Undefined 0 (0) 1
IOS Press

Ontop: Answering SPARQL Queries over Relational Databases

Diego Calvanese [a], Benjamin Cogrel [a], Sarah Komla-Ebri [a], Roman Kontchakov [b], Davide Lanti [a], Martin Rezk [a], Mariano Rodriguez-Muro [c], and Guohui Xiao [a]

[a] *Free University of Bozen-Bolzano*
{calvanese,bcogrel, sakomlaebri,dlanti,mrezk,xiao}@inf.unibz.it
[b] *Birkbeck, University of London*
roman@dcs.bbk.ac.uk
[c] *IBM TJ Watson*
mrodrig@us.ibm.com

Abstract. In this paper we present *Ontop*, an open-source Ontology Based Data Access (OBDA) system that allows for querying relational data sources through a conceptual representation of the domain of interest, provided in terms of an ontology, to which the data sources are mapped. Key features of *Ontop* are its solid theoretical foundations, a virtual approach to OBDA that avoids materializing triples and that is implemented through query rewriting techniques, extensive optimizations exploiting all elements of the OBDA architecture, its compliance to all relevant W3C recommendations (including SPARQL queries, R2RML mappings, and OWL 2 QL and RDFS ontologies), and its support for all major relational databases.

Keywords: Ontop, OBDA, Databases, RDF, SPARQL, Ontologies, R2RML, OWL

1. Introduction

Over the past 20 years we have moved from a world where most companies had one all-knowing self-contained central database to a world where companies buy and sell their data, interact with several data sources, and analyze patterns and statistics coming from all of them. The challenge is shifting from obtaining information to finding the *right* information. It has always been the case that information is power but today *attention* rather than information becomes the scarce resource, and those who can distinguish valuable information from background clutter gain power [16]. To separate the wheat from the chaff, the companies need a comprehensive understanding of their data and the ability to cope with diversity in the data.

Since the mid 2000s, *Ontology-Based Data Access* (OBDA) has become a popular approach to tackling this problem [27]. In OBDA, a conceptual layer is given in the form of an ontology that defines a shared vocabulary, models the domain, hides the structure of the data sources, and can enrich incomplete data with background knowledge. Then, queries are posed over this high-level conceptual view, and the users no longer need an understanding of the data sources, the relation between them, or the encoding of the data. Queries are translated by the OBDA system into queries over potentially very large (usually relational and federated) data sources. The ontology is connected to the data sources through a declarative specification given in terms of mappings that relate symbols in the ontology (classes and properties) to (SQL) views over data. The W3C standard R2RML [12] was created with the goal of providing a language for the specification of mappings in the OBDA setting. The ontology together with the mappings exposes a virtual RDF graph, which can be queried using SPARQL, the standard query language in the Semantic Web community. These virtual RDF graphs can be *materialized*, generating RDF triples that can be used with RDF triplestores, or alternatively they can be kept *virtual* and queried only

during query execution. The virtual approach avoids the cost of materialization and can profit from more than 30 years' maturity of relational systems (efficient query answering, security, robust transaction support, etc.).

To illustrate these concepts and the different notions in this paper, we will use the following running example. All the material needed to run this example in the OBDA system *Ontop* (and a complementary tutorial) can be found online[1].

Example 1.1 (Hospital Database)**.** We consider a hospital database with a single table `tbl_patient` that contains information about lung cancer patients. The table has 4 attributes: the patient identifier (`pid`), his/her name, the type of cancer (tumor) and its stage. The lung cancer can be of two types: Non-Small Cell Lung Carcinoma (NSCLC) and Small Cell Lung Carcinoma (SCLC), which are encoded in the table by a boolean value `type` as follows:

- `false` for NSCLC and `true` for SCLC.

The stage of the cancer is encoded by a positive integer value `stage` as follows:

- `1–6` for NSCLC stages I, II, III, IIIa, IIIb and IV,
- `7–8` for SCLC stages Limited and Extensive.

Finally, our sample table contains the following data:

pid	name	type	stage
1	'Mary'	false	4
2	'John'	true	7

Suppose we need a simple piece of information from this database: "Give me the names of patients with a tumor of stage IIIa". Even this simple query in this tiny database already presents some challenges, since to create the query and to understand and analyze the results we need to know how the information is encoded in the data. In the following sections we describe how to use the *Ontop* system to address this challenge by enhancing the database with a semantic layer. □

In this paper we present the OBDA system *Ontop*[2], a mature open-source system, which is currently being used in a number of projects. *Ontop* supports all the W3C recommendations related to OBDA: OWL 2 QL, R2RML, SPARQL, SWRL, and the OWL 2 QL entailment regime in SPARQL. The system is available as a Protégé 4 plugin, a SPARQL endpoint through Sesame Workbench, and a Java library supporting OWL API and Sesame API.

The structure of the paper is as follows. Section 2 presents a high-level overview of the architecture of *Ontop*. Section 3 describes the SPARQL query answering techniques implemented in *Ontop*. Section 4 outlines the applications of *Ontop*, in particular the Statoil and Siemens use cases in the context of the *Optique* project. Related SPARQL query answering systems are surveyed in Section 5. Section 6 is a retrospective on the development of *Ontop* over the past five years. Finally, Section 7 concludes the paper.

2. Architecture of *Ontop*

Ontop is an open-source[3] OBDA system released under the Apache license, developed at the Free University of Bozen-Bolzano. The *Ontop* system exposes relational databases as virtual RDF graphs by linking the terms (classes and properties) in the ontology to the data sources through mappings. This *virtual* RDF graph can then be queried using SPARQL, by translating the SPARQL queries into SQL queries over the relational databases. This translation process is transparent to the user.

The architecture of *Ontop*, which is illustrated in Fig. 1, can be divided in four different layers: *(i)* the inputs, i.e., the domain-specific artifacts such as the ontology, database, mappings, and queries; *(ii)* the core of the system in charge of query translation, optimization, and execution; *(iii)* the APIs exposing standard Java interfaces to users of the system; and *(iv)* the applications that allow end-users to execute SPARQL queries over databases.

We explore each of these components in turn.

2.1. Inputs: Ontology, Mappings, Queries, and Databases

To the best of our knowledge, *Ontop* is the first OBDA system that supports all the W3C recommendations related to OBDA: OWL 2 QL, R2RML, SPARQL, SWRL, and the OWL 2 QL entailment regime in SPARQL[4]. In addition, it supports all

[1] `https://github.com/ontop/ontop-examples/tree/master/swj-2015`

[2] `http://ontop.inf.unibz.it/`

[3] `http://github.com/ontop/ontop/`

[4] SWRL and the OWL 2 QL entailment regime are currently supported experimentally.

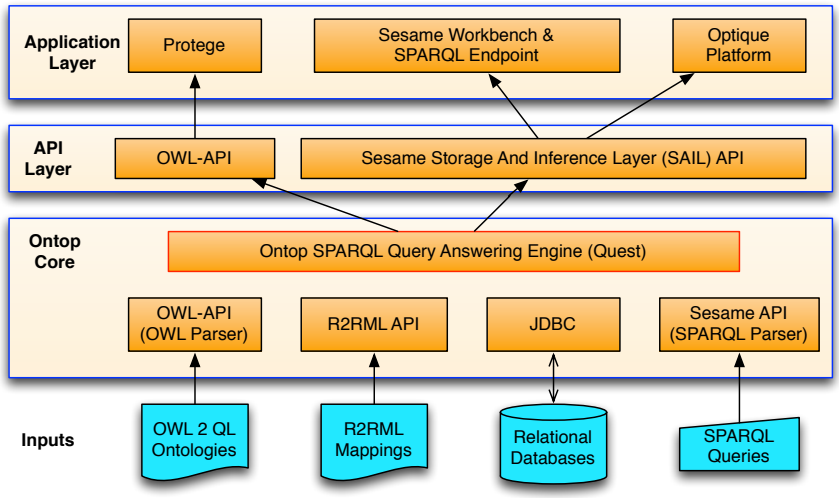


Fig. 1. Architecture of the *Ontop* system

the major commercial and open-source relational databases.

Ontology. *Ontop* allows for RDFS [6] and OWL 2 QL [24] as ontology languages. OWL 2 QL is based on the *DL-Lite* family of lightweight description logics [9,2], which guarantees that queries over the ontology can be rewritten into equivalent queries over the databases. Recently *Ontop* has been extended to support also a fragment of SWRL [43].

Example 2.1. The following ontology captures the domain knowledge of our running example. It describes the concepts of cancer and cancer patient with the following OWL axioms:

```
      :NSCLC rdfs:subClassOf :LungCancer .
       :SCLC rdfs:subClassOf :LungCancer .
 :LungCancer rdfs:subClassOf :Neoplasm .
:hasNeoplasm rdfs:domain :Patient .
:hasNeoplasm rdfs:range :Neoplasm .
    :hasName rdf:type owl:DatatypeProperty .
   :hasStage rdf:type owl:ObjectProperty .
```

In particular, classes `:NSCLS` and `:SCLC` are both subclasses of `:LungCancer` (that is, they both are types of lung cancer), which in turn is a subclass of `:Neoplasm`. The object property `:hasNeoplasm` has class `:Patient` as its domain and `:Neoplasm` as its range (in other words, it relates patients to neoplasms). We also have a datatype property `:hasName` and an object property `:hasStage`. □

Mappings. *Ontop* supports two mapping languages: the W3C RDB2RDF Mapping Language (R2RML), which is a widely used standard; and the native *Ontop* mapping language, which is easier to learn and use. *Ontop* allows users to convert native mappings into R2RML mappings and vice-versa. Intuitively, a mapping assertion consists of a source (an SQL query retrieving values from the database) and a target (defining RDF triples with values from the source).

Example 2.2. The ontology in Example 2.1 can be populated from the database in Example 1.1 by means of the following mappings:

```
:db1/{pid} rdf:type :Patient .
    ⟵ SELECT pid FROM tbl_patient
:db1/neoplasm/{pid} rdf:type :NSCLC .
    ⟵ SELECT pid, stage FROM tbl_patient
       WHERE type = false
:db1/neoplasm/{pid} rdf:type :SCLC .
    ⟵ SELECT pid, stage FROM tbl_patient
       WHERE type = true
:db1/{pid} :hasName {name} .
    ⟵ SELECT pid, name FROM tbl_patient
:db1/{pid} :hasNeoplasm :db1/neoplasm/{pid} .
    ⟵ SELECT pid FROM tbl_patient
:db1/neoplasm/{pid} :hasStage :stage-IIIa .
    ⟵ SELECT pid FROM tbl_patient
       WHERE stage = 4
```

(To ease readability we use a slightly simplified version of the *Ontop* native mappings syntax.) In this example, IRIs like `:hasStage` and `rdf:type` represent the constant components of the RDF triples. IRIs `:db1/{pid}` and `:db1/neoplasm/{pid}` are constructed using values from the database: in both cases `{pid}` is the value of the attribute `pid` in the respective SQL query. Similarly, `{name}` is the literal whose value is taken from the attribute `name` in the SQL

query of the mapping. Note that there are individuals that represent patients, `:db1/{pid}`, and individuals that represent their tumors, `:db1/neoplasm/{pid}`. This allows for a better modeling of the domain and allows the user to query specific properties of the tumor independently of the patient. □

Queries. *Ontop* supports essentially all features of SPARQL 1.0 and the OWL 2 QL entailment regime of SPARQL 1.1 [21]. Support for other features of SPARQL 1.1 (e.g., aggregates, property path queries and negation) is ongoing work.

Example 2.3. Recall our information need in Example 1.1: the names of all the patients who have a neoplasm (tumor) at stage IIIa. This can be represented by the following SPARQL query:

```
SELECT ?name WHERE {
  ?p rdf:type :Patient .
  ?p :hasName ?name .
  ?p :hasNeoplasm ?tumor .
  ?tumor :hasStage :stage-IIIa .}
```

The query would return `'Mary'` on our sample database. Observe that the vocabulary is more domain-oriented and independent of the encoding done in the database, e.g., there is no need anymore to use the specific values that encode types or stages. □

Databases. *Ontop* supports many relational database engines via JDBC. These include all major commercial relational databases (DB2, Oracle, and MS SQL Server) and the most popular open-source databases (PostgreSQL, MySQL, H2, and HSQL). In addition, *Ontop* can be used with federated databases (e.g., Teiid[5] or Exareme, formerly called ADP [42]) to support multiple data sources (e.g., relational databases, XML, CSV, and Web Services).

2.2. Ontop Core

The core of *Ontop* is the SPARQL engine *Quest*, which is in charge of rewriting SPARQL queries over the virtual RDF graph into SQL queries over the relational database (see Section 3).

[5] `http://teiid.jboss.org/`

2.3. API Layer

To allow developers build systems using *Ontop* as a Java library, *Ontop* implements two widely used Java APIs, which are also available as Maven artifacts:

- OWL API [15] is a reference implementation for creating, manipulating and serializing OWL ontologies. We extended the `OWLReasoner` interface to support SPARQL query answering.
- Sesame [7] is a de-facto standard framework for processing RDF data. *Ontop* implements the Sesame *Storage And Inference Layer* (SAIL) API supporting inferencing and querying over relational databases.

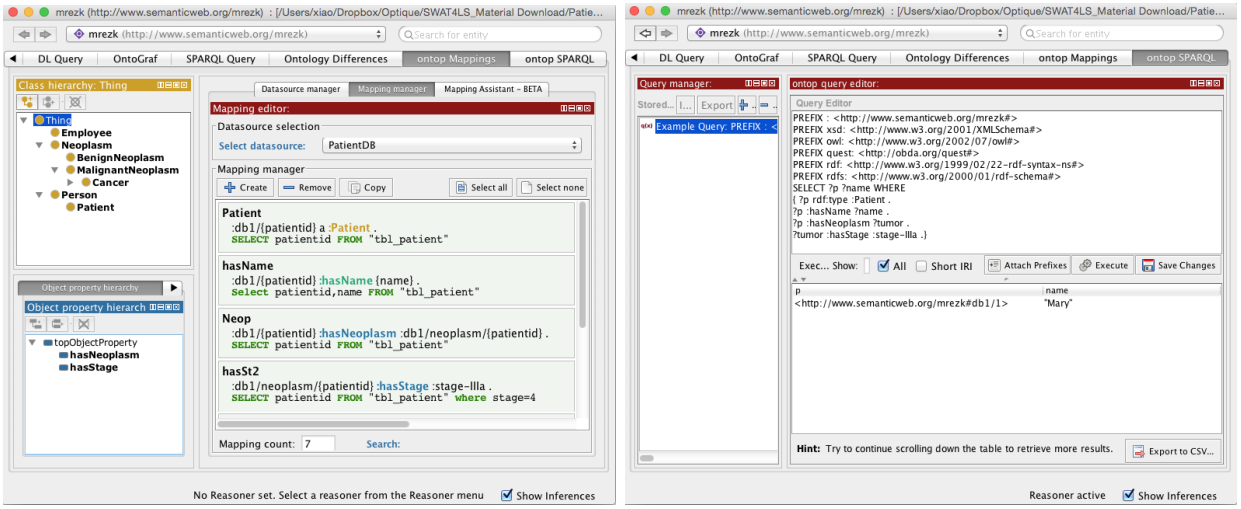
2.4. Application Layer

Ontop is available through a simple command line interface, but also through several applications accessing it via the above mentioned APIs. We describe three such applications, which we have been developing and maintaining together with *Ontop* over the past years.

– *Protégé.* *Ontop* implements a *Plugin for Protégé 4* based on OWL API. The plugin provides a graphical interface for various key functionalites related to OBDA: editing mappings, executing SPARQL queries, checking consistency of the ontology, automatically bootstrapping ontologies and mappings from the database, importing and exporting R2RML mappings, materializing RDF triples, etc. Figure 2 shows two screenshots of the *Ontop* Protégé Plugin for creating mappings and answering SPARQL queries from the running examples.

– *Sesame Workbench & SPARQL Endpoint*. Sesame OpenRDF Workbench is a web application for administrating Sesame repositories. We extended the Workbench to create and manage *Ontop* repositories using SAIL API. Such repositories can then be used as standard SPARQL endpoints. Figure 3 shows a screenshot of creating an *Ontop* repository in Sesame Workbench.

– *Optique Platform.* The Optique Platform complements *Ontop* by adding an intuitive visual query builder, tools for ontology and mapping management, a friendly query answering interface, and a database federation tool among other features [13]. *Ontop* is the core of the Optique Platform and is in charge of the query transformation module. The platform can query streaming data and exploit massive parallelism in the backend whenever possible.



(a) Mapping Editor (b) SPARQL Query Answering

Fig. 2. Screenshots of *Ontop* Protégé Plugin

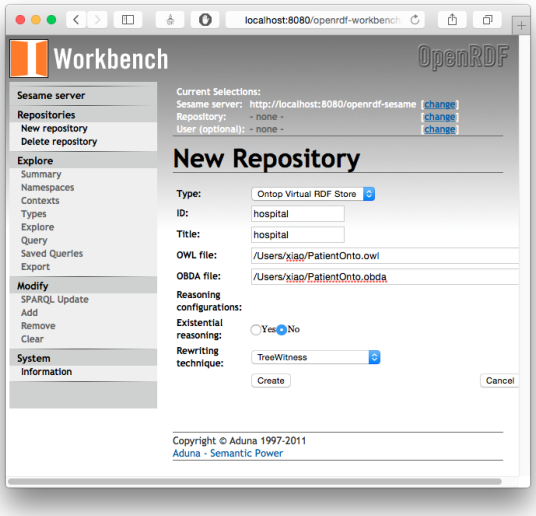


Fig. 3. Screenshot of *Ontop* Sesame Workbench

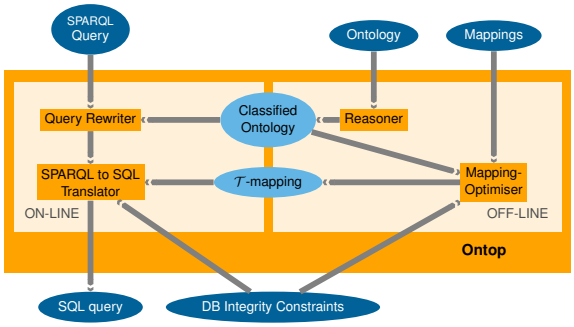


Fig. 4. The *Ontop* workflow

3. Answering SPARQL Queries

Ontop answers end-user's SPARQL queries by rewriting them into SQL queries and delegating their execution to the data sources. With this approach there is no need to apply rules to GBs of data to generate all the facts entailed by the ontology. The workflow of *Ontop* can be divided into the off-line and online stages and is illustrated in Fig. 4. The most critical task during start-up (the off-line stage) is compiling the ontology into the mappings and generating the so-called $\mathcal{T}$-mappings [34]. During the query execution (the online stage) *Ontop* transforms input SPARQL queries into optimized SQL queries exploiting the $\mathcal{T}$-mappings and the database integrity constraints. We now explain each of the two stages.

3.1. *Off-line Stage: Ontology and Mapping Compilation*

The off-line stage of *Ontop* processes the ontology, mappings, and database integrity constraints. This stage can be thought of as consisting of three phases: ontology classification, $\mathcal{T}$-mapping construction, and $\mathcal{T}$-mapping optimization. In the implementation of *Ontop*, however, the last two phases are performed simultaneously.

During the first phase, the ontology is loaded through OWL API and is classified using the built-in reasoner. The resulting complete hierarchy of properties and classes is stored in memory as a directed acyclic graph (DAG). For example, in the ontology in Example 2.1, both `:NSCLC` and `:SCLC` are subclasses of `:LungCancer`, which in turn is a subclass of `:Neoplasm`. It follows that every NSCLC and every SCLC is a form of Neoplasm:

```
:NSCLC  rdfs:subClassOf :Neoplasm .
 :SCLC  rdfs:subClassOf :Neoplasm .
```

The classification algorithm is based on a variant of graph reachability for the constructed DAG [30] (a similar algorithm was later described in [23]).

During the second phase, $\mathcal{T}$-mappings are constructed by composing the class and property hierarchies with the mappings [34,36]. For example, consider concept `:Neoplasm` in Example 2.1. Although it has no mappings defined by the user, the two inclusions derived above give rise to the following rules in the $\mathcal{T}$-mapping:

```
:db1/neoplasm/{pid} rdf:type :Neoplasm .
      ← SELECT pid, stage FROM tbl_patient
         WHERE type = false
:db1/neoplasm/{pid} rdf:type :Neoplasm .
      ← SELECT pid, stage FROM tbl_patient
         WHERE type = true
```

Finally, during the third phase, the $\mathcal{T}$-mappings are optimized by using disjunction (OR) and interval expressions in SQL and by applying Semantic Query Optimization (SQO) techniques (which will be described in Section 3.2.2). For instance, using disjunction, *Ontop* transforms the two rules above into the following single rule:

```
:db1/neoplasm/{pid} rdf:type :Neoplasm .
      ← SELECT pid, stage FROM tbl_patient
         WHERE type = false OR type = true
```

Such optimizations are known to be relatively expensive (for example, SQO is based on an NP-complete conjunctive query containment check) but are performed only once, during the off-line stage of *Ontop*, and therefore have no negative effect on query processing. On the other hand, the resulting $\mathcal{T}$-mappings produce all the RDF triples that can be inferred from the database and the ontology and so, during the online stage, the $\mathcal{T}$-mappings are used as the basis for the translation of individual triple patterns in SPARQL queries into SQL.

3.2. Online Stage: Query Answering

The online stage takes a SPARQL query and translates it into SQL by using the $\mathcal{T}$-mappings. We focus only on the translation of SELECT queries (ASK and DESCRIBE queries are treated analogously). In this process *Ontop* also optimizes the SQL query by applying the Semantic Query Optimization (SQO) techniques [11,20]. To ease the presentation we distinguish three phases in the query answering process:

(a) The SPARQL query is translated into SQL using $\mathcal{T}$-mappings.
(b) The resulting SQL query is optimized for efficient execution by the database engine.
(c) The optimized SQL query is then executed by the database engine, and the result set is translated into the answer to the original SPARQL query by creating necessary RDF terms.

Phases (a) and (b) are handled together in the implementation of *Ontop* (we, however, distinguish them here for the sake of clarity).

We elaborate now on the three phases of query answering.

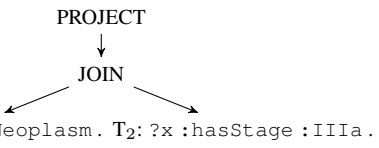
3.2.1. From SPARQL to SQL

Ontop internally represents the SPARQL query as a tree corresponding to the SPARQL algebra expression (generated by the Sesame SPARQL parser). Each node of the tree is transformed into the respective SQL expression. To illustrate the transformation, we continue with the running example.

Example 3.1. Consider the fragment of the query in Example 2.3 that retrieves all tumors of stage IIIa:

```
SELECT ?tumor WHERE {
   ?tumor rdf:type :Neoplasm ;
          :hasStage :stage-IIIa . }
```

(We note that the triple pattern `?tumor rdf:type :Neoplasm` was not needed in Example 2.3: indeed, *Ontop* can infer it from `?p :hasNeoplasm ?tumor` because the range of `:hasNeoplasm` is `:Neoplasm`.) The above query is represented by the following tree:



□

Input: SPARQL Query Q, $\mathcal{T}$-mappings $\mathcal{M}_\mathcal{T}$
Output: SQL Expression
1: S = List of nodes in Q in a bottom-up topological order
2: sqlM = a map from nodes to SQL expressions
3: **for** node $n \in S$ **do**
4: **if** n is triple pattern **then** ▷ Translating leaves
5: sqlM[n] = replace-Tmap-def(n, $\mathcal{M}_\mathcal{T}$)
6: **else** ▷ Translating non-leaf nodes
7: **if** n = JOIN(n_1,n_2) **then**
8: sqlM[n] = InnerJoin(sqlM[n_1],sqlM[n_2])
9: **else if** n = OPTIONAL(n_1, n_2, e) **then**
10: sqlM[n] = LeftJoin(sqlM[n_1],sqlM[n_2],e)
11: **else if** n = UNION(n_1, n_2) **then**
12: sqlM[n] = Union(sqlM[n_1],sqlM[n_2])
13: **else if** n = FILTER(n_1, e) **then**
14: sqlM[n] = Filter(sqlM[n_1], e)
15: **else if** n = PROJECT(n_1, p) **then**
16: sqlM[n] = Project(sqlM[n_1], p)
17: **end if**
18: **end if**
19: **end for**
20: **return** sqlM[S.last()]

Algorithm 1. Translating SPARQL into SQL

Next we explain how to produce the SQL expression from a SPARQL query using $\mathcal{T}$-mappings. Algorithm 1 is a simplified version of the process. It iterates over the nodes of the SPARQL algebra tree in a bottom-up fashion; more precisely, it goes through the list S in the topological sorting order. In our running example this list is [T_1, T_2, JOIN, PROJECT]. So, the algorithm starts by replacing each leaf of the tree, that is, a triple pattern of the form (s, p, o), by the union of the SQL queries defining its predicate (lines 4–5) in the $\mathcal{T}$-mapping. In this step, the algorithm implicitly considers two cases: (1) when p is an object or data property, such as `:hasStage` or `:hasName` or (2) when p is `rdf:type` and o is a class, such as `:Patient`.

Once it finishes processing the leaves, it continues to the upper levels in the tree (lines 7–17), where the SPARQL operators (PROJECT, JOIN, OPTIONAL, UNION, and FILTER) are translated into the corresponding SQL operators (Project, InnerJoin, LeftJoin, Union, and Filter, respectively). Once the root is translated the process is finished and the resulting SQL expression is returned.

Example 3.2. *Ontop* translates the SPARQL query in Example 3.1 into a SQL query (shown in Fig. 5a) with the following structure:

```
SELECT Q1.x FROM
((SELECT concat(":db1/neoplasm/", pid) AS x
   FROM tbl_patient
   WHERE type = false OR type = true) Q1
 JOIN
 (SELECT concat(":db1/neoplasm/", pid) AS x
   FROM tbl_patient WHERE stage = 4) Q2
 ON Q1.x = Q2.x)
```

(a) Non-optimized generated SQL query

```
SELECT concat(":db1/neoplasm/", Q.pid) AS x
FROM
 (SELECT T1.pid
  FROM tbl_patient T1 JOIN tbl_patient T2
     ON T1.pid = T2.pid
  WHERE (T1.type = false OR T1.type = true)
              AND T2.stage = 4) Q
```

(b) SQL query after the structural optimization

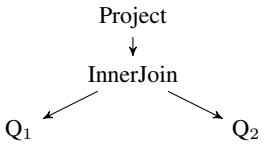
```
SELECT concat(":db1/neoplasm/", Q.pid) AS x
FROM
 (SELECT pid
  FROM tbl_patient
  WHERE (type = false OR type = true)
         AND stage = 4) Q
```

(c) SQL query after the self-join elimination

```
SELECT concat(":db1/neoplasm/", pid) AS x
FROM tbl_patient
WHERE (type = false OR type = true)
         AND stage = 4
```

(d) SQL query after the second structural optimization

Fig. 5. Example of SQL translation and optimization



The leaves, Q_1 and Q_2, are respectively the SQL definitions of the concept `:Neoplasm` and of the property `:hasStage` in the $\mathcal{T}$-mapping rules constructed during the off-line stage (see Section 3.1). Observe that without the $\mathcal{T}$-mapping optimizations in the off-line stage, the resulting SQL would contain a union in place of Q_1, increasing the complexity of the SQL query and so, having a negative effect on the query evaluation time. □

For the sake of simplicity we do not describe the translation of filter expressions and OPTIONALs (an

optimal translation of unions and empty expressions in the second argument is particularly challenging) and how to handle data types and functions in SQL expressions. Instead, we refer the interested reader to [38,21].

3.2.2. Optimizing the generated SQL queries

The generated SQL queries can already be executed by the DB engine but they are not necessarily efficient: they often contain subqueries, redundant self-joins and joins over complex expressions (such as string concatenations). Observe that joins over complex expressions, for instance, prevent the database engine from using indexes. To improve performance, *Ontop* employs a number of structural and semantic optimizations.

Structural Optimizations. *Ontop* applies three main structural optimizations: *(a)* pushing the joins inside the unions, *(b)* pushing the functions as high as possible in the query tree, and eliminating sub-queries. Returning to the running example, the SQL query obtained by these optimizations is shown in Fig. 5b: (b) and (c) convert the join over the complex expressions into a join over the attributes of the relations (effectively de-IRIing the join) and subsequently remove the subqueries.

Semantic Optimization. *Ontop* adopts techniques from the Semantic Query Optimization (SQO) [11,20] field. SQO refers to a set of techniques that perform the semantic analysis of SQL queries and transform them into a more efficient form, e.g., by removing redundant self-joins, and detecting unsatisfiable or trivially satisfiable conditions. SQO techniques often use database dependencies such as primary and foreign keys to reduce the size and complexity of the query. In our running example, these optimizations eliminate the self-join, which is redundant because `pid` is the primary key of `tbl_patient`; the result is shown in Fig. 5c. Observe that it has a subquery that could not be eliminated before, but because of SQO we can apply structural optimization (c) again to obtain a simpler SQL query, shown in Fig. 5d.

Observe that these optimizations complement and interact with each other. The optimization step is critical [22] and nontrivial. The translation of more complex queries is more involved and implies tackling the gap between the SQL and SPARQL semantics. This simple example is meant to provide an intuition; the interested reader is referred to [21,38].

3.2.3. Executing the Query over the Database

Since different database engines support slightly different SQL dialects, we have to adjust the SQL syntax accordingly. For instance, the operator for string concatenation is `||` in Postgres and `concat` in MySQL; in MySQL, one cannot cast a value to `Integer`, so we cast the value to `Signed` instead; Postgres internally changes unquoted identifiers (both column and alias names) to lowercase, while Oracle and H2 change unquoted identifiers to uppercase[6].

Finally *Ontop* sends the optimized SQL query to the database engine and translates the result into RDF terms (URIs or literals) to construct the answers to the SPARQL query. In the implementation, *Ontop* wraps the result set obtained from the database via JDBC and creates corresponding Java objects for OWL API or Sesame API.

3.2.4. Performance

The cost of query answering in *Ontop* can be split into three parts: *(i)* the cost of generating the SQL query, *(ii)* the cost of execution by the RDBMS, and *(iii)* the cost of fetching and transforming the SQL results into RDF terms. We have studied the performance of *Ontop* using several benchmarks (e.g., BSBM, FishMark, LUBM, NPD) and settings (e.g., different DB engines, number of clients, dataset size) [21,39,22,36]. The obtained results suggest that the performance of *Ontop* depends more on the complexity of the combination of ontology and mappings than on the size of the dataset. This is in line with the well-known theoretical results that state that the translation from SPARQL to SQL is exponential in the worst case [14]. In benchmarks like BSBM, FishMark, and LUBM, where the number of mappings and the number of ontological terms are small and the dataset ranges from 25 to 200 million triples, *Ontop* outperforms its competitors by orders of magnitude [39,21,36]. This performance is the result of *(i)* the fast SPARQL-to-SQL translation (around 4–15ms); *(ii)* the efficient optimization of the SQL; and *(iii)* the well-known efficiency of relational DB engines. For instance, in BSBM with 200 million triples, *Ontop* can run more than 400.000 queries per hour (44k query mixes per hour).

To better understand the performance of OBDA systems, we developed a more challenging benchmark, the NPD Benchmark [22], which reveals the strengths and pitfalls of OBDA. The benchmark comes with

[6] `https://github.com/ontop/ontop/wiki/Case-sensitivity-for-SQL-identifiers`

thousands of axioms in the ontology and rules in mappings, and a dataset containing up to 4 billion triples. The results comparing *Ontop* and Stardog show that indeed the approach is scalable but more work is needed to optimize the generated SQL queries.

The query set, which was obtained by interviewing users of the NPD dataset [41], can be divided into queries that are translated by *Ontop* into efficient SQL queries and queries that are translated into exponentially larger SQL queries. For instance, Query 12 from the benchmark is translated into a union of more than 10.000 SQL subqueries. *Ontop* outperforms Stardog whenever the SPARQL queries are not translated into very large SQL queries. In the remaining cases, Stardog outperforms *Ontop* by orders of magnitude. The results confirm that the exponential blow-up of the translation into SQL is a major source of performance loss in modern OBDA systems. If this issue is not handled properly, it can prevent OBDA systems from being deployed in production environments. We are currently working on different techniques for tackling this issue.

4. Industrial Applications

The adoption of *Ontop* by the community has been growing steadily since 2010: last year (2014) *Ontop* got 500 new registrations, the webpage got 7000 hits, and the mailing list in excess of 120 threads. Nowadays, *Ontop* is the core of the *Optique* Platform (developed in the *Optique* EU project [13]), and it is actively being used in academia[7] (e.g., in Semantic Mediator [5], for accessing electronic health records [31], and for querying temporal and streaming data in OBDA [26]) and being experimented with in industry.

In this section we describe the use cases of the two major industrial partners in the *Optique* Project, namely Siemens and Statoil, and what role *Ontop* is expected to play there.

Siemens. Siemens Energy is one of the four sectors of Siemens AG corporation. It is in charge of generating and delivering power from numerous sources. Siemens Energy runs several service centers for power plants. Each center monitors thousands of devices related to power generation, including gas and steam turbines, compressors, and generators. Each device is monitored by different sensors. All dynamic (observational) data from the sensors is stored in one large relational database (PostgreSQL) using more than 150 tables per device. About 30 GB of new sensor and event data is being generated every day, resulting in a total of 100 TB of timestamped data.

One of the main tasks of service engineers monitoring these devices is to promptly solve issues detected by gathering the relevant sensors data and analyzing it. Nowadays, the data gathering phase is the bottleneck of the process because it takes about 80% of the amount of time spent by engineers. The reason why the gathering phase is critical partially resides in the complexity and quantity of the data to be analyzed. Ideally the engineers should be able to access the data directly, by creating and combining queries in a way that is compatible with their knowledge. However, the data is often organized to better serve the applications rather than in a most intuitive way for the domain experts.

The OBDA approach to solving the problem consists of topping the database with an ontology that uses the terminology of the engineers and mediates between the engineers and the data [18]. Observe that in the Siemens scenario this requires handling historical and streaming data. It is worth noticing that OBDA with temporal and streaming data is still an open branch of research, and *Ontop* alone is not fully able to cope with all the requirements of this use case. However, *Ontop* can provide the backbone including the temporal and streaming dimensions in the *Optique* platform [26].

Statoil. Statoil is an international energy company with main activities in gas and oil extraction. It is headquartered in Norway and present in over 30 countries around the world.

Geologists at Statoil access several databases on a daily basis, of which one of the most relevant is the Exploration and Production Data Store (EPDS). EPDS is a large SQL database comprising over 1500 tables with information about historical exploration data (e.g., layers of rocks, porosity), production logs, maps, etc. It also contains business information such as license areas and companies. The schema is organized in such a way that direct data access by engineers (and geologists in particular) often becomes challenging or even impossible. Similarly to the Siemens use case, the main problem lies not only in the size of the schema and the data but also in the obscure structure of this legacy database. The solution currently adopted by Statoil relies on tools that enable domain experts to

[7] https://github.com/ontop/ontop/wiki/UseCases

combine different pre-defined queries. The problem of these pre-defined queries is that they often are too specific, or too general, or cannot be easily combined to obtain the desired results.

The role of *Ontop* (and *Optique*) in such scenario is helping the engineers formulate their own queries autonomously using the domain vocabulary [13]. Similarly to the Siemens use case, this will be achieved by enriching EPDS with an ontology expressed in terms familiar to the engineers.

5. Related SPARQL Query Answering Systems

In this section, we briefly review the most popular SPARQL query answering systems, which can be categorized into two major types: OBDA systems and triplestores. Their main features are summarized in Table 1.

Triplestores provide a flexible generic logical model that allows them to accept and store any set of RDF triples. However, if the triples are generated from external sources (e.g., relational databases) then an intermediate ETL (Extract, Transform and Load) process must be set up between these external sources and the triplestore. The ETL process can be expensive, especially when datasources are updated frequently.

OBDA systems, on the other hand, are set up over existing relational datasources and exploit their domain-specific schemas. By using ontologies and mappings, they expose the database as a virtual RDF graph that can be queried using SPARQL. No ETL is thus required.

Some triplestores and OBDA systems have reasoning capabilities. The most common strategy for triplestores is forward-chaining, which consists in extending the set of RDF triples by means of inferences according to a given set of rules. Thus, the OWL 2 RL profile of OWL 2 (and similar rule-based ontology languages) are most suitable for triplestores. Forward-chaining has some drawbacks: the inferences can be costly in terms of both time and space; moreover, updates and deletions of triples require additional bookkeeping for incremental reasoning.

In contrast to triplestores, the most common strategy for OBDA systems is query rewriting, and so the profile of OWL that is most suitable for this setting is OWL 2 QL. To guarantee rewritability, certain features, such as recursion and property chains, are not allowed in OWL 2 QL.

In the remainder of this section, we review various implementations from these two categories.

5.1. Triplestores

Virtuoso Universal Server[8] is a hybrid DBMS that can be used as a relational database, a triplestore, or an OBDA system. It has two editions: an open-source and a commercial one. From the perspective of answering SPARQL queries, Virtuoso is mostly used as a triplestore. It supports SPARQL 1.1 and, in this mode, it offers some backward- (by default) and forward-chaininqg capabilities for a limited subset of RDFS and OWL. When Virtuoso is used as a regular RDBMS, it can be extended into an OBDA system by setting up mappings in its own mapping language. However, this OBDA mode has several limitations: no reasoning capability is available and only a small fragment of R2RML is supported. Virtuoso can be accessed through the Sesame and Jena APIs.

GraphDB,[9] previously known as OWLIM [4], is a commercial triplestore developed by Ontotext. It fully supports SPARQL 1.1. OWL reasoning support is based on the forward-chaining rule-based materialization approach. This strategy naturally fits with the OWL 2 RL profile but is incomplete for OWL 2 QL [3]. GraphDB is accessible through the Sesame API.

Stardog[10] is a commercial triplestore developed by Clark&Parsia. It supports SPARQL 1.1 and several reasoning levels: RDFS, the three profiles (OWL 2 QL, OWL 2 EL, OWL 2 RL), and OWL 2 DL (however, completeness in OWL 2 DL is guaranteed only for schema reasoning). The reasoning level can be chosen by the user at query time. Stardog avoids eager materialization and the reasoning is partly based on query rewriting. It can be accessed through the Sesame API.

RDFox[11] is an in-memory triplestore developed at the University of Oxford. It implements a novel shared-memory parallel Datalog reasoning algorithm and supports OWL 2 RL reasoning by materialization [25]. The system is a cross-platform software written in C++ and comes with a Java wrapper supporting OWL API.

5.2. OBDA systems

D2RQ[12] is one of the pioneering OBDA systems, developed at the Free University of Berlin and DERI.

[8] http://virtuoso.openlinksw.com/
[9] http://www.ontotext.com/products/ontotext-graphdb/
[10] http://stardog.com/
[11] http://www.cs.ox.ac.uk/isg/tools/RDFox/
[12] http://d2rq.org/

	System	Reasoning	Mapping support	License	Starting year
Triplestore	Virtuoso	RDFS*	Native, R2RML*	GPL 2, Commercial	1999
	GraphDB	OWL 2 RL	-	Commercial	2005
	Stardog	OWL 2 */ SWRL*	-	Commercial	2012
	RDFox	OWL 2 RL / SWRL / Datalog	-	Academic	2013
OBDA	D2RQ	No	D2RQ Mapping, R2RML*	Apache 2	2004
	Mastro	OWL 2 QL	R2RML*	Academic	2006
	Ultrawrap	RDFS-Plus	Native, R2RML	Commercial	2012
	Morph-RDB	No	R2RML	Apache 2	2013
	Ontop	OWL 2 QL / SWRL*	*Ontop* Mapping, R2RML	Apache 2	2010

Table 1
Feature matrix of SPARQL query answering systems
(* indicates limited support)

This query rewriting system provides some query optimizations but these have often been reported as insufficient: for instance, the generated SQL queries can contain an excessive number of joins [29]. It provides its own mapping language, D2RQ, and supports only a fragment of R2RML. No inference mechanism is included. This software (last release in 2012) is available under an open-source license.

Mastro[13] is an OBDA system that shares common origins with *Ontop*. This query rewriting system supports reasoning over OWL 2 QL ontologies. Unlike other OBDA systems mentioned here, it supports only a restricted fragment of SPARQL that corresponds to unions of conjunctive queries. Mastro is available only for demonstration, testing, and evaluation purposes.

Ultrawrap[14] is an OBDA system commercialized by Capsenta. It was recently extended to support inference over an extension of RDFS with inverse and transitive properties [40]. Ultrawrap uses an analogue of $\mathcal{T}$-mappings of *Ontop*, which are called saturated mappings and which are used for creating regular and materialized views in the relational database.

Morph-RDB,[15] formerly called ODEMapster, is an open-source OBDA system supporting the R2RML and Direct Mappings standards. This system implements a number of query optimizations techniques such as the self-join elimination [29]. It, however, has no OWL inference capability.

[13] http://www.dis.uniroma1.it/~mastro/
[14] http://capsenta.com
[15] https://github.com/oeg-upm/morph-rdb

6. A Retrospective

Ontop has its roots in our initial work on the QuOnto and Mastro [1,8] systems (2006), in the OBDA plugin for Protégé [28], and in our DIG extension for Mastro [33]. QuOnto is a reasoner for the description logic *DL-Lite* with plain conjunctive query (CQ) answering. Mastro extends QuOnto with support for GAV (global as view) mappings for relational databases [27]. Both systems are maintained by Sapienza University of Rome. Our work enabled the use of these systems through the ontology editor Protégé 3 and the DIG reasoner API.

Using these tools we interacted with third parties to develop several OBDA applications [10,8,17,37,32] (for a full list, see [32]). Such interactions allowed us to test both the performance of the state-of-the-art query rewriting techniques and the feasibility of applying this technology in data integration and data access. The insights we obtained concern technique/optimization issues and API/features issues. These two paths of development characterized our work from then. We now briefly elaborate on them.

Reasoning, Optimization and Performance. The main issue initially was the large number of CQs generated by the rewriting technique implemented in QuOnto (the PerfectRef algorithm [9]), which sometimes, even for simple ontologies and mappings, produces in the order of hundreds of thousands of CQs. And although DBMSs do perform very well in general, they have problems with automatically generated queries (and this applies to both commercial and non-commercial database engines). To deal with the issue, we extended the PerfectRef algorithm by the Semantic Query Optimization (SQO) component, which removes redundant CQs and eliminates redundant self-

joins using database dependencies (foreign and primary keys) [32].

The work in this direction materialized in the first version of *Ontop* (2010), which at the time was called Quest (the name now refers only to the query answering engine). Quest focused on RDFS and OWL 2 QL (instead of *DL-Lite*) and SPARQL (instead of CQs). This required a new mapping language suitable for the ontology languages (RDFS and OWL). Quest is capable of working in two modes: *(i)* the virtual mode supports virtual RDF graphs via mappings, and *(ii)* the RDF triplestore mode stores triples directly in a relational database. Two further ideas were developed for improving performance: $\mathcal{T}$-mappings for the virtual mode [34,21] (c.f. Section 3.1) and the Semantic Index for the RDF triplestore mode [35]. The tree-witness query rewriting algorithm [19] was implemented to reduce the size of rewritings and take advantage of the $\mathcal{T}$-mappings and the Semantic Index. We also studied the execution of the SQL queries produced by *Ontop* and observed [32] that the generic database-centric SQO was not sufficient and other techniques were required for the issues specific to the context of mappings and ontologies (e.g., join conditions can be simplified by de-IRIing and the resulting joins eliminated as explained in Section 3.2.2).

The more recent lines of research in *Ontop* include the formalization of SPARQL in the context of OBDA [38], under the OWL 2 QL/RDFS entailment regimes [21], and SWRL with limited forms of recursion supported by SQL Common Table Expressions [43].

API, Features and Accessibility. With the first version of *Ontop*, we shifted our focus from the Description Logic domain to Semantic Web technologies, gradually increasing our support for RDF, RDFS, OWL 2 QL, and SPARQL. To support the OWL community, we included the OWL API and Protégé 4 interfaces for *Ontop*. To support RDF and the Linked Data community we created the Sesame API interface for *Ontop*, as well as an HTTP SPARQL end-point. With the publication of R2RML as the official W3C RDB2RDF mapping language, we extended *Ontop* to support it.

Ontop was initially released under a non-commercial use license before adopting the permissive Apache 2.0 license in 2013. In addition, the project is now hosted in GitHub so that anybody can easily download and contribute to it. On the software engineering side, to facilitate integration, building, testing, and distribution, *Ontop* was repackaged as a Maven project and has been available from the official Maven repository since 2013. We gradually introduced project-wide testing, starting with functional tests for the reasoning modules, query answering modules (including the DAWG tests for SPARQL 1.0), and virtual RDF modules (including the DAWG tests for R2RML). Now most JUnit tests ($\sim$2000) are automatically run with Travis-CI whenever new changes are pushed to GitHub.

7. Conclusion

In this paper we presented *Ontop*, a mature open-source OBDA system. It allows users to access relational databases through a conceptual representation of the domain of interest in terms of an ontology. The system is based on solid theoretical foundations and has been designed and implemented towards compliance with relevant W3C standards. It supports all major relational databases and implements multiple optimization techniques to offer a good level of performance. *Ontop* has been adopted in several academic and industrial use cases.

In the future, we plan to extend *Ontop* in the following directions:

(i) In order to further improve performance, we will investigate data-dependent optimizations.
(ii) We plan to support larger fragments of the SPARQL (e.g., aggregation, negation, and path queries) and R2RML (e.g., named graphs) standards.
(iii) For end-users, we will improve the GUI interfaces and add utilities to make *Ontop* even more user-friendly.
(iv) We plan to go beyond relational databases and support other kinds of data sources (e.g., graph databases and document-oriented databases).

Acknowledgements. 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] Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. QUONTO: QUerying ONTOlogies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI)*, pages 1670–1671, 2005.

[2] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The *DL-Lite* family and relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009.

[3] Barry Bishop and Spas Bojanov. Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM. In *Proc. of the 8th Int. Workshop on OWL: Experiences and Directions (OWLED)*, volume 796 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2011.

[4] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web J.*, 2(1):33–42, 2011.

[5] Béatrice Bouchou and Cheikh Niang. Semantic mediator querying. In *Proc. of the 18th Int. Database Engineering & Applications Symposium (IDEAS)*, pages 29–38. ACM Press, 2014.

[6] Dan Brickley and R. V. Guha. RDF vocabulary description language 1.0: RDF Schema. W3C Recommendation, World Wide Web Consortium, February 2004. Available at `http://www.w3.org/TR/rdf-schema/`.

[7] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *Proc. of the 1st Int. Semantic Web Conf. (ISWC)*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002.

[8] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The Mastro system for ontology-based data access. *Semantic Web J.*, 2(1):43–53, 2011.

[9] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[10] Diego Calvanese, C. Maria Keet, Werner Nutt, Mariano Rodriguez-Muro, and Giorgio Stefanoni. Web-based graphical querying of databases through an ontology: the WONDER system. In *Proc. of the 25th ACM Symposium on Applied Computing (SAC)*, pages 1388–1395, 2010.

[11] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Trans. on Database Systems*, 15(2):162–207, 1990.

[12] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium, September 2012. Available at `http://www.w3.org/TR/r2rml/`.

[13] Martin Giese, Peter Haase, Ernesto Jiménez-Ruiz, Davide Lanti, Özgür Özçep, Martin Rezk, Riccardo Rosati, Ahmet Soylu, Guillermo Vega-Gorgojo, Arild Waaler, and Guohui Xiao. Optique – zooming in on big data access. *IEEE Computer*, Accepted for publication, 2015.

[14] Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artificial Intelligence*, 213:42–59, 2014.

[15] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web J.*, 2(1):11–21, 2011.

[16] Joseph S. Nye Jr. The benefits of soft power. Technical report, Harvard University - Business School, 2004. Available at `http://hbswk.hbs.edu/archive/4290.html`.

[17] C. Maria Keet, Ronell Alberts, Aurona Gerber, and Gibson Chimamiwa. Enhancing web portals with Ontology-Based Data Access: the case study of South Africa's Accessibility Portal for people with disabilities. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED)*, volume 432 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2008.

[18] Evgeny Kharlamov, Nina Solomakhina, Özgür Lütfü Özçep, Dmitriy Zheleznyakov, Thomas Hubauer, Steffen Lamparter, Mikhail Roshchin, Ahmet Soylu, and Stuart Watson. How semantic technologies can enhance data access at Siemens Energy. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC)*, volume 8796 of *Lecture Notes in Computer Science*, pages 601–619. Springer, 2014.

[19] S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 275–285, 2012.

[20] Jonathan J. King. QUIST: A system for semantic query optimization in relational databases. In *Proc. of the 7th Int. Conf. on Very Large Data Bases (VLDB)*, pages 510–517. IEEE Computer Society, 1981.

[21] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC)*, volume 8796 of *Lecture Notes in Computer Science*, pages 552–567. Springer, 2014.

[22] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese. The NPD benchmark: Reality check for OBDA systems. In *Proc. of the 18th Int. Conf. on Extending Database Technology (EDBT)*, 2015.

[23] Domenico Lembo, Valerio Santarelli, and Domenico Fabio Savo. Graph-based ontology classification in OWL 2 QL. In *Proc. of the 10th Extended Semantic Web Conf. (ESWC)*, volume 7882 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2013.

[24] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, World Wide Web Consortium, December 2012. Available at `http://www.w3.org/TR/owl2-profiles/`.

[25] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel materialisation of Datalog programs in centralised, main-memory RDF systems. In *Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 129–137. AAAI Press, 2014.

[26] Özgür Lütfü Özçep and Ralf Möller. Ontology based data access on temporal and streaming data. In *Reasoning Web. Reasoning on the Web in the Big Data Era – 10th Int. Summer School Tutorial Lectures (RW)*, volume 8714 of *Lecture Notes in Computer Science*, pages 279–312. Springer, 2014.

[27] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[28] Antonella Poggi, Mariano Rodríguez-Muro, and Marco Ruzzi. Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proc. of the 4th Int. Workshop on OWL: Experiences and Directions (OWLED DC)*, 2008.

[29] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. Formalisa-

tion and experiences of R2RML-based SPARQL to SQL query translation using Morph. In *Proc. of the 23rd Int. World Wide Web Conf. (WWW)*, pages 479–490, 2014.

[30] S. Pugacs. Efficient query answering with semantic indexes. BSc thesis, KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, 2011.

[31] Alireza Rahimi, Siaw-Teng Liaw, Jane Taggart, Pradeep Ray, and Hairong Yu. Validating an ontology-based algorithm to identify patients with type 2 diabetes mellitus in electronic health records. *Int. J. Medical Informatics*, 83(10):768–778, 2014.

[32] Mariano Rodríguez-Muro. *Tools and Techniques for Ontology Based Data Access in Lightweight Description Logics*. PhD thesis, KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, 2010.

[33] Mariano Rodriguez-Muro and Diego Calvanese. Towards an open framework for ontology based data access with Protégé and DIG 1.1. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED)*, volume 432 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2008.

[34] Mariano Rodríguez-Muro and Diego Calvanese. Dependencies: Making ontology based data access work in practice. In *Proc. of the 5th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW)*, volume 749 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2011.

[35] Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over *DL-Lite* ontologies. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 308–318, 2012.

[36] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proc. of the 12th Int. Semantic Web Conf. (ISWC)*, volume 8218 of *Lecture Notes in Computer Science*, pages 558–573. Springer, 2013.

[37] Mariano Rodríguez-Muro, Lina Lubyte, and Diego Calvanese. Realizing ontology based data access: A plug-in for Protégé. In *Proc. of the ICDE Workshop on Information Integration Methods, Architectures, and Systems (IIMAS 2008)*, pages 286–289. IEEE Computer Society Press, 2008.

[38] Mariano Rodriguez-Muro and Martin Rezk. Efficient SPARQL-to-SQL with R2RML mappings. Technical report, Free University of Bozen-Bolzano, January 2014. Available at `http://www.inf.unibz.it/~mrezk/pdf/sparql-sql.pdf`.

[39] Mariano Rodriguez-Muro, Martin Rezk, Josef Hardi, Mindaugas Slusnys, Timea Bagosi, and Diego Calvanese. Evaluating SPARQL-to-SQL translation in Ontop. In *Proc. of the 2nd Int. Workshop on OWL Reasoner Evaluation (ORE)*, volume 1015 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, pages 94–100, 2013.

[40] Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. OBDA: Query rewriting or materialization? In practice, both! In *Proc. of the 13th Int. Semantic Web Conf. (ISWC)*, volume 8796 of *Lecture Notes in Computer Science*, pages 535–551. Springer, 2014.

[41] Martin G. Skjæveland and Espen H. Lian. Benefits of publishing the Norwegian Petroleum Directorate's FactPages as Linked Open Data. In *Proc. of Norsk informatikkonferanse (NIK 2013)*. Tapir, 2013.

[42] Manolis M. Tsangaris, George Kakaletris, Herald Kllapi, Giorgos Papanikos, Fragkiskos Pentaris, Paul Polydoras, Eva Sitaridi, Vassilis Stoumpos, and Yannis E. Ioannidis. Dataflow processing and optimization on grid and cloud infrastructures. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 32(1):67–74, 2009.

[43] Guohui Xiao, Martin Rezk, Mariano Rodriguez-Muro, and Diego Calvanese. Rules and ontology based data access. In *Proc. of the 8th Int. Conf. on Web Reasoning and Rule Systems (RR)*, volume 8741 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 2014.