Semantic Web ? (2011) 1–11
IOS Press

The Mastro System for Ontology-based Data Access

Editor(s): Thomas Lukasiewicz, Oxford University, UK
Solicited review(s): Carsten Lutz, Universität Bremen, Germany; Roman Kontchakov, Birkbeck College London, UK; one anonymous reviewer

Diego Calvanese [a] Giuseppe De Giacomo [b] Domenico Lembo [b] Maurizio Lenzerini [b] Antonella Poggi [b] Mariano Rodriguez-Muro [a] Riccardo Rosati [b] Marco Ruzzi [b] and Domenico Fabio Savo [b]

[a] *Free University of Bozen-Bolzano, Piazza Domenicani 3, I-39100, Bolzano, Italy*
Email: lastname@inf.unibz.it
[b] *Sapienza Universita di Roma,Via Ariosto 25, I-00185, Roma, Italy*
Email: lastname@dis.uniroma1.it

Abstract. In this paper we present Mastro, a Java tool for ontology-based data access (OBDA) developed at the University of Rome "La Sapienza" and at the Free University of Bozen-Bolzano. Mastro manages OBDA systems in which the ontology is specified in *DL-Lite*$_{A,id}$, a logic of the *DL-Lite* family of tractable Description Logics specifically tailored to ontology-based data access, and is connected to external JDBC enabled data management systems through semantic mappings that associate SQL queries over the external data to the elements of the ontology. Advanced forms of integrity constraints, which turned out to be very useful in practical applications, are also enabled over the ontologies. Optimized algorithms for answering expressive queries are provided, as well as features for intensional reasoning and consistency checking. Mastro provides a proprietary API, an OWLAPI compatible interface, and a plugin for the Protégé 4 ontology editor. It has been successfully used in several projects carried out in collaboration with important organizations, on which we briefly comment in this paper.

Keywords: Ontology-based data access, Description Logics, Reasoning over ontologies

1. Introduction

In this paper we present Mastro, a tool for ontology-based data access developed at the University of Rome "La Sapienza" and at the Free University of Bozen-Bolzano. Ontology-based data access (OBDA) refers to a setting in which an ontology is used as a high-level, conceptual view over data repositories, allowing users to access data without the need to know how they are actually organized and where they are stored (cf. Figure 1).

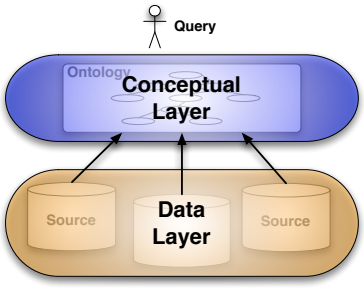

Fig. 1. Ontology-based Data Access

The OBDA approach turns out to be very useful in all scenarios in which accessing data in a unified and coherent way is difficult. This may happen for several reasons. For example, databases may have undergone several manipulations during the years, often for optimizing applications using them, and may have lost their original design. They may have

been distributed or replicated without a coherent design, so that the information turns out to be dispersed over several independent (maybe heterogeneous) data sources, and source data tend to be redundant and mutually inconsistent.

Through Mastro it is possible to design and manage OBDA systems, i.e., systems in which an ontology is connected to external data sources through mappings. As in data integration systems [22], we use mappings to specify the semantic correspondence between a unified view of the domain (called global schema in data integration terminology) and the data stored at the sources. The distinguishing feature of the OBDA approach, however, is the fact that the global unified view is given in terms of a conceptualization of the domain of interest, constructed independently from the representation adopted for the data stored at the sources. This choice provides several advantages: it allows for a declarative approach to data access and integration and provides a specification of the domain that is independent from the data layer; it realizes logical/physical independence of the information system, which is therefore more accessible to non-experts of the underlying databases; The conceptual approach to data access does not impose to fully integrate the data sources at once, as often happens in data integration mediator-based system, but the design can be carried out in an incremental way; The conceptual model available on the top of the system provides a common ground for the documentation of the data stores and can be seen as a formal specification for mediator design.

Mastro has solid theoretical basis [3,5,4,25]. The ontologies it manages are specified in *DL-Lite*$_{A,id}$, a logic of the *DL-Lite family* of tractable Description Logics (DLs), which are specifically tailored to the management and querying of ontologies in which the extensional level, i.e., the data, largely dominates the intensional level. From the point of view of the expressive power, *DL-Lite*$_{A,id}$ captures the main modeling features of a variety of representation languages, such as basic ontology languages and conceptual data models. Furthermore, it allows for specifying advanced forms of identification constraints [6]. General forms of integrity constraints, which essentially corresponds to generic first-order sentences, are also expressible over the ontology. We call these constraints *EQL constraints* and interpret them according to the so-called *epistemic semantics*, which is an approximation of first-order semantics adopted for the other *DL-Lite*$_{A,id}$ axioms that ensures decidability and tractability of reasoning [4]. We notice that the ability to specify both identification and expressive integrity constraints turned out to be very useful in practical experiences we conducted with Mastro [1,27], and that such constructs are not part of OWL 2, the current W3C standard language for specifying ontologies.

The mapping mechanism adopted by Mastro [25] allows for solving the so-called *impedance mismatch* problem, arising from the fact that, while the data sources store values, the instances of concepts in the ontology are objects. Answering unions of conjunctive queries in OBDA systems managed by Mastro can be done through a very efficient technique that reduces this task to standard SQL query evaluation. Indeed, conjunctive query answering has been shown to be in LogSpace (in fact in AC^0) w.r.t. data complexity, i.e., the complexity measured only w.r.t. the extensional level [5,25], which is the same complexity of evaluating SQL queries over plain relational databases. Even though very slight extensions of the expressive abilities of our system lead beyond this complexity bound [3], also queries that are more powerful than UCQs can be processed in Mastro via a similar SQL encoding. Such queries, which we call *EQL queries*, essentially correspond to all first-order queries expressible over the ontology, and are interpreted under the epistemic semantics [4].

Mastro is developed in Java and can be connected to any data management system allowing for a JDBC connection, e.g., a relational DBMS. In those cases in which several, possibly non-relational, sources need to be accessed, Mastro can be coupled with a relational data federation tool[1], which wraps sources and represents them as a single (virtual) relational database.

Mastro comes with its proprietary API, but is equipped also with an OWLAPI compatible interface that has been developed for interaction with OWLAPI compliant applications. In particular, such an interface has been exploited to implement

[1] E.g., IBM WebSphere Application Server (`http://www.ibm.com/software/webservers/appserv/was/`), Oracle Data Service Integrator (`http://www.oracle.com/us/products/middleware/data-integration/`).

the MASTRO plugin for the Protégé 4 ontology editor[2]. MASTRO is currently available for download at `http://www.dis.uniroma1.it/~quonto/`.

The rest of the paper is organized as follows. In Section 2, we briefly describe the framework of ontology-based data access. In Section 3, we provide an in-depth description of the main modules in which MASTRO is organized, briefly describing the procedures and algorithms they realize. In Section 4, we report on three main use cases in which MASTRO has been successfully trialed. In Section 5, we discuss related work, and in Section 6 we conclude the paper.

2. Ontology-based data access

In OBDA, the aim is to give users access to a data source or a collection thereof, by means of a high-level conceptual view specified as an ontology. The ontology is usually formalized in Description Logics (DLs) [2], which are logics that allow one to represent the domain of interest in terms of *concepts*, denoting sets of objects, *roles*, denoting binary relations between objects, and *attributes*, denoting relations between objects and values from predefined domains (such as strings, integers, etc.). A DL ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of a TBox $\mathcal{T}$, representing intensional knowledge, and an ABox $\mathcal{A}$ representing extensional knowledge.

MASTRO is able to deal with DL TBoxes that are expressed in *DL-Lite*$_{A,id}$, a member of the *DL-Lite* family of lightweight DLs [5]. In such DLs, a good tradeoff is achieved between the expressive power of the TBox language used to capture the domain semantics, and the computational complexity of inference, in particular when such a complexity is measured w.r.t. the size of the data. We don't specify here the formal syntax and semantics of *DL-Lite*$_{A,id}$, for which we refer to [6], but state only that this logic essentially captures standard conceptual modeling formalisms, such as UML Class Diagrams and Entity-Relationship (ER) Schemas. Indeed, *DL-Lite*$_{A,id}$ distinguishes at the semantic level between abstract objects and domain values, and allows one to express in a TBox the following kinds of logical assertions: *(i)* inclusion assertions between concepts (that include projections of roles on one of their components), expressing ISA between them, typing of relations, mandatory participation to roles or attributes, and disjointness between concepts (if the negation of a concept occurs in the right-hand side of the inclusion); *(ii)* inclusion assertions between roles and attributes, to express ISA between roles and attributes, and disjointness between roles and attributes (if negation is used in the right-hand side of the inclusion); *(iii)* functionality assertions, and complex forms of identification constraints[3]. An ABox contains assertions about specific individuals or values, such as the fact that an individual is an instance of a concept, that two individuals are related by a role, or that an attribute relates an individual to some value.

In OBDA, the extensional level is not represented directly by an ABox, but rather by a database that is connected to the TBox by means of suitable mapping assertions[4]. Such *mapping assertions* have the form $\Phi \rightsquigarrow \Psi$, where Φ, called the *body* of the assertion, is an arbitrary SQL query over the underlying database, and Ψ, called the *head*, is a conjunction of atoms whose predicates are the concepts, roles, and attributes of the TBox. Intuitively, such a mapping assertion specifies that the tuples returned by the SQL query Φ are used to generate the facts that instantiate the concepts, roles, and attributes in Ψ. Notice that, due to the fact that Ψ is a conjunction of atoms (as opposed to a query, possibly with existentially quantified variables), such mappings can be considered as a special form of *global-as-view* (GAV) mappings [22] (cf. also Section 5). Indeed, in order to overcome the so-called *impedance mismatch* between the database, storing values, and the TBox, maintaining objects, the mapping assertions are used to specify how to construct abstract objects from the tuples of values retrieved from the database. This is done by allowing one to use function symbols in the atoms in Ψ: together with the values retrieved by Φ, such func-

[2] `http://protege.stanford.edu/`

[3] Thanks to identification constraints we are able in *DL-Lite*$_{A,id}$ to also model, via reification, n-relations between concepts typical of UML Class Diagrams and ER schemas.

[4] Note that, in the following, with some abuse of terminology, when we use the term "ontology" in the context of OBDA, we implicitly refer to a TBox only.

tion symbols generate so called *object terms*, which serve as object identifiers for individuals in the ontology. We notice that the semantics we adopt in MASTRO (see also below) establishes that different terms denote different objects (unique name assumption), so that different terms never need to be equated during reasoning, which is coherent with the assumption of not having existentially quantified variables in the body of mappings.

As an example, consider the mapping assertion

$$\texttt{SELECT SSN, name FROM TABPERS WHERE age <= 5} \rightsquigarrow \mathit{Child}(\mathbf{p}(\texttt{SSN})),\ \mathit{Name}(\mathbf{p}(\texttt{SSN}), \texttt{name})$$

which specifies how to construct instances of the concept *Child* and the attribute *Name* in the ontology starting from the database relation `TABPERS` having among its columns `SSN`, `name` and `age`. As shown in the example, the mappings used in MASTRO allow one to establish correspondences between elements, and instances thereof, belonging to schemas expressed in different languages (and over different alphabets), acting thus as a powerful reconciling mechanism.

The semantics of DLs is given in terms of standard first-order interpretations. Traditional intensional reasoning tasks w.r.t. a given TBox are concept satisfiability and concept subsumption [2]. Among the extensional reasoning tasks w.r.t. a given ontology $\langle \mathcal{T}, \mathcal{A} \rangle$, the most relevant ones are *ontology satisfiability*, i.e., checking whether $\langle \mathcal{T}, \mathcal{A} \rangle$ admits at least one model, and *query answering*, which amounts to computing the certain answers to queries. The *certain answers* to a query Q over $\langle \mathcal{T}, \mathcal{A} \rangle$ are those tuples that are in the evaluation of Q in every model of $\langle \mathcal{T}, \mathcal{A} \rangle$. For the logics of the *DL-Lite* family it has been shown that for unions of conjunctive queries (UCQs), under the unique name assumption, query answering can be carried out efficiently in the size of the data, by reducing it to SQL query evaluation over the ABox seen as a database [5]. Also satisfiability, which is easily reducible to query answering, can be solved through the same mechanism. All the notions given above can be easily generalized to OBDA systems, where a TBox $\mathcal{T}$ is connected to an external database $\mathcal{D}$ through mappings $\mathcal{M}$, denoted $\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$. In particular, the *models* of $\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ are those interpretations of $\mathcal{T}$ that satisfy the assertions in $\mathcal{T}$ and that are consistent with the tuples retrieved by $\mathcal{M}$ from $\mathcal{D}$ (see [25] for the formal details). Satisfiability amounts to checking whether $\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ admits at least one model, while answering a query Q amounts to computing the tuples that are in the evaluation of Q in every model of $\langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$. When $\mathcal{T}$ is a *DL-Lite*$_{A,id}$ TBox and $\mathcal{M}$ is a set of assertions of the form $\Phi \rightsquigarrow \Psi$ described above, both satisfiability and conjunctive query answering can be reduced to SQL query evaluation through an extension of the techniques for *DL-Lite*$_{A,id}$ ontologies $\langle \mathcal{T}, A \rangle$ mentioned above. Such techniques are implemented in MASTRO and are briefly described in Section 3.1, whereas we refer to [5,25] for a more complete treatment.

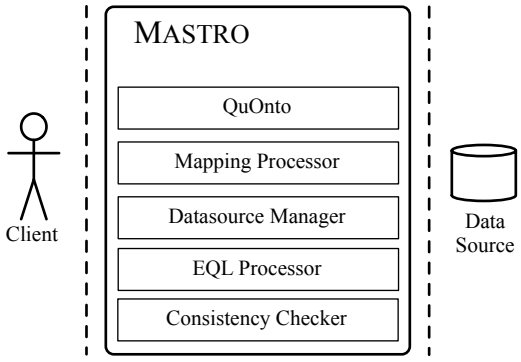

Fig. 2. MASTRO basic architecture

3. Architecture of MASTRO

In this section, we describe the general architecture of MASTRO, and some of the important aspects of the modules that constitute the system, shown in Figure 2. We first present each such module and the functionalities that concur to realize the services that MASTRO offers. Such services amounts to intentional reasoning, conjunctive query answering, mapping management, EQL query and constraints management, and consistency checking. We then provide the description of the interfaces available for accessing MASTRO's functionalities.

3.1. The MASTRO *modules*

QUONTO *module.* QUONTO is a reasoner for *DL-Lite*$_{A,id}$ that provides intentional reasoning services, i.e., concept satisfiability, concept sub-

sumption, etc., as well as *reformulation* of UCQs. In short, the reformulation process takes as input a UCQ Q expressed over a *DL-Lite*$_{A,id}$ TBox $\mathcal{T}$ and returns a UCQ Q_r that represents the *perfect reformulation of Q with respect to $\mathcal{T}$*. The evaluation of Q_r over any *DL-Lite*$_{A,id}$ ABox $\mathcal{A}$ returns the certain answers to Q with respect to $\langle \mathcal{T}, \mathcal{A} \rangle$. The reformulation engine implemented in QUONTO is based on the PerfectRef algorithm presented in [5], adapted to deal natively with OWL constructs captured by *DL-Lite*$_{A,id}$ and not explicitly considered in [5], e.g., qualified existential restrictions in the right-hand side of concept inclusions (see [10] for details). Also, the reformulation is enhanced with optimizations that allow for reducing the number of queries it generates.

Mapping Processor module. Since MASTRO does not explicitly manage ABoxes, but rather accesses data stored in external systems via mappings, the set of queries Q_r is not evaluated over an ABox, but processed according to the mappings to obtain a query that can be evaluated over the data sources. Indeed, Q_r is phrased in terms of concepts and roles of the TBox $\mathcal{T}$ and to obtain a query expressed in terms of the vocabulary of the sources, it is necessary to perform a further rewriting step dependent on the mapping, which, roughly, substitutes the TBox predicates occurring in Q_r with their definitions provided by the mapping assertions. Such a process is called *query unfolding* and is carried out by the Mapping Processor module. Generally speaking, query unfolding is quite a straightforward procedure, widely applied in data integration applications. In MASTRO, however, query unfolding is complicated by the presence of function symbols in mapping assertions (see Section 2). To deal with such aspects, the Mapping Processor implements the partial evaluation-based unfolding technique from [25], which we briefly summarize in the following: *(i)* we "split" each mapping assertion $\Phi(\vec{v}) \rightsquigarrow \Psi(\vec{w})$, where $\vec{v}$ is a sequence of variables and $\vec{w}$ is a sequence of terms, into a set of assertions having exactly one ontology predicate in the head, i.e., for each predicate symbol S occurring in $\Psi(\vec{w})$, we write an assertion of the form $\Psi(\vec{v}) \rightsquigarrow S(\vec{w}_s)$, where the terms in $\vec{w}_s$ are contained in $\vec{w}$; *(ii)* we associate an auxiliary predicate *aux* to each SQL query $\Phi(\vec{v})$, thus obtaining a set of assertions of the form $aux(\vec{v}) \rightsquigarrow S(\vec{w}_s)$; *(iii)* we transform each such assertion into a logic program clause of the form $S(\vec{w}_s) :- aux(\vec{v})$; *(iv)* we compute the partial evaluation of Q_r with respect to the logic program gathering all clauses constructed in the previous steps, i.e., a new set of conjunctive queries phrased only in terms of the auxiliary predicates[5]; and, last, *(v)* we translate the partial evaluation into an SQL query over the sources, by replacing each auxiliary predicate with the associated SQL view. The use of partial evaluation and auxiliary predicates gives us the flexibility to work on the unfolding process at an abstract level, independently from the type of data sources and the specific form of the views that we associate to the auxiliary predicates. Further details on the technique can be found in [25,27].

Datasource Manager module. This module is responsible for maintaining the connections to the data sources, for coordinating query execution, and for the management of database resources such as pointer maintenance and transaction management. The most relevant feature of this module is the ability to parallelize the execution of the queries in the perfect reformulation to improve query answering performance. This feature is key for fast query processing even in the case where a very big number of queries is generated by the reformulation-unfolding process. Depending on the system configuration, several execution threads are spawned and queries are assigned to the different execution threads. Each thread manages a group of queries, which are executed sequentially within the thread itself. When the processing of any of these queries terminates, the result set associated to the answer is forwarded to a *result set wrapper* that keeps receiving the results of subsequent queries, while forwarding them progressively to the client.

EQL Processor module. This module provides the ability to specify and execute EQL queries (cf. Section 1). Syntactically, an EQL query is an SQL query specified over virtual relations expressed as UCQs over the ontology. As for the semantics, answering an EQL query consists in computing the extension of each virtual relation, i.e., its certain answers with respect to the ontology, the mapping

[5] The term *partial evaluation* is due to the connection of this technique with the optimization technique from the logic programming literature that carries the same name.

and the source data, and evaluating the SQL query over all such extensions. This actually corresponds to putting each virtual relation under the scope of an epistemic operator (see [4] for details). Rather than computing the extension of the virtual relations, the EQL Processor exploits the reformulation service offered by the QuOnto module, and the unfolding service provided by the Mapping Processor module, in order to rewrite each inner query into an SQL query over the sources, thus transforming the entire EQL query into an SQL query over the sources. Such a query is then sent to the Datasource Manager, which is in charge of evaluating it. EQL queries are extremely useful when the expressive power of UCQs is not enough. For example, they allow for expressing negation and comparison in queries. Also, through EQL queries it is possible to specify powerful integrity constraints over the ontology (as shown in [9]), which go beyond the expressivity of the DLs of the *DL-Lite* family.

Consistency Checker module. This module allows Mastro to verify whether an OBDA system it manages is satisfiable. By virtue of the characteristics of *DL-Lite*$_{A,id}$, such a check can be reduced to answering suitable queries posed over the ontology, each one associated to a TBox assertion or to an EQL constraint that can be violated by data at the sources. The Consistency Checker therefore relies on the services for conjunctive and EQL query answering provided by the modules we described above. Indeed, apart from basic features, which enable for checking the violation of functionality and disjointness constraints, the Consistency Checker allows one to verify consistency of identification and EQL constraints, which is reduced to answering EQL queries over the ontology. The Consistency Checker can also localize data that violate TBox assertions and/or constraints over the ontology. It can indeed generate those queries (UCQs or EQL) whose answers return data that cause inconsistencies.

3.2. Interfaces

Mastro's functionalities can be accessed in three ways: through a proprietary API, through an OWLAPI compatible interface, and by means of a plugin for the Protégé 4 ontology editor. In particular, Mastro's proprietary API is used to integrate all the modules that compose the system. This API is also used to implement specific procedures required during the deployment of the tool in application scenarios, as the ones described in Section 4. The API also provides parser facilities for loading ontologies with mappings using Mastro's own XML and functional-style syntax [9] and allows for accessing all the functionalities offered by the system.

The OWLAPI compatible interface is built on top of Mastro's API. This public interface allows for a straightforward integration of Mastro with OWLAPI[6]-OBDALib[7] applications. The main access point of this API is the so-called MastroOWLReasoner, an implementation of the OWLReasoner interface from the OWLAPI, and of the OBDAReasoner interface, which is part of the OBDALib. Through functionalities provided by the OWLReasoner interface, clients have access to Mastro's services associated with traditional OWL reasoners, i.e., concept subsumption, satisfiability, etc. Through the OBDAReasoner interface, clients can access Mastro's OBDA related functionalities, i.e., specification of ontology with mappings, conjunctive query answering, etc.

In order to provide its functionalities, the OWLAPI compatibility layer relies on an OWL to *DL-Lite*$_{A,id}$ translator module that is able to process OWL 2 ontologies represented by means of OWLAPI objects and produce *DL-Lite*$_{A,id}$ ontologies represented by Mastro's internal API objects. In particular, OWL 2 being a very expressive language, it may happen that some OWL 2 assertions cannot be translated into corresponding *DL-Lite*$_{A,id}$ assertions, and hence are disregarded by the module.

Last but not least, Mastro can also be accessed by means of a Protégé 4 plugin. This plugin is built on top of the MastroOWLReasoner (cf. Figure 3) and exposes all functionalities of Mastro, allowing for the use of the facilities offered by Protégé for ontology editing and by the OBDA Plugin for editing of mappings towards external data sources. The Mastro plugin extends Protégé with new features to express assertions that are not part of the OWL 2 language but that are supported by Mastro, such as identification and EQL constraints.

[6] http://owlapi.sourceforge.net/
[7] http://obda.inf.unibz.it/

Fig. 3. MASTRO interfaces

4. The system at work: experiences on real cases

In order to demonstrate the usefulness of OBDA and the feasibility and efficiency of the MASTRO system, we report on three real world applications in which it has been experimented.

SELEX Sistemi Integrati. SELEX Sistemi Integrati (SELEX-SI) is a Finmeccanica Company, and is a world leader in the provision of integrated defence and air traffic control systems. We considered Configuration and Data Management (C&DM) in SELEX-SI and focused on a significant portion of the data manipulated in this context [1].

C&DM is a technical management model that is central to all SELEX-SI activities since it governs the entire products' life cycle. We mainly focused on the data concerning the design and the production of components that are used to realize complex systems, the physical deployment of such components, and the analysis of their obsolescence. At the time of the experimentation, such data were stored in various, partially overlapping and continuously evolving sources, and managed by five different systems under diverse data models (relational, XML-based, etc.). We used MASTRO to integrate such data, in such a way that relevant queries connected to important C&DM informational needs could be automatically processed.

Specifically, we first used the external data federation tool Websphere Federation Server[8] to present to MASTRO all the data sources as a single relational database. Thus, we obtained a relational schema managed by Websphere, with around 50 relational tables, with an average of 15 attributes each.

After the federation step described above, we conducted an in-depth analysis of the C&DM domain, which led to an ontology

A challenge in this experimentation has been providing the possibility of comparing analogous data from different sources, e.g., to answer queries like "is the component declared obsolete in source A obsolete also according to source B?". This required to introduce some predicates in the ontology to explicitly represent the data sources and pieces of information stored in each source.

The outcome of the experimentation was very promising. In particular, it demonstrated the usefulness of MASTRO to easily and efficiently access information in all cases where a user would query separately each data source, and manually combine the single answers. This appeared to be even more remarkable, given that the data sources were known to evolve rapidly. Also, the possibility of comparing through the ontology the data content of different sources turned out to be crucial for the success of the experimentation.

Monte dei Paschi di Siena. Within a joint project with Banca Monte dei Paschi di Siena (MPS)[9], Free University of Bozen-Bolzano, and SAPIENZA Università di Roma, we used MASTRO for accessing a set of data sources from the actual MPS data repository by means of an ontology [27]. In particular, we focused on the data exploited by MPS personnel for risk estimation in the process of granting credit to bank customers. 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. Such source data are managed by a dedicated application, which is in charge of guaranteeing data integrity (in fact, the underlying database does not force constraints on data). Not only the application performs various updates, but data is updated on a daily basis to identify connections between customers that are relevant for the credit rating estimation.

The main challenge that we tackled within the experimentation was the ontology and mapping design. This was a seven man-months process that required to both inspect the data source and inter-

[8] `http://www-306.ibm.com/software/data/integration/federation_server/`.

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

view domain experts, and was complicated by the fact that the source was managed by a specific application. The resulting OBDA system is defined in terms of approximately 600 *DL-Lite*$_{A,id}$ assertions over 79 concepts, 33 roles and 37 attributes, and 200 mapping assertions.

The experimentation showed that the usefulness of the Mastro system goes beyond data integration applications and embraces data quality management. In particular, it confirmed the importance of several distinguished features of our system, namely, identification constraints and epistemic constraints, which have been used extensively to model important business rules. Checking that such rules were satisfied by data retrieved from the sources through mappings has been the main objective of the project. With respect to this, we highlight two kinds of data quality problems that we were able to detect, one related to unexpected incompleteness in the data sources, and the other one related to inconsistency in the data.

Our work has also 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. At present we are still working with MPS in order to extend the work to cover the core domain of the MPS information system, with the idea that the ontology-based approach could result in a basic step for the future IT architecture evolution.

Network inventory systems. Finally, we briefly mention an experimentation we carried out in the telecommunication context, and specifically in the domain of network inventory systems. Within this experimentation, the customer was interested in accessing, through a conceptual view, a 3 million tuples source database, stored in 45 relational tables managed by the Oracle 10g RDBMS.

The distinguishing characteristic of this experimentation is the size of the OBDA system we realized. The ontology is formed by a *DL-Lite*$_{A,id}$ TBox involving 112 concepts, 84 roles, and 15 attributes, and consisting of 920 axioms, among which 91 are identification constraints and EQL constraints. Furthermore, the mapping layer is formed by 348 mapping assertions.

As well as for the MPS case study, in this case the experience demonstrated the usefulness of Mastro for data quality management. Actually, the experimentation revealed a huge amount of "dirty data" in the exploited data source and allowed the customer to gain an insight into where the data dirtiness came from. On the other hand, the experimentation showed an actual need for meta-level management capabilities. This need appears to be aligned with the work on higher-order ontology languages we are currently carrying out [12], which however requires further investigation to be incorporated within Mastro.

According to the idea that the quality of the data stored in the sources can be measured in terms of the amount of data respecting the constraints implied by the domain description offered by the ontology, we laid down the basis of a methodology for using Mastro for data quality management. The last two experiences we have mentioned, have revealed how the same ontology can be used both for querying data and for checking the quality of data sources. The different objectives in the two cases only partially influence the design of the mapping, whereas the ontology design turns out to be an independent task. Another important lesson concerns the mapping generation: according to our experiences, due to the complexity of extracting the right semantics of the source tables, the bulk of the work in mapping specification has to be essentially carried out manually.

5. Comparison with other approaches and tools

To the best of our knowledge, Mastro is the only system currently available that allows for both sound and complete conjunctive query answering over an ontology and for connecting it to external data sources with powerful mappings, and that at the same time is very efficient in doing this, even in data intensive applications. This is possible by virtue of the nice computational characteristics of *DL-Lite*, which still hold when it is used in combination with the mappings allowed in Mastro.

Differently from QuOnto, the reasoning engine at the basis of Mastro, other well-known DL reasoners such as RacerPro [17], Pellet [29], Fact++ [32], and HermiT [28] are essentially focused on standard DL reasoning services, whereas

only limited forms of query answering are supported, i.e., instance checking/retrieval or *grounded* conjunctive query answering. Grounded conjunctive queries are essentially characterized by the fact that general joins typical of CQs are performed only on individuals explicitly mentioned in the ABox. For this reason some of the entailed answers to CQs are missed in grounded conjunctive query answering, thus only approximating computation of certain answers. Although some optimizations have been implemented, such systems are not able to deal with very large ABoxes (e.g., with several millions of membership assertions) as the ones we considered in our experiments. This is mainly due to the inherent computational complexity of answering queries in the expressive DL languages supported by the above mentioned systems.

In KAON2 [19], reasoning is not based on tableaux calculus, as in the above systems, but on a reduction of $\mathcal{SHIQ}$ ontologies (an expressive fragment of OWL DL) into Disjunctive Datalog. Experimental results show that KAON2 outperforms Pellet and Racer for ontologies with simple TBoxes and large ABoxes [18]. Such ABoxes, however, do not contain more than hundreds of thousands of assertions, which is in general exceeded in OBDA applications, and again only *grounded* conjunctive query answering is supported.

The system SHER [13] implements algorithms based on an *ABox summarization* technique, which are aimed at scalable *grounded* conjunctive query answering over $\mathcal{SHIQ}$ ontologies. This approach requires to manipulate the ABox, which is stored in an RDBMS, and is therefore not extendible to an OBDA scenario, where data sources are in general outside of the control of the integration system and are accessed at query time only.

Both OWLIM[10] and Oracle 11g[11] allow for native storage and management of RDF data and support reasoning for RDFS and some fragments of OWL 2, including the tractable profile OWL 2 RL[12], for which conjunctive query answering has been shown to be PTIME-complete[13]. In both such tools inference is performed ahead query time, and inferred triples are materialized. As for SHER, this makes such tools not directly extendible to an OBDA scenario. Furthermore, the form of reasoning they support is not fully characterized from a formal point of view.

Differently from the above systems, the Virtuoso object-relational database engine[14] provides a SPARQL access to RDF data with reasoning support at query time. However, inference is limited to consider very few RDFS and OWL assertions, and is therefore in general incomplete for OWL and its standard fragments.

Apart QUONTO, other *DL-Lite* based approaches and reasoners have been developed. In [20] an alternative approach to query answering is presented. Besides a (less complex) query reformulation step, such an approach requires to suitably "extend" the ABox (managed by a RDBMS) with the aim of reducing the amount of rewritten queries produced by the reformulation step. Results given by first experiments support well this approach (notice that in QUONTO the size of the reformulation may be exponential in the size of the input query). However, once again, the ABox manipulation that it requires makes it difficult to apply in an OBDA scenario.

The REQUIEM reasoner [24] implements a rewriting algorithm which reduces the number of queries in the final reformulation, still being purely intensional like QUONTO. However, it currently supports none of the QUONTO advanced features, such as identification or EQL constraint management, nor mappings to external databases.

The OWLGres prototype [30], which allows for TBox specification in *DL-Lite*, uses the PostgreSQL DBMS for the storage of the ABox, and provides conjunctive query processing. The algorithm for query answering implemented in OWLGres, however, is not complete with respect to the computation of the certain answers to user queries. More details on the comparison between OWLGres and QUONTO can be found in [10].

None of the above mentioned systems provides a mechanism for connecting an ontology to external independent data sources with powerful map-

[10] `http://www.ontotext.com/owlim/`

[11] `http://www.oracle.com/it/products/database/`

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

[13] Notice that *DL-Lite* is instead at the basis of another profile, namely OWL 2 QL, which enables conjunctive query answering in LOGSPACE (in fact AC^0).

[14] `http://virtuoso.openlinksw.com/`

pings. In fact, Virtuoso, as well as, Ontobroker[15], which is a commercial version of KAON2, provide some form of support to information integration, which allows for accessing multiple ontologies or data sources. Such features however are not characterized in a formal way and cannot be framed in terms of the conceptual architecture at the basis of semantic data integration [22]. A similar observation can be made on commercial products nowadays offered by several major software vendors (Oracle, IBM, Microsoft, etc.): such tools can be seen as a collection of wrappers allowing the users to access a variety of data sources and to see such sources as a single database. However, while suitable for system interoperation, no real semantic integration is carried out. Such systems may have to be considered more as data federation tools rather than semantic data integration tools. From the research point of view, semantic data integration [22] has been studied deeply in the last two decades, producing a number of interesting results. The various approaches can be classified according to the form they adopt for the mapping that connects the global view to the sources. In the *global-as-view (GAV)* approach, in which the entities of the global schema are defined by means of queries over the sources, whereas in the *local-as-view (LAV)* approach source entities are defined by means of queries over the global schema. We notice that currently Mastro adopts a very general form of GAV mapping. Examples of GAV systems are TSIMMIS [14], and Garlic [31]. Information Manifold [23], InfoMaster [15], and Picsel [16] are instead notable examples of LAV systems.

Notice, however, that all the data integration systems mentioned above suffer from some weaknesses from the modeling perspective, mainly due to the limited expressive power of the languages provided to model the global schema of the integration system. In this direction, Mastro aims at overcoming this limitation by providing the best expressive power allowed while preserving tractability of conjunctive query answering and of the integration tasks.

[15] http://www.ontoprise.de/en/home/products/ontobroker/

6. Conclusions

In this paper we presented Mastro, a system for ontology-based data access, which provides a comprehensive solution to such a problem by offering features both for specifying and reasoning on an ontology, and for mapping external data sources to it. Efficient algorithms for advanced forms of query answering are implemented, which enable effective data access. Experiences on real cases yielded very encouraging results, showing the applicability of the Mastro approach to real-world problems.

We plan to extend Mastro in the following directions:

(i) Enriching the ontology representation and reasoning layer with *inconsistency tolerant* capabilities: indeed, when integrating different data sources under the same ontology, it may happen that the reconciled data do not satisfy the ontology. In such cases, repairing the data could be inconvenient, or not possible at all. However, it is possible and important to exploit techniques for consistent query answering [8] in order to make Mastro able to support meaningful query answering even in the presence of inconsistent data. Theoretical results at the basis of the approach we want to implement can be found in [21].

(ii) Allowing for more expressive forms of mappings: LAV mappings could be adopted for those settings in which source data may be incomplete with respect to the ontology used to access the sources underlying the system.

(iii) Implementing "write-also" capabilities: most of the studies carried out in information integration, and the systems proposed to solve the integration problem are mainly oriented towards a read-only approach. This means that the data flows from the sources to the global ontology only. However, several studies have been carried out on the *update* problem [11,7], attempting to reflect over the sources an update expressed in terms of the global ontology. We already carried out some experiments in this direction and plan to extend Mastro in order to support such features.

(iv) Finally, we are currently working to optimize the reformulation step in QuOnto, following the line of research of [26,24], in order to reduce the number of queries produced: this aspect may have a crucial impact on the performance of the whole system.

Acknowledgments. This research has been partially supported by the EU under FP7 project ACSI – Artifact-Centric Service Interoperation (grant n. FP7-257593), and by Regione Lazio under the project "Integrazione semantica di dati e servizi per le aziende in rete".

References

[1] A. Amoroso, G. Esposito, D. Lembo, P. Urbano, and R. Vertucci. Ontology-based data integration with MASTRO-I for configuration and data management at SELEX Sistemi Integrati. In *Proc. of SEBD 2008*, pages 81–92, 2008.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[3] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of KR 2006*, pages 260–270, 2006.

[4] 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.

[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. Path-based identification constraints in description logics. In *Proc. of KR 2008*, pages 231–241, 2008.

[7] D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. Evolution of *DL-Lite* knowledge bases. In *Proc. of ISWC 2010*, volume 6496 of *LNCS*, pages 112–128. Springer, 2010.

[8] J. Chomicki. Consistent query answering: Five easy pieces. In *Proc. of ICDT 2007*, volume 4353 of *LNCS*, pages 1–17. Springer, 2007.

[9] C. Corona, E. Di Pasquale, A. Poggi, M. Ruzzi, and D. F. Savo. When OWL met *DL-Lite* .... In *Proc. of SWAP 2008*, 2008.

[10] C. Corona, M. Ruzzi, and D. F. Savo. Filling the gap between OWL 2 QL and QuOnto: ROWLKit. In *Proc. of DL 2009*, volume 477 of *CEUR,* `ceur-ws.org`, 2009.

[11] G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the update of description logic ontologies at the instance level. In *Proc. of AAAI 2006*, pages 1271–1276, 2006.

[12] G. De Giacomo, M. Lenzerini, and R. Rosati. Towards higher-order *DL-Lite*. In *Proc. of DL 2008*, volume 353 of *CEUR,* `ceur-ws.org`, 2008.

[13] 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.

[14] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *J. of Intelligent Information Systems*, 8(2):117–132, 1997.

[15] M. R. Genereseth, A. M. Keller, and O. M. Duschka. Infomaster: An information integration system. In *Proc. of ACM SIGMOD*, pages 539–542, 1997.

[16] F. Goasdoue, V. Lattes, and M.-C. Rousset. The use of CARIN language and algorithms for information integration: The Picsel system. *Int. J. of Cooperative Information Systems*, 9(4):383–401, 2000.

[17] V. Haarslev, R. Möller, and M. Wessel. Description logic inference technology: Lessions learned in the trenches. In *Proc. of DL 2005*, volume 147 of *CEUR,* `ceur-ws.org`, 2005.

[18] U. Hustadt, B. Motik, and U. Sattler. A decomposition rule for decision procedures by resolution-based calculi. In *Proc. of LPAR 2004*, pages 21–35, 2004.

[19] U. Hustadt, B. Motik, and U. Sattler. Reasoning in description logics by a reduction to Disjunctive Datalog. *J. of Automated Reasoning*, 39(3):351–384, 2007.

[20] 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*, 2010.

[21] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Inconsistency-tolerant semantics for description logics. In *Proc. of RR 2010*, 2010.

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

[23] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogenous information sources using source descriptions. In *Proc. of VLDB'96*, 1996.

[24] H. Pérez-Urbina, B. Motik, and I. Horrocks. A comparison of query rewriting techniques for *DL-lite*. In *Proc. of DL 2009*, volume 477 of *CEUR,* `ceur-ws.org`, 2009.

[25] 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.

[26] R. Rosati and A. Almatelli. Improving query answering over *DL-Lite* ontologies. In *Proc. of KR 2010*, 2010.

[27] D. F. Savo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodríguez-Muro, V. Romagnoli, M. Ruzzi, and G. Stella. MASTRO at work: Experiences on ontology-based data access. In *Proc. of DL 2010*, volume 573 of *CEUR,* `ceur-ws.org`, pages 20–31, 2010.

[28] R. Shearer, B. Motik, and I. Horrocks. HermiT: A highly-efficient OWL reasoner. In *Proc. of OWLED 2008*, volume 432 of *CEUR,* `ceur-ws.org`, 2008.

[29] E. Sirin and B. Parsia. Pellet: An OWL DL reasoner. In *Proc. of DL 2004*, volume 104 of *CEUR,* `ceur-ws.org`, 2004.

[30] M. Stocker and M. Smith. Owlgres: A scalable OWL reasoner. In *Proc. of OWLED 2008*, volume 432 of *CEUR,* `ceur-ws.org`, 2008.

[31] M. Tork Roth, M. Arya, L. M. Haas, M. J. Carey,

W. F. Cody, R. Fagin, P. M. Schwarz, J. T. II, and E. L. Wimmers. The Garlic project. In *Proc. of ACM SIGMOD*, page 557, 1996.

[32] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of IJCAR 2006*, pages 292–297, 2006.