRO at Work: Experiences on Ontology-based Data Access

Domenico Fabio Savo[1], Domenico Lembo[1], Maurizio Lenzerini[1], Antonella Poggi[1], Mariano Rodriguez-Muro[2], Vittorio Romagnoli[3], Marco Ruzzi[1], Gabriele Stella[3]

[1] SAPIENZA Università di Roma
lastname@dis.uniroma1.it

[2] Free University of Bozen-Bolzano
rodriguez@inf.unibz.it

[3] Banca Monte dei Paschi di Siena
firstname.lastname@banca.mps.it

Abstract. We report on an experimentation of Ontology-based Data Access (OBDA) carried out in a joint project with SAPIENZA University of Rome, Free University of Bolzano, and Monte dei Paschi di Siena (MPS), where we used MASTRO for accessing, by means of an ontology, a set of data sources of the actual MPS data repository. By both looking at these sources, and by interviews with domain experts, we designed both the ontology representing the conceptual model of the domain, and the mappings between the ontology and the sources. The project confirmed the importance of several distinguished features of *DL-Lite*$_{A,Id}$ to express the ontology and has shown very good performance of the MASTRO system in all the reasoning tasks, including query answering, which is the most important service required in the application.

1 Introduction

While the amount of data stored in current information systems continuously grows, turning these data into information is still one of the most challenging tasks for Information Technology. The task is complicated by the proliferation of data sources both in single organizations, and in open environments. Specifically, the information systems of medium and large organizations are typically constituted by several, independent, and distributed data sources, and this poses great difficulties with respect to the goal of accessing data in a unified and coherent way. Such a unified access is crucial for getting useful information out of the system, as well as for taking decision based on them. 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 [7].

The following are some of the reasons why a unified access to data sources is problematic.

- Despite the fact that the initial design of a collection of data sources (e.g., a database) is adequate, corrective maintenance actions tend to re-shape the data sources into a form that often diverges from the original conceptual structure.
- It is common practice to change a data repository so as to adapt it both to specific application-dependent needs, and to new requirements. The result is that data sources often become data structures coupled to a specific application (or, a class of applications), rather than application-independent databases.

– The data stored in different sources tend to be redundant, and mutually inconsistent, mainly because of the lack of central, coherent and unified data management tasks.

In principle, there are two alternative solutions to the above problems. One solution is the re-engineering of the information system, i.e., the design of a new, coherent, and unified data repository serving all the applications of the organization [8], and replacing the original data sources. This approach is unfeasible in many situations, due to cost and organization problems. The other solution is to create a new stratum of the information system, co-existing with the data sources, according to the "data integration" paradigm [1]. Such new stratum is constituted by (i) a global (also called "mediated") schema, representing the unified structure presented to the clients, and (ii) the mapping relating the source data with the elements in global schema. There are two methods for realizing such stratum, called materialized and virtual. In the materialized approach, called data warehousing, the global schema is populated with concrete data deriving from the sources. In the virtual approach, data are not moved, and queries posed to the system are answered by suitably accessing the sources [9]. The latter approach, which is the one referred to in this work, is preferable in a dynamic scenario, where sources may be updated frequently, and clients want to use up-to-date information.

In current data integration tools the global schema is expressed in terms of a logical database model, e.g. the relational data model [1]. It is well-known that the abstractions and the constructs provided by this kind of data models are influenced by implementation issues. It follows that the global schema represents a sort of unified data structure accommodating the various data at the sources, and the client, although freed from physical aspects of the source data (where they are, and how they can be accessed), is still exposed to issues concerning how data are packed into specific structures.

To overcome these problems, we recently proposed the notion of *ontology-based data integration*, also called *ontology-based data access* (OBDA) [14,12][1], whose basic idea is to express the global schema as an ontology, i.e., a conceptual specification of the application domain. With this idea, the integrated view that the 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 and relationships in the domain of interest, with the mapping acting as the reconciling mechanism between the conceptual level and the data sources. Besides this characteristic, OBDA also exploits reasoning on the ontology in computing the answers to queries, thus (at least partially) overcoming possible incompleteness that may be present in the data.

In this paper we report on an experimentation of OBDA carried out in a joint project by Banca Monte dei Paschi di Siena (MPS)[2], Free University of Bozen-Bolzano, and SAPIENZA Università di Roma, where we used MASTRO [13] for accessing, by means of an ontology, a set of data sources from the actual MPS data repository. MASTRO is an OBDA system extending the QUONTO[3] reasoner, which is based on, *DL-Lite*$_{A,Id}$ [2],

[1] The two terms have very similar meaning. We tend to use the term "ontology-based data integration" in scenarios where the data sources are heterogenous (i.e., managed by different data management systems), and distributed, which is not the case in the project described here.

[2] MPS is one of the main banks, and the head company of the third banking group in Italy (see `http://english.mps.it/`).

[3] `http://www.dis.uniroma1.it/quonto`

one of the logics of the *DL-Lite* family [4]. The OBDA scenario refers to a set of 12 relational data sources, collectively containing about 15 million tuples. By both looking at these sources, and by interviews with domain experts, we designed both the ontology representing the conceptual model of the domain, and the mapping between the ontology and the sources. The ontology comprises 79 concepts and 33 roles, and is expressed in terms of approximately 600 *DL-Lite*$_{A,Id}$ axioms. The relationships between the ontology and the sources are expressed in terms of about 200 mapping assertions. The results of the experimentation can be summarized as follows.

1) In the context of the MPS scenario, OBDA has indeed addressed many of the data access issues mentioned before. The system provides the users with the possibility of querying the data sources by means of the conceptual model of the domain, and this opens up the possibility for a variety of users of extracting information from a set of data sources that previously were accessed through specific applications.

2) The project confirmed the importance of several distinguished features of *DL-Lite*$_{A,Id}$, namely, identification constraints, and epistemic queries. Both features are missing in the standard ontology language OWL 2. In particular, we believe that the absence of identification constraints in OWL 2 may hamper the usefulness of such language in ontology-based data access.

3) MASTRO has shown very good performance in all the reasoning tasks, including query answering, which is the most important service required in the application. This has been achieved by specific optimizations designed within this project of the MASTRO query answering algorithm, in particular concerning the phase of unfolding the query against the mapping.

4) The experience in this project has shown that OBDA can be used for checking the quality of data sources. There are basically two kinds of data quality problems that our system is able to detect, one related to unexpected incompletenesses in the data sources, and the other one related to inconsistencies present in the data. The OBDA system designed for the MPS scenario has been able to provide useful information in order to improve both aspects of data quality.

5) Our work has pointed out the importance of the ontology itself, as a precious documentation tool for the organization. Indeed, the ontology developed in our project is adopted in MPS as a specification of the relevant concepts in the organization.

6) The OBDA system serves also as an inspiration for devising new data governance tasks. Relying on OBDA services, queries such as "how is a certain concept (e.g., *customer*) represented in a specific data source (e.g., table $GZ0005$)?" can now be answered, simply by exploiting both the ontology and the mappings designed in the project, and the query reformulation capability of MASTRO.

The paper is organized as follows. Section 2 presents a brief description of MASTRO. Sections 3 illustrates the scenario of our experimentation. Section 4 presents the ontology and the mapping. Section 5 illustrates the use of MASTRO in the scenario. Section 6 concludes the paper.

2 The MASTRO system

MASTRO is an OBDA system jointly developed at the SAPIENZA University of Rome and Free University of Bozen-Bolzano. MASTRO allows for the definition of

DL-Lite$_{A,Id}$ [2] ontologies connected through semantic mappings to external independent relational databases storing data to be accessed. Thus, differently from other approaches to ontology definition and reasoning [10,6,11], the extensional level of the ontology, namely, the instances of concepts and roles, are not explicitly asserted and possibly managed by a DBMS, but are specified by mapping assertions describing how they can be retrieved from the data at the sources. In the following we briefly sketch the architecture of the system, distinguishing between "Ontology Definition Module", "Mapping Manager", "Data Source Manager", and "Reasoner".

The *Ontology Definition Module* provides mechanisms for the specification of the ontology as a *DL-Lite*$_{A,Id}$ TBox. *DL-Lite*$_{A,Id}$ is a Description Logic (DL) belonging to the *DL-Lite* family, which adopts the Unique Name Assumption, and provides all the constructs of OWL 2 QL[4], a tractable profile of OWL 2, plus functionality and identification assertions, with the limitation that these kind of assertions cannot involve sub-roles. These last features, while enhancing the expressive power of the logics, do not endanger the efficiency of both intensional reasoning, and query answering. In other words, the computational complexity of these tasks is the same as in OWL 2 QL, namely PTIME with respect to the size of the TBox, and LOGSPACE in the size of the data at the sources.

The *Mapping Manager* supports the definition of mapping assertions relating the data at the sources to the concepts in the ontology. The mapping assertions supported by MASTRO are a particular form of GAV mappings [9]. More specifically, a mapping assertion is an expression of the form $\psi \rightsquigarrow \varphi$ where ψ is an arbitrary SQL query over the database, and φ is a *DL-Lite*$_{A,Id}$ conjunctive query without existential variables. As described in [12], data extracted by means of query ψ are used, together with suitable Skolem functions, to build the logic terms representing the object identifiers, thus solving the impedance mismatch problem between data at the sources and instances of the ontology. The Mapping Manager interacts with the *Data Source Manager*, which is in charge of the communication with the underlying relational sources, providing transparent access to a wide range of both commercial and freeware relational DBMSs[5].

Finally, the *Reasoner* exploits both the TBox and the mapping assertions in order to (i) check the satisfiability of the whole knowledge base, and (ii) compute the answer to the queries posed by the users. Such module is based on QUONTO, a reasoner for the *DL-Lite* family that uses query rewriting as a main processing technique. The two main run-time services provided by the reasoner are query answering, and consistency check. The MASTRO process to answer conjunctive queries (CQs) is inspired by the one implemented in the QUONTO system. First, the query posed by the user over the ontology is reformulated in terms of the inclusion assertions expressed among concepts and roles; second, such rewriting is *unfolded* according to the mapping assertions in order to generate an SQL query which can be directly issued over the relational data source. It can be shown that the answers to such an SQL query are exactly the answers logically implied by the whole knowledge base [2]. As a further powerful feature, MASTRO is able to answer EQL (Epistemic Query Language) queries [3], i.e., first-order logic queries over the ontology interpreted under an epistemic semantics. Finally, MASTRO provides the

[4] `http://www.w3.org/TR/owl2-profiles/`

[5] No relational sources can be accesses by means of suitable wrapping tools

consistency check capability. By virtue of the characteristics of *DL-Lite*$_{A,Id}$, MASTRO reduces consistency checking to verifying whether queries generated for disjointness assertions, functionality assertions, identification constraints and EQL constraints return an empty result. To this aim, a boolean query is automatically generated for every such construct and then rewritten, unfolded, and evaluated over the database.

3 Case study: The domain of experimentation

The data of interest in our case study are those exploited by MPS personnel for risk estimation in the process of granting credit to bank customers. A customer may be a person, an ordinary company, or an holding company. Customers are ranked with respect to their credit worthiness, which is established considering various circumstances and credit/debit positions of customers. In addition to customer information, data of interest regard company groups to which customers belong, and business relations between bank customers (in particular, fifteen different kinds of such relations are relevant). Hereinafter, such groups of customers will be called *Clusters of Connected Customers (CCCs)*. A 15 million tuple database, stored in 12 relational tables managed by the IBM DB2 RDBMS, has been used as data source collection in the experimentation. Figure 1 shows a summary of the data sources. Such data sources are managed by a specific application. The application is in charge of guaranteeing data integrity (in fact, the underlying database does not force constraints on data). Not only this application performs various updates, but an automatic procedure is executed on a daily basis to examine the data collected in the database so as to identify connections between customers that are relevant for the credit rating calculus. Based on these connections, customers are grouped together to form CCCs. For each cluster, several data are collected that characterize the kinds of connections holding among cluster members (i.e., specifying juridical, economic, or financial aspects of connections).

Source name	Source Decription	Source size
GZ0001	Data on customers	3.463.083
GZ0002	Data on juridical connections between customers	157.280
GZ0003	Data on guarantee connection between customers	1.270.333
GZ0004	Data on economical connections between customers	104.033
GZ0005	Data on corporation connections between customers	1.021.779
GZ0006	Data on patrimonial connections between customers	809.321
GZ0007	Data on company groups	55.362
GZ0012	Customers loan information	5.966.948
GZ0015	Data on monitoring and reporting procedures	1.243
GZ0101	Data on membership of customers into CCCs	2.225.466
GZ0102	Information on CCCs	663.656
GZ0104	Data on bank credit coordinators for juridical CCCs	38.457

Fig. 1. Data sources

Data source schemas have undergone many changes in the years, trying to adapt to the changes in the application. The result is a stratification of the data source which causes an extended use of control fields, validity flags, and no longer used attributes in the source schemas. Consequently, an increasing effort for the management of the data sources is required, which has to be completely entrusted to the management applications rather than the domain experts. The aim of the experimentation has been to prove the validity of the OBDA approach in all cases in which companies need to access efficiently their information assets.

4 Case study: ontology, mapping, and methodology

The process that led us to realize the OBDA system for the MPS case study has been carried out essentially in two main phases: in the first one, we have developed the ontology, whereas in the second one we have specified the mapping between the ontology and the data sources.

To be as much independent as possible from the actual source database, in the first phase we carried out an in-depth analysis of the business domain following a top-down approach. Therefore, after identifying the central concepts and the main relations between them, we iteratively refined the ontology, being supported in each development cycle by the experts from MPS. The top-down approach turned out to be fundamental for the success of the entire project, since in this way we were able to avoid that the data model provided by the schema of the data sources could affect the definition of the ontology, thus achieving complete separation between the conceptual layer and the logical/physical layer of the system. In fact, further information on the model coming from the analysis of the sources has been exploited only towards the end of the design process, in order to refine the realized ontology.

The final ontology comprises 79 concepts, 33 roles, 37 concept attributes, and is expressed in terms of about 600 *DL-Lite*$_{A,Id}$ axioms, including 30 identification constraints (IDCs), plus 20 EQL constraints (EQLCs). Basically, the ontology is constructed around the concepts *Customer*, *CompanyGroup*, *CCC*, and various kinds of relations existing between customers (cf. Section 3).

In the following, we report on a series of modeling issues we dealt with during the ontology definition phase. First, we observe that in the domain we have analyzed, several properties of individuals depend on time. It has been therefore necessary in the ontology to take trace of the changes of such properties, maintaining the information on the validity periods associated with each such change. Even though from a very abstract point of view, such properties might be considered roles or attributes, to properly model the temporal dimension, each such role or attribute needs to be in fact *reified* in the ontology. A timestamp attribute has been associated to each concept introduced by the reification process, together with a suitable identification constraint ensuring that no two instances of each such concept refer to the same period of time.

Example 1. The membership of a customer in a cluster of connected customers is a time-dependent notion which is associated with a validity period. A crucial requirement is that a customer is not member of two clusters at the same time. In the ontology, this is modeled by the following assertions.

1. $\exists inGrouping \sqsubseteq Customer$
2. $\exists inGrouping^- \sqsubseteq Grouping$
3. $\exists relativeTo \sqsubseteq Grouping$
4. $\exists relativeTo^- \sqsubseteq CCC$
5. $Grouping \sqsubseteq \exists inGrouping^-$
6. $Grouping \sqsubseteq \exists relativeTo$
7. (funct $relativeTo$)
8. (funct $inGrouping^-$)
9. $Grouping \sqsubseteq \delta(timestamp)$
10. $(id\ Grouping\ inGrouping^-, timestamp)$

The concept *Grouping* can be seen as the reification of the notion of membership of a customer in a CCC. Assertions (1) – (8) realize reification. Assertion (9) imposes that a *timestamp* is associated to each instance of *Grouping*. Finally, assertion (10) is the IDC imposing that no two distinct instances of $Grouping$ exist that are connected to the same pair constituted by a value for the attribute $timestamp$ and an object filler for $inGrouping^-$, thus specifying that a customer is never grouped at the same time in two CCCs.

Identification constraints turned out to be an essential modeling construct, not only for correctly modeling the temporal dimension through reification, but also for expressing important integrity constraints over the ontology that could not be captured otherwise, as shown next in Example 2.

Example 2. Two types of clusters of connected customers are of interest represented by the concepts *JuridicalCCC* and *EconomicCCC*, respectively. Consider then the following identification constraint on *JuridicalCCC*.

$(id\ JuridicalCCC\ timestamp,\ relativeTo^{-} \circ ?actualGrupping \circ inGrouping^{-} \circ inMembership \circ ?Holding \circ hasMembership^{-})$

Such constraint specifies that no two distinct instances of *JuridicalCCC* exist that are connected to the same pair constituted by a value for $timestamp$ and an object filler for the *path* $relativeTo^{-} \circ ?actualGrupping \circ inGrouping^{-} \circ inMembership \circ ?Holding \circ hasMembership^{-}$. Intuitively, the path navigates through the roles of the ontology, using the construct $?C$ to test that the path passes through instances of C. Since the role $hasMembership$ is typed in the ontology by the concept *CompanyGroup*, the identification constraint actually says that for a certain timestamp no two juridical CCCs exists that are connected via the above path to the same company group.

Globally, we have specified more than 30 IDCs in the ontology. None of these presently correspond to integrity constraints at the data sources. This is because, as it is usual in practice, very few integrity constraints are explicitly asserted at the sources. Thus, our ontology plays an important role in representing business rules not explicitly reflected in the data repository of the organization.

EQLCs turned out to be another important means for correct domain modeling. Such constraints indeed permit to overcome some expressiveness limitations of *DL-Lite*$_{A,Id}$, without causing any computational blow up. Indeed, EQLCs are interpreted according to a suitable semantic approximation (cf. Section 2). In this experimentation we have heavily used EQLCs to express, e.g., hierarchy completeness and other important business constraints, otherwise not expressible in our ontology.

Example 3. An important constraint we want to force on the ontology is that for every customer which has a guarantor for a loan we have to know the amount of bank credit provided to the customer. This is specified through the following EQLC, which is expressed in SparSQL, a query language presented in [5] based on SPARQL and SQL:

```
EQLC( verify not exists (
   SELECT withGuarantor.cus, withGuarantor.t
   FROM sparqltable( SELECT ?cus ?t
                     WHERE{ ?cus :isLinked ?link.
                            ?link rdf:type 'GuaranteeRelations'.
                            ?link :timestamp ?t}) withGuarantor
   WHERE (withGuarantor.cus, withGuarantor.t) NOT IN (
          SELECT withCredit.cus, withCredit.t
          FROM sparqltable( SELECT ?cus ?amnt ?t
                     WHERE{  ?cus :hasLoan ?loan.
                      ?loan :creditAmount ?amnt.
                      ?loan :timestamp ?t }) withCredit ) ) )
```

The above constraint says that no customer cus exists, such that cus is connected to an instance of the concept $GuaranteeRelations$ at the time t, and cus has not a "known" $creditAmount$ at the same time t. It is worth noticing that OWL 2, despite its expressiveness, does not allow for expressing the above constraint.

Let us now turn our attention to mapping specification. The mapping specification phase has required a complete understanding and an in-depth analysis of the data sources, which highlighted some modeling weaknesses present in the source database schema: various modifications stratified in the years over the original data schema have partially transformed the data sources, which now reveal some problems related to redundancy, inconsistency, and incompleteness in the data. Localizing the right data to be mapped to ontology constructs has thus required the definition of fairly complex mapping assertions, as shown in Example 4.

Example 4. Consider the following mapping assertion specifying how to construct instances of *JuridicalCCC* using data returned by an SQL query accessing both the table $GZ0102$, which contains information about CCCs, and the table $GZ0007$, which contains information about the company groups.

```
SELECT id_cluster, timestamp_val FROM GZ0102, GZ0007
WHERE GZ0102.validity_code = 'T' AND GZ0102.id_cluster <> 0
     AND GZ0007.validity_code = 'T' AND GZ0007.id_group <> 0
     AND GZ0102.id_cluster = GZ0007.id_group
```

$\rightsquigarrow \; JuridicalCCC(ccc(id\_cluster, timestamp\_val))$

From the data source analysis it turned out that each CCC that has an identifier ($GZ0102.id\_cluster$) coinciding with the identifier of a company group ($GZ0007.id\_group$) is a juridical CCC. Such a property is specified in the SQL query in the mapping through the join between $GZ0102$ and $GZ0007$ ($GZ0102.id\_cluster = GZ0007.id\_group$). Notice that invalid tuples (those with $validity\_code$ different from 'T') and meaningless tuples (those with $id\_cluster$ or id_group equal zero) are excluded from the selection. The query returns pairs of $id\_cluster$ and $timestamp\_val$, which are used as arguments of the function $ccc()$ to build logic terms representing objects that are instances of *JuridicalCCC*, according to the method described in [12].

The mapping specification phase has produced around 200 mapping assertions, many of which are quite involved. Their design has been possible by a deep understanding of the tables involved, their attributes, and the values they store. We initially tried to automate this process with the help of current tools for automatic mapping generation, but, due to the complexity of extracting the right semantics of the source tables, we failed. This is in line with our past experience on mapping design: the bulk of the work in mapping specification has to be essentially carried out manually.

5 The system at work

In this section we discuss the actual use of MASTRO in the MPS scenario. As a general comment, we remark that the OBDA system we designed for this scenario allowed to overcome many of the data access problems we have discussed in the previous sections. In particular, querying the data sources through the conceptual view provided by the ontology enabled various kinds of users, not necessarily experts of the application managing data at the sources, to profitably access such data. In what follows, we concentrate on two crucial aspects of our experience: the use we made of MASTRO to check the quality of the data sources, and the impact that certain characteristics of

the MPS scenario have had on the evolution of the system in terms of its tuning and optimizations.

As mentioned in the introduction, we faced two main issues concerning the quality of the data sources, namely incompleteness and inconsistency in the data at the sources. Detecting data incompleteness has been possible by exploiting the MASTRO query answering services, and more precisely, by inspecting the rewriting and the unfolding that MASTRO produces in the query answering process. Let us see this on an example. To retrieve from the data sources the identification codes of all company groups, MPS operators simply use a single SQL query projecting out the `id_code` from the table *GZ0007*, which contains information about company groups. Surprisingly, using the ontology to obtain all company codes, we actually get a larger answer set, by posing over the ontology the obvious corresponding query $q(y) \leftarrow CompanyGroup(x), id\_code(x, y)$. The reason for such a difference in the answers resides in the fact that the query that MASTRO asks to the source database, and that is automatically produced by the rewriting and unfolding procedures of MASTRO, is much more complex than the query used by the MPS operators. By reasoning over the ontology, and exploiting the mapping assertions, MASTRO accesses all the source tables that store codes of company groups, and this set of tables does not in fact contain only the codes of company groups that occur in table *GZ0007*. Such a result showed that some foreign key dependencies constraining the identification codes stored in the table *GZ0007* were in fact missing in the source database, and that such a table should not been considered complete with respect to such information.

We turn now to data inconsistency issues. In *DL-Lite*$_{A,Id}$, inconsistencies are caused by data that violate the assertions of the ontology, specifically disjointness assertions, functionality constraints, identification constraints, and EQL constraints. Also, causes of inconsistencies can be easily localized by retrieving the minimal set of data that produce each single violation. We actually modified the classical consistency check of MASTRO in order to identify the offending data, in particular exploiting the feature of answering EQL queries (cf. Section 2) and their ability to express negation. Consider for example the relation $linkedTo$, which is declared to be inverse functional (i.e., (funct $linkedTo^-$)). In order to detect the violation of such constraint and the guilty data, we use the following EQL query:

```
SELECT testview.l, testview.c1, testview.c2
FROM sparqltable (SELECT ?l ?c1 ?c2
             WHERE{?c1:linkedTo?l. ?c2:linkedTo?l}) testview
WHERE testview.c1 <> testview.c2
```

Switching our attention to the performance of the system, there are two sources of complexity to be considered in the query answering and consistency checking services provided in MASTRO, the query *reformulation* and query *unfolding* procedures. Reformulation introduces complexity since it may produce an exponential number of queries to be answered. Nevertheless, in the case of the MPS ontology, this potential drawback did not occur. Indeed, in most cases, the number of queries produced by this step was small (between 1 and 25). In contrast, the query unfolding step presented challenges that led to several important improvements in MASTRO, briefly discussed below.

In complex scenarios, such as the one we considered in our experimentation, we found that the most critical aspect for performance is what we call *query structure*, i.e.,

the form of the SQL queries issued to the source database. Query structure is characterized by the specific technique used to produce SQL queries out of queries formulated over TBox predicates ($\mathcal{T}$-predicates).

In MASTRO, query unfolding is based on the use of *SQL views* over the source database. More specifically, the mapping is first pre-processed so as to have only assertions in which the query over the ontology contains just one predicate (splitting). Then, all assertions referring to the same $\mathcal{T}$-predicate are combined together in order to have one SQL view, which we call $\mathcal{T}$*-view*, for each predicate. Essentially, the view is obtained taking the union of the SQL queries occurring in the left-hand side of the assertions, and pushing the construction of logic terms representing instances of concepts and roles in the view itself. Unfolding a query specified over $\mathcal{T}$-predicates amounts therefore to simply unfold each query atom with the corresponding $\mathcal{T}$-view. For example, if the split mapping assertions for the role *linkedTo* are

m_1: `SELECT .... WHERE cd_tp = 503` $\rightsquigarrow linkedTo(\mathbf{cus}(idcus), \mathbf{link}(linkid))$
m_2: `SELECT .... WHERE cd_tp = 501` $\rightsquigarrow linkedTo(\mathbf{cus}(idcus), \mathbf{link}(linkid))$

then, the following view, `linkedto_Tview`, is associated to the *linkedTo* predicate:

```
SELECT 'cus('||idcus||')' as term1, 'link('||linkid||')' as term2
FROM (SELECT .... WHERE cd_tp = 503) view_m1
UNION
SELECT 'cus'(||idcus||')' as term1, 'link('||linkid||')' as term2
FROM (SELECT ....  WHERE cd_tp = 501) view_m2
```

Notice that in the `SELECT` clause we build logical terms by means of simple SQL string concatenation operations, indicated with the `||` operator. Then, the query $q(X) \leftarrow linkedTo(X, Y)$ is unfolded into `SELECT term1 FROM linkedto_Tview`.

Despite its simplicity, we found out that, in scenarios characterized by a high volume of data and complex and numerous mapping assertions, this approach fail, due to low performance of the generated queries. For example, in our test cases, queries with a single atom that involve database relations with high volume of data often required several minutes to be answered. More complex queries, with more than 2 atoms and involving also big relations, would often require hours or would even not terminate. The reason for this bad performance is in the limitations of DBMS query planners in handling subqueries in the `FROM` clause, and joins between terms representing objects, rather than directly on database values. What we observed is that, in order to deal with subqueries, query planners rely on a process called query *flattening*, in which the query planner attempts to rephrase a query with subqueries into a new query with no subqueries. If the query planner is not successful in this attempt, e.g., due to the complexity of the subqueries, it will resort to subquery materialization, an extremely expensive operation when the volume of data is high.

In order to avoid materialization and joins between object terms, and in general, to increase the chances of the query planner to produce a *good plan*, we devised a strategy that led us to produce queries that are as simple as possible with respect to subqueries. This led us to adopt what we call an $\mathcal{M}$*-view* approach to unfolding. In this approach, we build simpler views, one for each SQL query in the split mapping assertions, and we associate *all* of them to the corresponding $\mathcal{T}$-predicate. For example, in the previous case we would define the two views below

```
view_m1 = SELECT .... WHERE cd_tp = 503
view_m2 = SELECT .... WHERE cd_tp = 501
```

and the unfolding of the query $q(X) \leftarrow linkedTo(X, Y)$ would be as follows

```
SELECT 'cus('||idcus||')' FROM view_m1
UNION
SELECT 'cus('||idcus||')' FROM view_m2
```

Notice that in this case, the construction of the object term `'cus('||idcus||')'` is in the external `SELECT` clause and is not pushed into the views in the `FROM` clause.

What is important to note here is the exchange of simplicity of the unfolding procedure for simplicity of the structure of the queries being generated (i.e., less nesting in the subqueries) and a new exponential growth in the amount of queries sent to the database, e.g., now for every $linkedTo$ atom in a query, we will produce an SQL query taking the union of at least two queries, one where we only use `view_m1` and one with `view_m2`. Although this growth could seem problematic, we have found that the increase in the performance of executing each individual query pays off the increase in the number of queries to be executed. Moreover, since these queries are independent, we can use parallelism in query execution to improve performance even more.

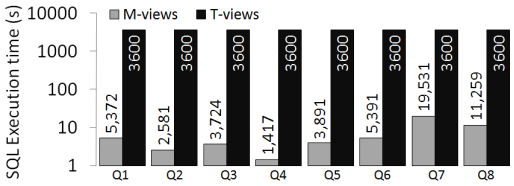

Fig. 2. $\mathcal{M}$-views vs. $\mathcal{T}$-views using an execution timeout of 1hr.

To give an idea of the effectiveness of the described optimizations, we present in Figure 2 the data about the execution of a collection of 8 representative queries (the units of the vertical axis are seconds). These queries are all of interest to MPS, and challenging in terms of number of atoms, complexity of the unfolding and the volume of data accessed.

6 Conclusions

From the point of view of MPS, the project has provided very useful results in various areas of interest:

– Data integration, providing the capability of accessing application data in a unified way, by means of queries written at a logical/conceptual level by end-users not necessarily acquainted with the characteristics of the application;

– Database quality improvement, providing tools for monitoring the actual quality of the database, both at an intensional and an extensional level;

– Knowledge sharing, providing, with the ontology-based representation of the application domain, an efficient means of communicating and sharing knowledge and information throughout the company.

The plan is to continue the experience by extending the work to other MPS applications, with the idea that the ontology-based approach could result in a basic step for the future IT architecture evolution, oriented towards Service-oriented architectures and Business Process Management.

References

1. P. A. Bernstein and L. Haas. Information integration in the enterprise. *Comm. of the ACM*, 51(9):72–79, 2008.
2. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati. Ontologies and databases: The *DL-Lite* approach. In S. Tessaris and E. Franconi, editors, *Reasoning Web Summer School 2009*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.
3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of IJCAI 2007*, pages 274–279, 2007.
4. 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.
5. C. Corona, E. D. Pasquale, A. Poggi, M. Ruzzi, and D. F. Savo. When OWL met DL-Lite... In *SWAP-08*, 2008.
6. J. Dolby, A. Fokoue, A. Kalyanpur, L. Ma, E. Schonberg, K. Srinivas, and X. Sun. Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In *Proc. of ISWC 2008*, volume 5318 of *LNCS*, pages 403–418. Springer, 2008.
7. L. M. Haas. Beauty and the beast: The theory and practice of information integration. In *Proc. of ICDT 2007*, volume 4353 of *LNCS*, pages 28–43. Springer, 2007.
8. J. Henrard, D. Roland, A. Cleve, and J.-L. Hainaut. Large-scale data reengineering: Return from experience. In *WCRE '08: Proceedings of the 2008 15th Working Conference on Reverse Engineering*, pages 305–308. IEEE Computer Society, 2008.
9. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.
10. C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In *Proc. of IJCAI 2009*, pages 2070–2075, 2009.
11. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic*, 2009. To appear.
12. 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.
13. A. Poggi, M. Rodriguez, and M. Ruzzi. Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. In K. Clark and P. F. Patel-Schneider, editors, *Proc. of OWLED 2008 DC*, 2008.
14. M. Rodriguez-Muro, L. Lubyte, and D. Calvanese. Realizing ontology based data access: A plug-in for Protégé. In *Proc. of IIMAS 2008*, pages 286–289. IEEE CS Press, 2008.