The *Ontop* Framework for Ontology Based Data Access

Timea Bagosi[1], Diego Calvanese[1], Josef Hardi[2], Sarah Komla-Ebri[1], Davide Lanti[1], Martin Rezk[1], Mariano Rodríguez-Muro[3], Mindaugas Slusnys[1], and Guohui Xiao[1]

[1]Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
[2] Obidea Technology, Indonesia
[3] IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

1 Ontology Based Data Access

Ontology Based Data Access (OBDA) [4] is a paradigm of accessing data trough a conceptual layer. Usually, the conceptual layer is expressed in the form of an RDF(S) [10] or OWL [15] ontology, and the data is stored in relational databases. The terms in the conceptual layer are mapped to the data layer using mappings which associate to each element of the conceptual layer a (possibly complex SQL) query over the data sources. The mappings have been formalized in the recent R2RML W3C standard [6]. This virtual graph can then be queried using an RDF query language such as SPARQL [7].

Formally, an OBDA system is a triple $\mathcal{O} = \langle \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$, where:

- $\mathcal{T}$ is the intensional level of an ontology. We consider ontologies formalized in description logics (DLs), hence $\mathcal{T}$ is a DL TBox.
- $\mathcal{S}$ is a relational database representing the sources.
- $\mathcal{M}$ is a set of mapping assertions, each one of the form

$$\Phi(\boldsymbol{x}) \leftarrow \Psi(\boldsymbol{x})$$

  where
  - $\Phi(\boldsymbol{x})$ is a query over $\mathcal{S}$, returning tuples of values for $\boldsymbol{x}$
  - $\Psi(\boldsymbol{x})$ is a query over $\mathcal{T}$ whose free variables are from $\boldsymbol{x}$.

The main functionality of OBDA systems is query answering. A schematic description of the query transformation process (usually SPARQL to SQL) performed by a typical OBDA system is provided in Figure 1. In such an architecture, queries posed over a conceptual layer are translated into a query language that can be handled by the data layer. The translation is independent of the actual data in the data layer. In this way, the actual query evaluation can be delegated to the system managing the data sources.

2 The *Ontop* Framework

Ontop is an open-source OBDA framework released under the Apache license, developed at the Free University of Bozen-Bolzano[1] and currently acts as the query transformation module of the EU project Optique[2].

[1] `http://ontop.inf.unibz.it`
[2] `http://www.optique-project.eu`

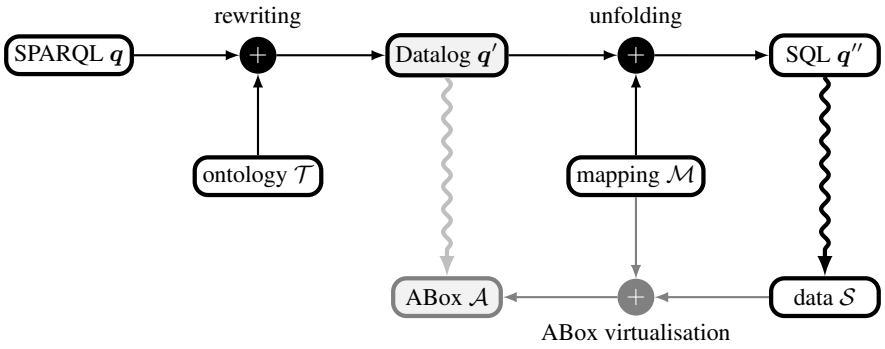

Fig. 1. Query processing in an OBDA system

As an OBDA system, to the best of our knowledge, *Ontop* is the first to support all the following W3C recommendations: OWL, R2RML, SPARQL, SWRL and SPARQL OWL 2 QL regime. In addition, all the major commercial and free databases are supported. For each component of the OBDA system, *Ontop* supports the widely used standards:

Mapping *Ontop* supports two mapping languages: (1) the native *Ontop* mapping language which is easy to learn and use and (2) the RDB2RDF Mapping Language (R2RML) which is a W3C recommendation.

Ontology *Ontop* fully supports OWL 2 QL ontology language [11], which is a superset of RDFS. OWL 2 QL is based on the DL-Lite family of description logics [5], which are lightweight ontologies and guarantee queries over the ontology can be rewritten to equivalent queries over the data source. Recently *Ontop* is also extended to support the linear recursive fragment of SWRL (Semantic Web Rule Language) [8,16].

Data Source *Ontop* supports all the databases which implement SQL 99. These include all major relational database systems, e.g., PostgreSQL, MySQL, H2, DB2, ORACLE, and MS SQL Server.

Query *Ontop* essentially supports all the features of SPARQL 1.0 and SPARQL OWL QL Regime of SPARQL 1.1 [9]. Supporting of other features in SPARQL 1.1 (e.g., aggregates, property path queries, negations) is ongoing work.

The core of the *Ontop* is the SPARQL engine Quest which supports RDFS and OWL 2 QL entailment regimes by rewriting the SPARQL queries (over the virtual RDF graph) to SQL queries (over the relational database). *Ontop* is able to generate efficient (and highly optimized [13,14]) SQL queries, that in some cases are very close to the SQL queries that would be written by a database expert.

The *Ontop* framework can be used as:

- a *plugin for Protege 4* which provides a graphical interface for mapping editing and SPARQL query execution,
- a *Java library* which implements both OWL API and Sesame API interfaces, available as maven dependencies, and
- a *SPARQL end-point* through Sesame's Workbench.

3 A Demo of the Movie Scenario

In this section, we describe a complete demo of *Ontop* using the movie scenario [12]. The datasets and systems are available online[3].

3.1 Movie Scenario Dataset

the Movie Ontology The movie ontology *MO* aims to provide a controlled vocabulary to semantically describe movie related concepts (e.g., Movie, Genre, Director, Actor) and the corresponding individuals ("Ice Age", "Drama", "Steven Spielberg" or "Johnny Depp") [3]. The ontology contains concept hierarchies for movie categorization that enables user-friendly presentation of movie descriptions in the appropriate detail. There are several additions to the ontology terminology due to the requirements in the demo, e.g., concepts `TVSeries` and `Actress`.

IMDb data IMDB's data is provided as text files [4] which need to be converted into an SQL file using a third party tool. Our IMDB raw data was downloaded in 2010 and the SQL script was generated using IMDbPY[5]. IMDbPY generates an SQL schema (tables) appropriate for storing IMDB data and then reads the IMDB plain text data files to generate the SQL INSERT commands that populate the tables. It can generate PostgreSQL, MySQL and DB2 SQL scripts. In this demo we use a PostgreSQL compatible script and database takes up around 6GB on the disk.

Mappings The mappings for this scenario are natural mappings that associate the data in the SQL database to the movie ontology's vocabulary. They are "natural" mapping, in the sense that the only purpose of the mappings was to be able to query the data through the ontology. There was no intention to highlight the benefits of any algorithm or technique used in *Ontop*. The first version of the mappings for this scenario were developed by students of Free University of Bolzano as part of an lab assignment. The current mappings are the improved version of those create by our development team.

Queries We included around 40 queries which are in the file `movieontology.q` and can be used to explore the data set. The queries have different complexities, going from very simple to fairly complex. Note that some form of inference (beyond simple query evaluation) is involved in most of these queries, in particular, hierarchies are often involved.

3.2 Using Protege Plugin

We demonstrate how to use *Ontop* as a protege plugin. The steps are:

[3] `https://github.com/ontop/ontop/wiki/Example_MovieOntology`
[4] `http://www.imdb.com/interfaces`
[5] `http://imdbpy.sourceforge.net`

(1) Start PostgreSQL with IMDb data.
(2) Start Protege with ontop plugin from command line.
(3) Open the OWL file `movieontology.owl` from Protege. The *Ontop* plugin will also automatically open the mapping file `movieontology.obda` and query file `movieontology.q`.
(4) Check the ontology and mappings. Two screen shots of the ontology and mappings are shown in Figure 2 and 3.
(5) Start the Quest reasoner from the menu.
(6) Run sample queries and check the generated SQLs. For example, we can execute the query "Find names that act as both the director and the actor at the same time produced in Eastern Asia" as shown in Figure 4.

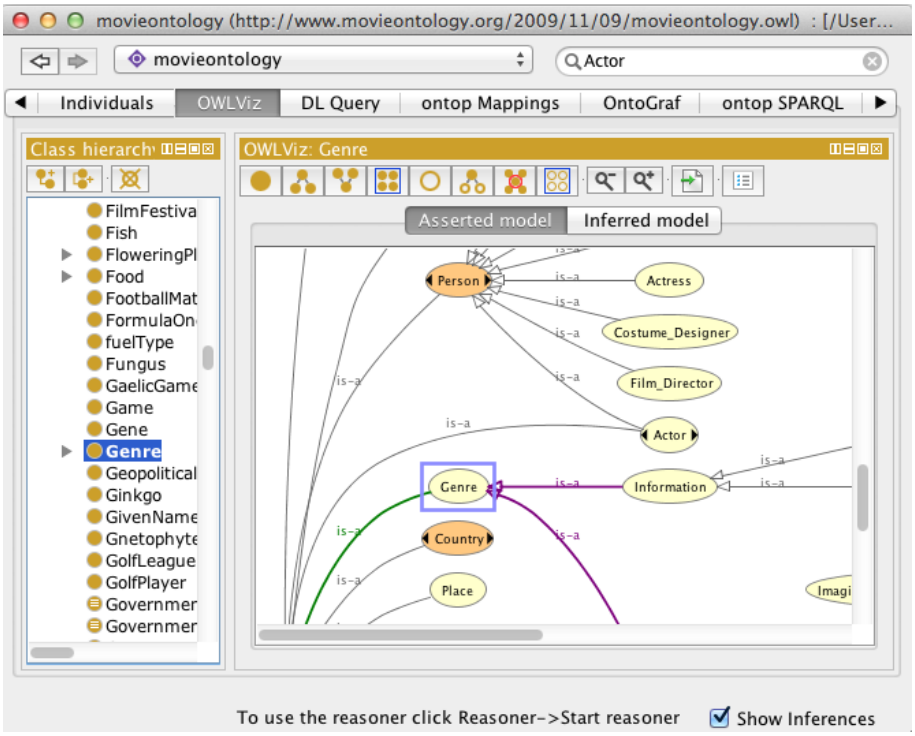

Fig. 2. Movie ontology

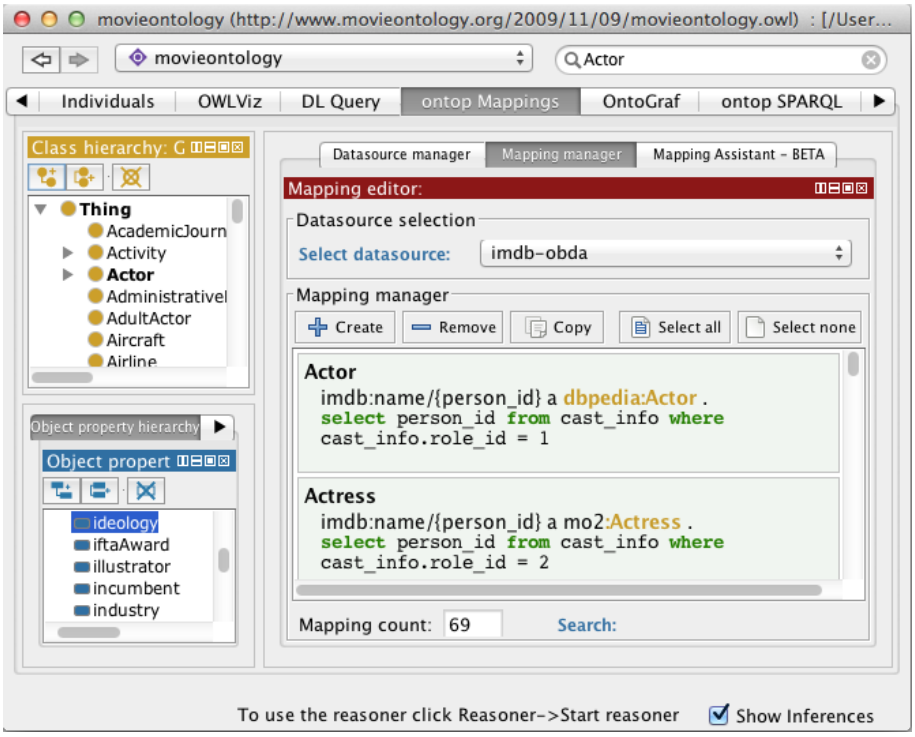

Fig. 3. Movie mappings

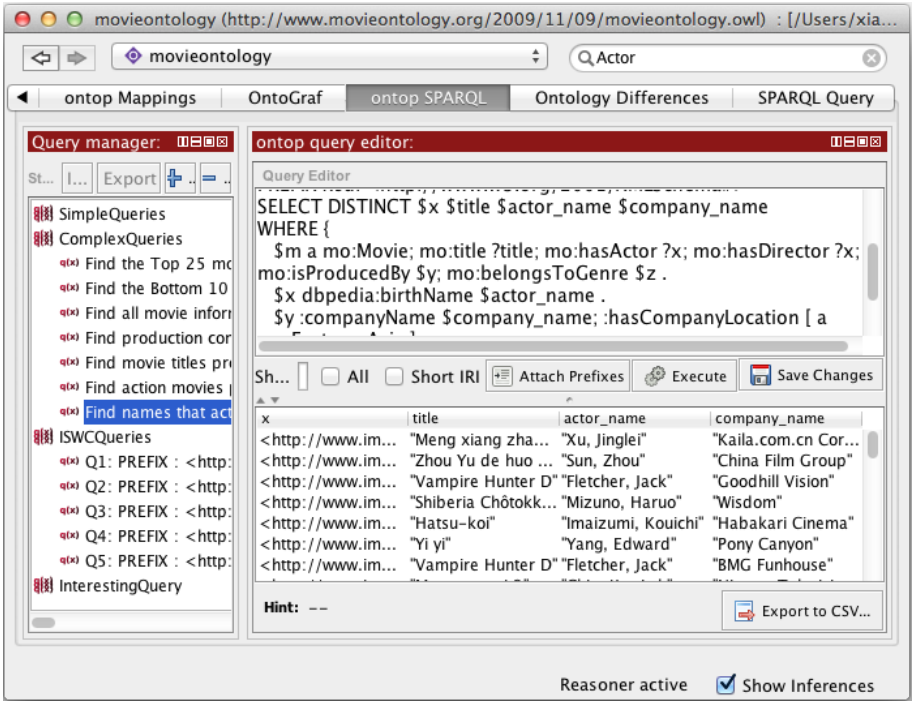

Fig. 4. Example query

3.3 Using Java API

We show how the movie scenario can be implemented using the *Ontop* java libraries through OWL API and sesame API. The complete code for the demo is available online[6].

Using OWL API The OWL API is a Java API and reference implementation for creating, manipulating and serializing OWL Ontologies [2]. In the first example we use OWL API to execute all the 40 SPARQL queries over the movie ontology, using the mapping in our obda format and a PostgreSQL database with the IMDb data.

Ontop uses Maven to manage the dependencies. Since the release of version 1.10, *Ontop* itself has been deployed to the central maven repository. All artifacts have the same groupId `it.unibz.inf.ontop`. In this example we use the OWL API interface of *Ontop*, so we put the following in the `pom.xml`:

```
<dependency>
    <groupId>it.unibz.inf.ontop</groupId>
    <artifactId>ontop-quest-owlapi3</artifactId>
    <version>1.12.0</version>
</dependency>
```

Moreover we need the dependency for PostgreSQL JDBC driver as shown below.

```
<dependency>
     <groupId>postgresql</groupId>
     <artifactId>postgresql</artifactId>
     <version>9.0-801.jdbc4</version>
</dependency>
```

The files needed to start the *Ontop* reasoner are the ontology file `movieontology.owl` and the obda file `movieontology.obda`. The obda file contains both mappings and database settings. This allows to access the data in the PostgreSQL database using the mappings in the OBDA model. First we load the OWL file and OBDA file:

```
// Loading the OWL file using OWL API
OWLOntologyManager manager;
manager = OWLManager.createOWLOntologyManager();
OWLOntology ontology;
ontology = manager.loadOntologyFromOntologyDocument
       ((new File(owlFile)));

// Loading the OBDA file
OBDAModel obdaModel = fac.getOBDAModel();
ModelIOManager ioManager = new ModelIOManager(obdaModel);
ioManager.load(obdaFile);
```

Next we create a new instance of the reasoner (QuestOWL reasoner), adding the necessary preferences to prepare its configuration. We prepare the `QuestOWLConnection` for querying.

[6] `https://github.com/ontop/ontop-examples`

```
QuestOWLFactory factory = new QuestOWLFactory();
factory.setOBDAController(obdaModel);

//Setting preferences putting Quest in virtual mode.
factory.setPreferenceHolder(p);
QuestPreferences preference = new QuestPreferences();
preference.setCurrentValueOf
    (QuestPreferences.ABOX_MODE, QuestConstants.VIRTUAL);

// Creating a new instance of the reasoner
QuestOWL reasoner;
reasoner = (QuestOWL) factory.createReasoner
         (ontology, new SimpleConfiguration());

// Now we are ready for querying
QuestOWLConnection conn = reasoner.getConnection();
QuestOWLStatement st = conn.createStatement();
```

Ontop supports a file format of multiple SPARQL queries. Here we execute each query using the file `movieontology.q` of 40 queries. Within the instance each SPARQL query is translated in an SQL query, which allows to retrieve the results from the PostgreSQL database. For simplicity, we only display to the user the number of results of the query and the time required for the execution.

```
// Loading the query file
QueryController qc = new QueryController();
QueryIOManager qman = new QueryIOManager(qc);
qman.load("src/main/resources/example/movie/movieontology.q");

// Execute each query
for (QueryControllerGroup group : qc.getGroups()) {
    for (QueryControllerQuery query : group.getQueries()) {

        System.out.println("Executing␣query:␣" + query.getID());
        System.out.println("Query:␣\n" + query.getQuery());

        long start = System.nanoTime();
        QuestOWLResultSet res = st.executeTuple(query.getQuery());
        long end = System.nanoTime();
        double time = (end - start) / 1000;
        int count = 0;
        while (res.nextRow()) {
            count += 1;
        }

        System.out.println("Total␣result:␣" + count);
        System.out.println("Elapsed␣time:␣" + time + "␣ms");
   }
}
```

At the end of the execution we close all connections and we dispose of the reasoner.

```
//Close connection and resources
if (st != null && !st.isClosed()) {
     st.close();
}
if (!conn.isClosed()) {
    conn.close();
}
reasoner.dispose();
```

Using Sesame API OpenRDF Sesame is a de-facto standard framework for processing RDF data and includes parsers, storage solutions (RDF databases a.ka. triplestores), reasoning and querying, using the SPARQL query language [1].

In the second example we show how to create a repository and execute a single query using Sesame API. First we need to add the Sesame API module of *Ontop* as a dependency to the pom file `pom.xml`.

```
<dependency>
    <groupId>it.unibz.inf.ontop</groupId>
    <artifactId>ontop-quest-sesame</artifactId>
    <version>1.12.0</version>
</dependency>
```

Then we set up the repository and create a connection. The repositories must always be initialized first. We get the repository connection that will be used to execute the query.

```
// Creating and initializing the repository
boolean existential = false;
String rewriting = "TreeWitness";
SesameVirtualRepo repo = new SesameVirtualRepo
       ("test_repo", owlFile, obdaFile, existential, rewriting);
repo.initialize();

RepositoryConnection conn = repo.getConnection();
```

We load the SPARQL file `q1Movie.rq` which contains the same query that we used for the Protege example.

```
//Loading the SPARQL file
String queryString = "";

BufferedReader br = new BufferedReader(new FileReader(sparqlFile));
String line;
while ((line = br.readLine()) != null) {
    queryString += line + "\n";
}
```

```
System.out.println();
System.out.println("The␣input␣SPARQL␣query:");
System.out.println("======================");
System.out.println(queryString);
System.out.println();
```

Now we are ready to execute the query using the created Sesame repository connection and output the results of the SPARQL from the database.

```
// Executing the query
Query query = conn.prepareQuery(QueryLanguage.SPARQL, queryString);

TupleQuery tq = (TupleQuery) query;

TupleQueryResult result = tq.evaluate();

while (result.hasNext()) {
    for (Binding binding : result.next()) {
            System.out.print(binding.getValue() + ",␣");
    }
    System.out.println();
}
```

Finally we close all the connections and release the resources.

```
//Close result set to release resources
result.close();

// Finally close connection to release resources
System.out.println("Closing..,");
conn.close();
```

Acknowledgement. This paper is supported by the EU under the large-scale integrating project (IP) Optique (Scalable End-user Access to Big Data), grant agreement n. FP7-318338.

References

1. OpenRDF Sesame, http://www.openrdf.org/, accessed: 2014-08-27
2. OWL API, http://owlapi.sourceforge.net/, accessed: 2014-08-27
3. Bouza, A.: MO – the movie ontology (2010), http://www.movieontology.org, [Online; 26. Jan. 2010]
4. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: 5th Int. Reasoning Web Summer School Tutorial Lectures (RW 2009), vol. 5689, pp. 255–356. Springer (2009)
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
6. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium (Sep 2012), available at http://www.w3.org/TR/r2rml/
7. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Recommendation, World Wide Web Consortium (Mar 2013), available at http://www.w3.org/TR/sparql11-query
8. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, World Wide Web Consortium (2004)
9. Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G., Zakharyaschev, M.: Answering SPARQL queries over databases under OWL 2 QL entailment regime. In: Proc. of International Semantic Web Conference (ISWC 2014). Lecture Notes in Computer Science, Springer (2014)
10. Manola, F., Mille, E.: RDF primer. W3C Recommendation, World Wide Web Consortium (Feb 2004), available at http://www.w3.org/TR/rdf-primer-20040210/
11. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 web ontology language: Profiles. W3C Recommendation, World Wide Web Consortium, http://www.w3.org/TR/owl2-profiles/ (2012)
12. Rodriguez-Muro, M., Hardi, J., Calvanese, D.: Quest: Effcient SPARQL-to-SQL for RDF and OWL. In: Glimm, B., Huynh, D. (eds.) International Semantic Web Conference (Posters & Demos). CEUR Workshop Proceedings, vol. 914. CEUR-WS.org (2012)
13. Rodriguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: Ontop of databases. In: Proc. of the 12th Int. Semantic Web Conf. (ISWC 2013). vol. 8218, pp. 558–573. Springer (2013)
14. Rodriguez-Muro, M., Rezk, M., Hardi, J., Slusnys, M., Bagosi, T., Calvanese, D.: Evaluating SPARQL-to-SQL translation in Ontop. In: Proc. of the 2nd Int. Workshop on OWL Reasoner Evaluation (ORE 2013). CEUR Workshop Proceedings, vol. 1015, pp. 94–100 (2013)
15. W3C OWL Working Group: OWL 2 web ontology language document overview (second edition). W3C Recommendation, World Wide Web Consortium (2012), available at http://www.w3.org/TR/owl2-overview/
16. Xiao, G., Rezk, M., Rodriguez-Muro, M., Calvanese, D.: Rules and ontology based data access. In: Mugnier, M.L., Kontchakov, R. (eds.) Proc. 8th International Conference on Web Reasoning and Rule Systems (RR 2014). Lecture Notes in Computer Science, Springer (2014)