Mastro Studio: a system for Ontology-Based Data Management

Cristina Civili, Marco Console, Domenico Lembo,
Lorenzo Lepore, Riccardo Mancini, Antonella Poggi,
Marco Ruzzi, Valerio Santarelli, and Domenico Fabio Savo

DIAG, Sapienza Università di Roma
lastname@dis.uniroma1.it

1 Introduction

Ontology-based data access (OBDA) is a computing paradigm in which access to data is realized through a three-level architecture, constituted by an ontology, a set of data sources, and the mapping between the two.

In this paper we present the MASTRO STUDIO system for data management based on the OBDA paradigm [5]. MASTRO STUDIO is based on the MASTRO reasoner for OBDA, and, therefore, inherits from MASTRO the characteristics which we discuss in this and the following paragraph. Ontologies in MASTRO are specified in logics of the *DL-Lite* family of Description Logics [4, 5]. Such logics, which are at the base of the OWL 2 QL profile, allow to capture the main modeling features of a variety of representation languages, such as basic ontology languages and conceptual data models, and at the same time maintain computational complexity of reasoning low, in particular when computed with respect to the size of the input data only (i.e., in data complexity). Data sources in MASTRO are seen as a single relational database. When more than one source or even non-relational sources need to be accessed, such a database can be obtained through the use of off-the-shelf relational data federation tools. Finally, the mapping is essentially a set of GAV mapping assertions [7], which associate ontology elements with queries specified on the underlying database.

By virtue of these design choices, query answering in MASTRO can be done through a very efficient technique that reduces this task, via query rewriting, to standard SQL query evaluation.

Besides reasoning capabilities offered by MASTRO, MASTRO STUDIO is also equipped with a web-based graphical user interface (GUI) which allows for advanced mechanisms for the inspection of the components of an OBDA specification, i.e., the ontology, the mapping and the data sources. In particular, it allows for the representation of the ontology in a graphical form, resembling Entity-Relationship modeling, which makes the ontology accessible to non-experts of logical and ontology formalisms. Also, MASTRO STUDIO provides wiki-like documentation in which every element of the ontology is associated with a natural language description, as well as with all ontology axioms and mapping assertions in which it is involved. The MASTRO STUDIO GUI is realized through the Drupal[1] content management system.

[1] `http://drupal.org`

In the last few years, several works have been conducted on OBDA in a simplified setting where no mappings are used to connect the (intentional level of the) ontology to external data sources [8, 12]. The only notable exception besides MASTRO STUDIO is Quest [10], which has indeed common roots with our system. Quest is a system for query answering over *DL-Lite*$_A$ ontologies, which can work in both "classical" (i.e., with a local ABox) and "virtual" mode (i.e., exploiting mappings). Although first experiments show effectiveness of Quest in the classical scenario [10], the development of its usage in the virtual mode is still ongoing. Finally, we observe that, to the best of our knowledge, MASTRO STUDIO is the only full-fledged ontology-based data management system which provides, along with OBDA functionalities, advanced features for documenting and inspecting an OBDA specification.

2 Technical background

We recall here the notions of OBDA specification and OBDA semantics, and survey the main reasoning services and optimizations offered by MASTRO STUDIO. These (optimized) reasoning services are in fact inherited from the MASTRO reasoner, in which they are realized, and suitably exposed as web services by MASTRO STUDIO. For these reasons, in the rest of this section we refer directly to the Mastro reasoner.

OBDA specification. In MASTRO, an OBDA specification is a triple $\langle \mathcal{O}, \mathcal{M}, \mathcal{D} \rangle$, where $\mathcal{O}$ is an ontology, $\mathcal{D}$ is a relational database instance, and $\mathcal{M}$ is the mapping between $\mathcal{O}$ and $\mathcal{D}$. More precisely, $\mathcal{O}$ is specified in a logic of the *DL-Lite* family of lightweight Description Logics (DLs) [4, 5]. DLs are decidable fragments of first-order logic (FOL) that allow 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. DLs of the *DL-Lite* family have been specifically designed for OBDA and allow for a good tradeoff between the expressive power of the language and the computational complexity of reasoning. Notably, query answering in such DLs can be done in LOGSPACE with respect to data complexity. *DL-Lite* logics essentially capture standard conceptual modeling formalisms, such as UML Class Diagrams and Entity-Relationship Schemas, and are at the basis of OWL 2 QL, one of the tractable profiles of OWL 2, the current W3C standard language for ontologies[2]. $\mathcal{M}$ is a set of assertions of the form $\Phi \rightsquigarrow \psi$, where Φ is an SQL query specified over the schema of $\mathcal{D}$, and ψ is an element of the ontology $\mathcal{O}$, i.e., a concept, a role, or an attribute (see also [9]). Intuitively, such a mapping assertion specifies that the tuples returned by the query Φ are used to generate the facts that instantiate ψ. $\mathcal{M}$ is therefore a GAV mapping, according to the data integration terminology [7].

OBDA semantics. The semantics of an OBDA specification is given in terms of FOL interpretations. A FOL interpretation $\mathcal{I}$ is a model for an ontology $\mathcal{O}$ if it satisfies (in the classical FOL sense) all logical axioms specified in $\mathcal{O}$ [4]. Then, given an OBDA specification $\mathcal{B} = \langle \mathcal{O}, \mathcal{M}, \mathcal{D} \rangle$, a FOL interpretation $\mathcal{I}$ is a *model for* $\mathcal{B}$ if (i) $\mathcal{I}$ is a model for $\mathcal{O}$, and (ii) $\mathcal{I}$ satisfies $\mathcal{M}$, i.e., for each mapping assertion $\Phi \rightsquigarrow \psi$ and each tuple $\boldsymbol{t}$ in the evaluation of Φ over $\mathcal{D}$, $\mathcal{I}$ satisfies the fact $\psi(\boldsymbol{t})$ (see also [9]). Notice that the above notion of mapping satisfaction corresponds to the classical notion of satisfaction

[2] `http://www.w3.org/TR/owl-profiles/`

of *sound* GAV mapping in data integration [7]. An OBDA $\mathcal{B}$ is *satisfiable* if $\mathcal{B}$ admits at least one model.

Reasoning in Mastro. Reasoning services that do not consider data are called *intensional*. Among these services, MASTRO STUDIO allows for the computation of all subsumption relationships inferred in an ontology between concepts, roles, and attributes. This, in particular, enables the construction of the classification tree of the ontology [2].

The main task involving data performed by MASTRO is to answer (unions of) conjunctive queries ((U)CQs) posed over the ontology $\mathcal{O}$ of an OBDA system $\mathcal{B} = \langle \mathcal{O}, \mathcal{M}, \mathcal{D} \rangle$. Answering one such query Q amounts to computing its *certain answers*, denoted *CertAns*$(Q, \mathcal{B})$, i.e., the tuples that are in the interpretation of Q in every model of $\mathcal{B}$ (the FOL interpretation of a UCQ is the standard one [1])[3].

In MASTRO, certain answers to queries are computed through a query rewriting process. The basic notion underlying this approach is the one of perfect rewriting: a query Q_{DB} over $\mathcal{D}$ is a *perfect rewriting* of a query Q under $\mathcal{B}$ if the evaluation of Q_{DB} over $\mathcal{D}$ returns the set *CertAns*$(Q, \mathcal{B})$. The perfect rewriting of a UCQ Q posed over $\mathcal{O}$ can be obtained in two steps: (i) compute an *ontology-rewriting* Q' of Q with respect to the ontology $\mathcal{O}$; (ii) compute the *mapping-rewriting* of Q' by using the mapping $\mathcal{M}$, thus obtaining an SQL query on $\mathcal{D}$. Intuitively, an ontology-rewriting of Q is another query Q', expressed over $\mathcal{O}$, 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 $\mathcal{O}$, i.e., *CertAns*$(Q, \langle \mathcal{O}, \mathcal{M}, \mathcal{D} \rangle) =$ *CertAns*$(Q', \langle \emptyset, \mathcal{M}, \mathcal{D} \rangle)$. This step is realized in MASTRO through the algorithm Presto [11]), which rewrites Q into a new UCQ Q' over $\mathcal{O}$. Then, the mapping-rewriting step can be seen as a variant of the unfolding procedure in GAV data integration, as it essentially substitutes each atom in the query Q' with the SQL query that the mapping associates to the atom predicate. After the rewriting process, the query is fully expressed in SQL and can be directly evaluated over the sources.

We notice also that checking ontology satisfiability in *DL-Lite* can be reduced to query answering. In particular, to each ontology axiom we can associate a query aiming at identifying the existence of counterexamples, i.e., data violating such axiom (e.g, data contradicting axioms imposing disjointness of concepts or functionality of roles). This is indeed the way ontology satisfiability is realized in MASTRO.

Optimizations. The perfect rewriting produced as described above is a union of SQL queries which may often contain a huge number of disjuncts. This is mainly due to the mapping-rewriting step, which combines in all possible ways the various mapping queries associated to each atom predicate, 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 [6]. However, in general, not all such disjuncts really contribute to the computation of the certain answers (for example, because a disjunct is contained into another). We developed in the MASTRO reasoner a mechanism that is able to prune the rewriting and produce another perfect rewriting of smaller size. The adoption of this technique by MASTRO allows to reduce the evaluation time of the final rewriting.

Also, to further optimize the rewriting process, MASTRO allows for the use of so-called *perfect mapping assertions*. Given an OBDA specification $\mathcal{B}$, a perfect mapping

[3] In fact, MASTRO even allows for processing more expressive queries interpreted under a semantics that approximate standard FOL semantics (see [3] for details).

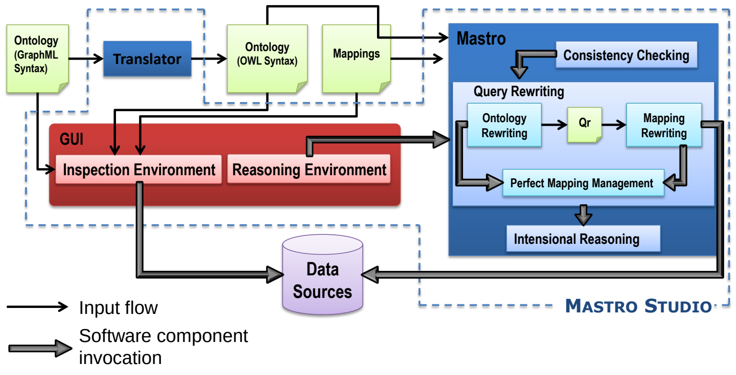

Fig. 1. The MASTRO STUDIO system architecture

assertion is a pair $\langle cq, cq_{DB}\rangle$ such that cq is a conjunctive query and cq_{DB} is a perfect rewriting of q under $\mathcal{B}$. Perfect mapping assertions of the above form can be used during both the ontology-rewriting and mapping-rewriting steps in the following way: if a conjunctive query q to be rewritten contains a subquery cq, MASTRO substitutes cq with cq_{DB} (modulo some variable unification), and makes the rewriting process to continue only on the remaining part of q. It can be shown that this is a drastic optimization allowing to heavily reduce the size of the perfect rewritings.

Notice also that perfect mappings may be obtained by simply storing the perfect rewritings computed by MASTRO itself. In other words, the set of perfect mappings can be considered as a *memory* of the previous perfect rewritings, suitably pruned according to the first optimization described above. For further details, see [6].

3 The MASTRO STUDIO system

The base principle adopted in the design of MASTRO STUDIO is to provide a seamless access to the ontology description and the reasoning services over it. The MASTRO STUDIO GUI is web-based and is realized through the Drupal open source CMS (Content Management System). Via the GUI, the user can access two different environments: the *Inspection Environment* and the *Reasoning Environment*. The first environment provides the user with functionalities for easily inspecting all the OBDA system components, whereas the second one allows for invoking various reasoning services, and is therefore tightly coupled with the underlying reasoner (cf. Figure 1).

GUI inspection environment. This environment allows for three main functionalities, each realized by a specific component: ontology inspection, mapping inspection and data source inspection. *Ontology inspection* enables in-depth ontology navigation by means of the visualization of both a graphical representation of the ontology and its specification through the OWL functional syntax[4], as well as the provision of hypertextual descriptions of ontology elements, organized in the form of a wiki.

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

The graphical representation of the ontology provided by MASTRO STUDIO has a graph-like structure, similar to that of an Entity-Relationship diagram. It allows for a gentle inspection of the ontology, accessible also to non-experts of logical and ontology formalisms. The ontology graph is encoded into GraphML[5], a standard XML-based graph exchange format. Such encoding, besides being used as input to the inspection environment for visualization of the ontology diagram, is transformed into OWL functional syntax through the *Translator* module (cf. Figure 1), which generates the corresponding *DL-Lite* axioms specified through the standard OWL functional syntax.

The ontology inspection component also contains wiki-like documentation of the ontology, provided through the use of contributed Drupal modules such as Wikitools, Freelinking, and Flexifilter[6]. Moreover, a custom module has been added to the Drupal core in order to automatically generate the wiki pages associated to the ontology. Starting from an ontology, the module allows to create a wiki page for each concept, role and attribute, according to a predefined template that includes some asserted information, as well as axioms and mappings that are related to the element documented in the page. These pages are stored through the CMS and can be manually edited by the user in order to enrich the documentation with human-friendly information such as textual descriptions. The documentation can be inspected through a tree-menu that represents the hierarchies of concepts, roles and attributes as asserted in the ontology. The *Mapping inspection* and the *Data Source inspection* components provide the ability to inspect respectively the mapping assertions and the data sources. In particular the latter allows the user to visualize the structure of the source relations and also to pose direct SQL queries over it.

GUI reasoning environment. The second environment is structured on the basis of the main reasoning services provided by MASTRO STUDIO. In particular, it enables for invoking intensional reasoning, ontology satisfiability, and query answering services. As for intensional reasoning, the user is provided with the visualization of all subsumptions inferred by the ontology, relying on the underlying intensional reasoning module (cf. Figure 1). Concerning ontology satisfiability, the user can get an indication of which axioms are contradicted by source data, and a preview of such data (counterexamples). Furthermore, the environment allows to specify queries (in SPARQL syntax) over the ontology and to visualize their certain answers returned by the reasoner. On user demand, details of the rewriting process, such as the ontology-rewriting and the mapping-rewriting, can be shown.

MASTRO reasoner. It is constituted by three main modules, i.e., the query rewriting, the consistency checking, and the intensional reasoning module.

The *query rewriting* module realizes the query rewriting process and its optimizations described in Section 2. In particular, the *ontology rewriting* sub-module receives the user query Q from the GUI and the OWL syntax specification of the ontology as inputs and produces Q_r, which is the ontology-rewriting of Q. Q_r is then passed to the *mapping rewriting* sub-module (cf. Figure 1), which takes as input also the mapping specification and computes the perfect rewriting Q'_{SQL} of Q. The module is also in charge of pruning Q'_{SQL} according to the first optimization described in Section 2.

[5] `http://graphml.graphdrawing.org/`

[6] `http://drupal.org/project/`

The resulting query Q''_{SQL} is then sent to the underlying DBMS for evaluation. Furthermore, it is also passed to the *perfect mapping manager* sub-module, which stores (subject to user confirmation) the perfect mapping $\langle Q, Q''_{SQL} \rangle$. Such module feeds both the ontology and the mapping rewriting modules, that can make use of perfect mapping assertions to "freeze" portions of the query to be rewritten (cf. Section 2).

The *intensional reasoning* module realizes the intensional reasoning tasks described in the previous section. Besides providing its result to the reasoning environment of the GUI, it also gives the computed subsumptions as input to the query rewriting module since such subsumptions are needed for the execution of the Presto algorithm [11].

Finally, the *consistency checking* module realizes the ontology satisfiability method sketched in the previous section, verifying the consistency of each ontology axiom by producing the associated query and sending it to the query rewriting module for reformulation and evaluation. The results are returned to the GUI reasoning environment.

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).

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley Publ. Co., 1995.
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, 2nd edition, 2007.
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. G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Mastro: A reasoner for effective ontology-based data access. In *Proc. of ORE-2012*, volume 858 of *CEUR,* `ceur-ws.org`, 2012.
6. F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proc. of EDBT 2013*, 2013.
7. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246, 2002.
8. 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.
9. 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.
10. M. Rodriguez-Muro and D. Calvanese. High performance query answering over *DL-Lite* ontologies. In *Proc. of KR 2012*, pages 308–318, 2012.
11. R. Rosati and A. Almatelli. Improving query answering over *DL-Lite* ontologies. In *Proc. of KR 2010*, pages 290–300, 2010.
12. T. Venetis, G. Stoilos, and G. B. Stamou. Incremental query rewriting for OWL 2 QL. In *Proc. of DL 2012*, 2012.