Semantic Web 0 (0) 1
IOS Press

Ontop: Answering SPARQL Queries over Relational Databases

Editor(s): Óscar Corcho, Universidad Politécnica de Madrid, Spain
Solicited review(s): Jean Paul Calbimonte, École Polytechnique Fédérale de Lausanne, Switzerland; José Luis Ambite, University of Southern California, USA; one anonymous reviewer

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

Abstract. 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, which avoids materializing triples and is implemented through the query rewriting technique, 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 operated with a single 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 them. The focus 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 [28]. To separate the wheat from the chaff, 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 for tackling this challenge [42]. In OBDA, a conceptual layer is provided in the form of an *ontology* that defines a shared vocabulary, models the domain, hides the structure of the data sources, and enriches 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 the data. The W3C standard R2RML [17] was cre-

ated with the goal of providing a language for specifying 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. This virtual RDF graph can be *materialized*, generating RDF triples to be used with RDF triplestores, or alternatively it 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 database systems (efficient query answering, security, robust transaction support, etc.).

To illustrate these concepts and different notions in this article, we will use the following running example. All the material required 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 on 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:

- NSCLC: `1`–`6` for stages I, II, III, IIIa, IIIb, and IV, respectively;
- SCLC: `1` and `2` for stages Limited and Extensive, respectively.

Our sample table contains the following data:

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

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, because in order to formulate the query and to understand and analyze the results we need to know how the information is encoded in the data. In this paper, we describe how to use the *Ontop* system to address this challenge by enhancing the database with a semantic layer. □

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 plugin, a SPARQL endpoint through Sesame Workbench, and a Java library supporting OWL API and Sesame API.

The structure of the article is as follows. Section 2 presents a high-level overview of the architecture of *Ontop*. Section 3 surveys additional tools that can be used with *Ontop* for creating, deploying, and querying OBDA systems. Section 4 describes the SPARQL query answering techniques implemented in *Ontop*. Section 5 outlines the applications of *Ontop*, in particular the Statoil and Siemens use cases in the context of the *Optique* EU project. Related SPARQL query answering systems are surveyed in Section 6. Section 7 is a retrospective on the development of *Ontop* over the past five years. Finally, Section 8 concludes the article.

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 layers: *(i)* the inputs, i.e., the domain-specific artifacts such as the ontology, mappings, database, 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.

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

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

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

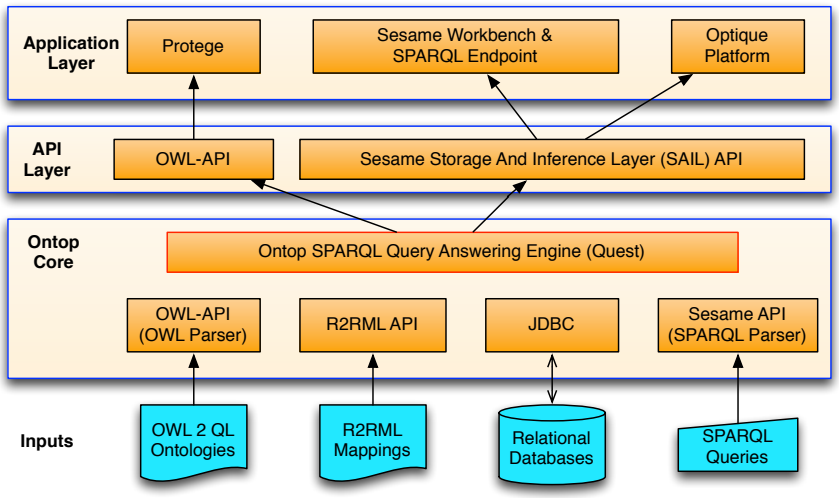


Fig. 1. Architecture of the *Ontop* system

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 major commercial and open-source relational databases.

Ontology. *Ontop* uses RDFS [8] and OWL 2 QL [39] as ontology languages. OWL 2 QL is based on the *DL-Lite* family of lightweight description logics [11,3], which guarantees that queries over the ontology can be rewritten into equivalent queries over the data alone. Recently *Ontop* has been extended to support also a fragment of SWRL [61].

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 a owl:DatatypeProperty .
   :hasStage a owl:ObjectProperty .
```

In particular, classes `:NSCLC` and `:SCLC` are subclasses of `:LungCancer` (that is, 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* includes tools for converting native mappings into R2RML mappings and vice-versa. Intuitively, a mapping assertion consists of a source, which is an SQL query retrieving values from the database, and a target, which constructs 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 in the simplified *Ontop* native mapping syntax:

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

```
:db1/{pid} a :Patient .
    ← SELECT pid FROM tbl_patient
:db1/neoplasm/{pid} a :NSCLC .
    ← SELECT pid FROM tbl_patient
      WHERE type = false
:db1/neoplasm/{pid} a :SCLC .
    ← SELECT pid 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 and type = false
```

In this example, IRIs like `:hasStage` and `rdf:type` (abbreviated as `a`) 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 a 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. □

Mappings are also used for data integration. To model an entity, for instance, a patient, that is represented by different objects in different datasources, there are in principle two options. First, one can virtually merge different objects representing the same entity by generating the same URI for them. Second, when the first option is not available, one can use `owl:sameAs` in the target of the mappings to explicitly state the equality between objects [12].

Queries. *Ontop* supports essentially all features of SPARQL 1.0 as well as the OWL 2 QL entailment regime of SPARQL 1.1 [35]. Implementation of other features of SPARQL 1.1 (e.g., aggregates, property path queries, negation) is ongoing work.

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

```
SELECT ?name WHERE {
  ?p a :Patient ;
     :hasName ?name ;
     :hasNeoplasm ?tumor .
  ?tumor a :Neoplasm ;
         :hasStage :stage-IIIa . }
```

On our sample database, the query would return `'Mary'`. Observe that the vocabulary is more domain-oriented and independent of the representation in the database and there is no need to be aware of the specific values that encode types or stages of cancer in the database. □

Databases. *Ontop* supports standard relational database engines via JDBC. These include all major commercial relational databases (DB2, Oracle, MS SQL Server) and the most popular open-source databases (PostgreSQL, MySQL, H2, HSQL). In addition, *Ontop* can be used with federated databases (e.g., Teiid[5] or Exareme[6], formerly called ADP [59]) 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 and ontology into SQL queries over the relational database (see Section 4).

2.3. API Layer

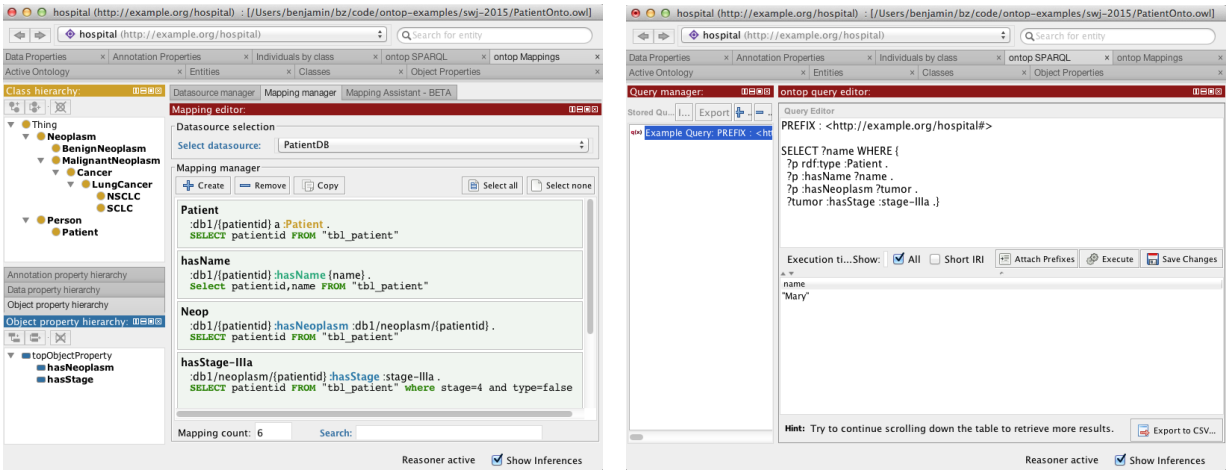
System developers can use *Ontop* as a Java library: *Ontop* implements two widely-used Java APIs, which are also available as Maven artifacts.

- OWL API [25] is a reference implementation for creating, manipulating, and serializing OWL ontologies. We extended the `OWLReasoner` Java interface to support SPARQL query answering.
- Sesame [9] 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 also available through a simple command-line interface and through several applications accessing it via the aforementioned APIs. We describe three such applications, which we have developed and maintained together with *Ontop* over the past years.

[5] `http://teiid.jboss.org`
[6] `http://www.exareme.org`



(a) Mapping editor

(b) SPARQL query answering

Fig. 2. Screenshots of the *Ontop* Protégé Plugin

– *Protégé. Ontop* implements a *plugin for Protégé* based on OWL API. The plugin provides a graphical interface for various key functionalities related to OBDA: editing mappings, executing SPARQL queries, checking consistency of the ontology, 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 example.

– *Sesame Workbench and 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 user-friendly query answering interface, and a database federation tool, among other features [22]. *Ontop* is the core of the Optique Platform and is in charge of the query transformation module. The platform can exploit massive parallelism in the backend whenever possible, and it also supports streaming data (in the streaming scenario, *Ontop* is used only for query rewriting).

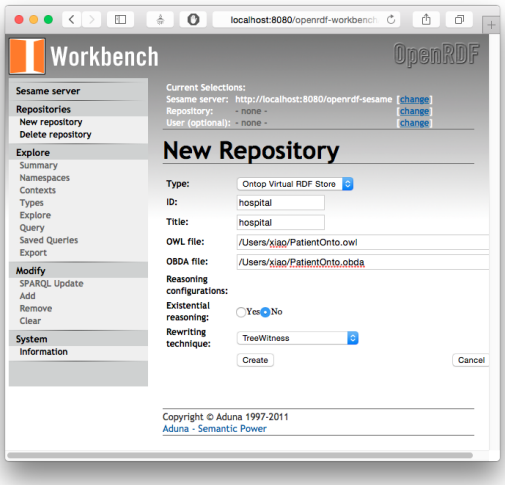


Fig. 3. Screenshot of the *Ontop* Sesame Workbench

3. Ecosystem

Complementary to *Ontop*'s core functionalities, there are additional tools that support the tasks involved in creating and deploying OBDA systems. We now provide a brief overview of some of the tools for bootstrapping mappings and ontologies, for federating data sources at SQL and SPARQL levels, and for formulating queries.

3.1. Mapping Generation

The process of creating mappings is probably the most complex step in setting up an OBDA system. It involves writing individual queries for each table and column that needs to be aligned with the ontology's vocabulary. A number of tools for (semi-)automatic creation of mappings have been implemented.

Mapping Bootstrappers automatically generate mappings and vocabularies from database schemas. *Ontop* includes a mapping bootstrapper, which is available as a command-line tool and as part of the Protégé plugin. Most of the existing tools generate mappings that follow the *Direct Mapping*[7] specification, a W3C recommendation for a default way of generating RDF triples from SQL databases. The specification contains guidelines on *(i)* how to use the values of primary, foreign, and unique keys to create IRIs for subjects and objects in RDF triples, *(ii)* how to use table names to define classes, and *(iii)* how to use table and column names to define properties. The specification also deals with low-level issues such as casting SQL values to RDF literals, handling null values, etc. Although the (default) direct mapping is not necessarily useful as a mapping by itself, many mapping bootstrappers take it as a baseline for mapping generation.

It is important to note, however, that a direct mapping is usually not sufficient to capture the semantics of the data such as class hierarchies encoded in columns with codes or IDs (e.g., `type = false` indicates small cell lung cancer), or the conditions that a certain value in a column identifies the object(s) that belong to a certain class (e.g., a patient with stage 1 to 4 is a high risk patient). Systems like MIRROR [21] and BootOX [27] support automatic generation of such complex mappings by exploiting the common patterns that schema managers use to encode the semantics of the domain in the relational database.

Once mappings have been generated with any of these tools, any query engine for virtual RDF graphs like *Ontop* can use them.

Semi-automatic Mapping Generation. Mappings can be constructed semi-automatically: the system suggests new mappings by analyzing the data sources and the existing mappings and the user guides the process. Such systems, e.g., Karma [34] and Clio [20], rely on schema matching techniques developed for data integration [18].

[7] `http://www.w3.org/TR/rdb-direct-mapping`

3.2. Ontology Bootstrapping and Matching

A basic vocabulary of classes and properties can be obtained from the table and column names in a database. However, such a vocabulary lacks ontological axioms that describe its semantics. *Ontology bootstrappers* are tools that extract RDFS or OWL axioms using schema information (such as integrity constraints) and/or the data in the database. For instance, BootOX [27] can be used for ontology bootstrapping.

However, the quality of automatically bootstrapped ontologies is usually not sufficient to allow their direct use for querying data sources. Moreover, users might want to use also well-established domain ontologies in combination with bootstrapped ones. So, the bootstrapped and domain ontologies need to be aligned. Ontology matching techniques [19], which are able to perform such alignment, can be seen as an extension of the schema matching techniques mentioned earlier. For example, BootOX uses the ontology matching tool LogMap [26].

3.3. SQL Federation

One way in which *Ontop* supports data integration scenarios is through SQL federation. A *federated database* is a DBMS that maps multiple independent databases into a single *virtual* schema. The designer of a federated database chooses how to map the independent databases into the virtual schema by, e.g., creating one-to-one mappings, renaming elements of the schema, or creating virtual tables from SQL views. Some federated databases also use wrappers for non-SQL databases (e.g., XML) to provide a uniform user interface for the client. SQL execution is coordinated by the federation engine, which exploits techniques for planning and executing cross-database joins with guarantees of concurrency and transaction control.

Most major DBMSs support federated schemas that integrate independent servers of their own kind: for example, MySQL creates federated schemas over independent MySQL servers. Federated schemas with databases from multiple vendors are supported by systems like UnityJDBC, IBM Websphere, MS SQL Server, and Oracle; open-source solutions are provided by JBoss's Teiid and Exareme, both of which are supported by *Ontop*.

3.4. SPARQL Federation

Another setup in which *Ontop* can be used in data integration scenarios is through SPARQL federation. As with federated SQL databases, SPARQL federation involves multiple and independent SPARQL end-points that are queried through a single entry point. We distinguish two forms of federation available: seamless federation and SPARQL 1.1 SERVICE federation.

Seamless federation is very similar to SQL federation. That is, a system manager creates a federated SPARQL end-point where she configures access to independent and remote end-points. Clients submit plain SPARQL queries to the federated end-point, unaware of the existence of the remote end-points. As with SQL federation, the federation system is responsible for finding the most efficient way of executing queries (in particular, it is extremely important to minimize the amount of data transferred between end-points). Anapsid [2] and FedX [55] are examples of such systems.

In contrast, *SERVICE federation* involves direct references to remote end-points in SPARQL 1.1 queries: the `SERVICE` keyword is used to scope a subgraph to a particular end-point. For example:

```
SELECT ?s ?o {
  ?s a :Patient .
  SERVICE ex:endpoint { ?s foaf:knows ?o } }
```

When a `SERVICE` keyword is encountered, the SPARQL engine delegates the evaluation of the enclosed graph pattern to the SPARQL end-point specified in the `SERVICE` call. The result is retrieved by the local SPARQL engine and used to continue the evaluation of the SPARQL query. SPARQL federation through `SERVICE` calls is available in most SPARQL 1.1 compliant engines.

In contrast to seamless federation, using the `SERVICE` keyword does not require the federation engine to know about the remote end-points *a priori*. In fact, the end-point URL itself can be a variable in the query, getting values as the query gets executed.

Ontop can be used in both of these setups by deploying an *Ontop* SPARQL end-point (see Section 2.4).

3.5. Ontology-based Query Interface

The task of formulating SPARQL queries can be challenging for end-users. There are several ontology-based visual query interfaces to ease this task, e.g., OptiqueVQS [58], QueryVOWL [24], SEWASIE VQS [15], and Faceted search [60]. These tools can be used together with *Ontop*. In particular, OptiqueVQS is the query interface in the *Optique* platform.

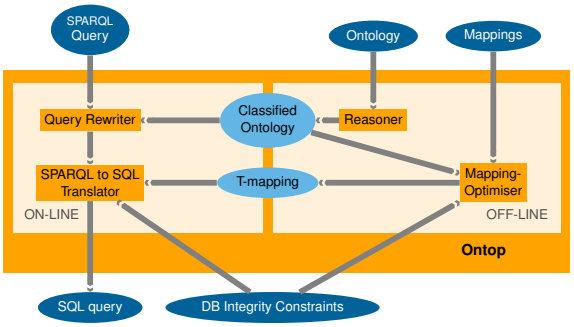


Fig. 4. The *Ontop* workflow

4. Answering SPARQL Queries

Ontop answers end-user's SPARQL queries by rewriting them into SQL queries and delegating execution of the SQL queries to the data sources. With this approach there is no need to apply rules to the data sources to materialize all the facts entailed by the ontology. The workflow of *Ontop* can be divided into an off-line and online stages and is illustrated in Fig. 4. The most critical task during start-up (the off-line stage) is generating the so-called T-mappings [49] by compiling the ontology into the mappings. During query execution (the online stage), *Ontop* transforms an input SPARQL queries into an optimized SQL query using the T-mappings and database integrity constraints. We now explain each of the two stages.

4.1. 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: (1) ontology classification, (2) T-mapping construction, and (3) T-mapping optimization. In the implementation of *Ontop*, however, the last two phases are performed simultaneously.

In Phase 1, the ontology is loaded through OWL API and is classified using the built-in OWL 2 QL reasoner. The resulting complete hierarchy of properties and classes is stored in memory as a directed acyclic graph. For example, in the ontology in Ex-

ample 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 [45] (a similar procedure was later described in [37]).

In Phase 2, T-mappings are constructed by compiling the complete class and property hierarchies into the mappings [49,51]. For example, consider concept `:Neoplasm` in Example 2.1. Although it has no rules in the mappings defined by the user, the two class inclusions derived above give rise to the following rules in the T-mapping:

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

Finally, in Phase 3, the T-mappings are optimized by using disjunction (`OR`) and interval expressions in SQL and by applying the semantic query optimization (SQO) techniques (which will be described in Section 4.2.2). For instance, using disjunction, *Ontop* transforms the two rules above into a single rule

```
:db1/neoplasm/{pid} a :Neoplasm .
    ← SELECT pid 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 the online stage of query processing. On the other hand, the resulting T-mappings define all the triples in the virtual RDF graph that includes all the inferences due to the ontology (under the entailment regime). Thus, during the online stage, the T-mappings are used directly for the translation of individual triple patterns in SPARQL queries into SQL.

4.2. Online Stage: Query Answering

The online stage takes a SPARQL query and translates it into SQL by using the 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 SQO techniques [16,33]. We distinguish three phases in the query answering process. (1) The SPARQL query is translated into SQL using T-mappings. (2) The resulting SQL query is optimized for efficient execution by the database engine. (3) 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 the necessary RDF terms. Note, however, that Phases 1 and 2 are handled together in the implementation of *Ontop* and we separate them here only for the sake of clarity. We elaborate now on the three phases of query answering.

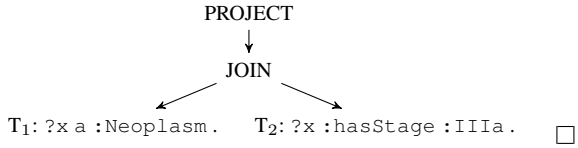
4.2.1. From SPARQL to SQL

Ontop internally represents the SPARQL query as a tree of the 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 4.1. Consider the fragment of the query in Example 2.3 that retrieves all tumors of stage IIIa:

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

(Note that triple pattern `?tumor a :Neoplasm` was redundant in Example 2.3: indeed, *Ontop* can infer it from `?p :hasNeoplasm ?tumor` because the range of `:hasNeoplasm` is `:Neoplasm`. On the other hand, the users are not expected to perform inferences and, in fact, often include such redundant triples.) The above query is represented by the following tree:



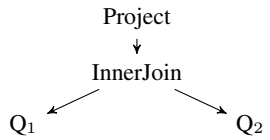
□

Next we explain how to produce the SQL expression from a SPARQL query using 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 of nodes in the tree of query Q 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, which is a triple pattern of the form (s, p, o), by the union of the SQL queries defining its

predicate in the T-mapping (lines 4–5). In this step, the algorithm implicitly considers two cases: *(i)* when p is an object or data property such as `:hasStage` or `:hasName`, or *(ii)* when p is `a` (`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 (JOIN, OPTIONAL, UNION, FILTER, and PROJECT) are translated into the corresponding SQL operators (InnerJoin, LeftJoin, Union, Filter, and Project, respectively). Once the root is translated, the process is finished and the resulting SQL expression is returned.

Example 4.2. *Ontop* translates the SPARQL query in Example 4.1 into an SQL query of the following structure (see Fig. 5a):



The leaves, Q_1 and Q_2, are the SQL definitions of the concept `:Neoplasm` and property `:hasStage`, respectively, in the T-mapping rules constructed during the off-line stage (see Section 4.1). Observe that without the T-mapping optimizations in the off-line stage, the resulting SQL would contain a union in place of Q_1, which would increase the complexity of the SQL query and so, would have a negative effect on the query evaluation time. □

Algorithm 1. Translating SPARQL into SQL
Input: SPARQL query Q, T-mappings $\mathcal{M}$
Output: SQL expression

```
1: S ← list of nodes in Q in a bottom-up topological order
2: sql ← empty map from nodes to SQL expressions
3: for node n ∈ S do
4:     if n is triple pattern then                ▷ translating leaves
5:         sql[n] ← replace-Tmap-def(n, M)
6:     else                                       ▷ translating non-leaf nodes
7:         if n = JOIN(n1,n2) then
8:             sql[n] ← InnerJoin(sql[n1], sql[n2])
9:         else if n = OPTIONAL(n1, n2, e) then
10:            sql[n] ← LeftJoin(sql[n1], sql[n2], e)
11:        else if n = UNION(n1, n2) then
12:            sql[n] ← Union(sql[n1], sql[n2])
13:        else if n = FILTER(n1, e) then
14:            sql[n] ← Filter(sql[n1], e)
15:        else if n = PROJECT(n1, p) then
16:            sql[n] ← Project(sql[n1], p)
17:        end if
18:    end if
19: end for
20: return sql[S.last()]
```

```
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 AND type = false) 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
        AND T2.type = false) 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 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 AND stage = 4
```

(d) SQL query after the second structural optimization

Fig. 5. Example of SQL translation and optimization

For the sake of simplicity we do not describe the translation of filter expressions and OPTIONAL (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 [53,35].

4.2.2. Optimizing the generated SQL queries

The generated SQL queries can already be executed by the database engine but they are inefficient: they often contain subqueries, redundant self-joins, and joins over complex expressions such as string concatenations (the latter, for instance, prevent the database engine from using indexes). *Ontop* employs a number of

structural and semantic optimizations to simplify and improve performance of produced SQL queries.

Structural Optimizations. *Ontop* applies three main structural optimizations: *(i)* pushing the joins inside the unions, *(ii)* pushing the functions as high as possible in the query tree, and *(iii)* eliminating sub-queries. Returning to the running example, the SQL query obtained by these optimizations is shown in Fig. 5b: optimizations *(ii)* and *(iii)* 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 area of SQO [16,33]. In general, SQO refers to the semantic analysis of SQL queries and use of database integrity constraints, such as primary and foreign keys, to reduce the size and complexity of the query, e.g., by removing redundant self-joins, and detecting unsatisfiable or trivially satisfiable conditions. In our running example, SQO eliminates the self-join, which is redundant because `pid` is the primary key of `tbl_patient`; it also simplifies the WHERE clause because the condition `(type = false OR type = true)` is implied by `(type = false)`. The resulting SQL query is shown in Fig. 5c. Observe that it has a sub-query, `Q`, that could not be eliminated before but, after the SQO step, structural optimization *(iii)* can be applied again to eliminate sub-query `Q` and obtain an even simpler SQL query, which shown in Fig. 5d.

Observe that these optimizations interact with and complement each other. The optimization step is critical [36] and nontrivial. This simple example illustrates the basic principles. The translation of complex queries is more involved and takes account of the gap between the SQL and SPARQL semantics. The interested reader is referred to [35,53].

4.2.3. Executing Query over the Database

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

As the final step, *Ontop* sends the generated 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.

4.3. 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, and NPD) and settings (e.g., various database engines, number of clients, dataset size) [35, 54,36,51]. 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. On the one hand, this is in line with the well-known theoretical results on the price of OBDA: the transformation of the query suffers an exponential blow-up in the worst case [23]. On the other hand, on the standard query rewriting benchmarks (LUBM, Adolena, etc.), the tree-witness query rewriting algorithm implemented in *Ontop* produces rewritings shorter and simpler than all other tools; moreover, it is also faster [51]. As a consequence, *Ontop* can efficiently perform ontological inferences in the virtual RDF graph mode without any need for materialization: on IMDb, for example, it is competitive with such materialization-based systems as OWLIM (GraphDB) [51] and, on LUBM, it outperforms reasoner-based systems, especially on large data instances [35]. In benchmarks like BSBM and FishMark, where the number of mappings is small and the datasets range from 25 to 200 million triples, *Ontop* outperforms its competitors (D2RQ, OWLIM, Stardog, Virtuoso) by orders of magnitude [54]. This performance is the result of *(i)* the fast SPARQL-to-SQL translation (4–15ms); *(ii)* the efficient optimization of the SQL; and *(iii)* the well-known efficiency of relational databases. 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 [36], which reveals the strengths and pitfalls of OBDA. It is based on the original NPD FactPages ontology, mappings and queries [57]. The NPD FactPages original data is published by the Norwegian Petroleum Directorate (NPD) and the query

set was obtained by interviewing users of NPD FactPages.[8] This setting thus provides a realistic account of the information needs in the modeled scenario. The ontology and mappings contain thousands of axioms and rules and our benchmark comes with a dataset of up to 4 billion triples, which were obtained from the original NPD FactPages dataset. The results comparing *Ontop* and Stardog on the NPD Benchmark show [36] that our approach is scalable but more work is needed to optimize the generated SQL queries. Indeed, while the optimizations currently implemented in *Ontop* result in efficient SQL translations for most of the queries, some cases are still challenging. *Ontop* outperforms Stardog whenever the SPARQL query is translated into a small SQL query. But in those few cases when the generated SQL queries are large unions of subqueries, Stardog still outperforms *Ontop*. We are currently working on various techniques for tackling this issue.

5. Industrial Applications

Adoption of *Ontop* by the community has been growing steadily in the past six years. In 2015, the *Ontop* bundle was downloaded more than 1800 times[9], the webpage got 12K hits, and the mailing list more than 200 topics. Since November 2015, IBM has been contributing to the *Ontop* code and using it, e.g., for data integration [38]. Also, in November 2015, Complexible Inc. integrated *Ontop* code into Stardog v4.

Ontop is actively used in academia.[10] For example, the EPNet project[11] relies on *Ontop* to improve access for scholars to historical and cultural data on food production and commercial trade system during the Roman Empire from several data sources [14]. Also, *Ontop* is used in Semantic Mediator [7], for accessing electronic health records [46], and for querying temporal and streaming data in OBDA [41].

Ontop is the core component of the *Optique* Platform, which is developed in the EU large scale integrating project *Optique* [22] and commercialized by fluid Operations (fluidOps)[12]. In the rest of this section, we describe the use cases of the two major industrial partners in the *Optique* project, namely Statoil [30] and Siemens [41], and the role *Ontop* plays there.

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 require access to a number of large databases on a daily basis. One of them, for example, the Exploration and Production Data Store (EPDS), comprises over 1500 SQL tables with information on historical exploration data (e.g., layers of rocks, porosity), production logs, and maps, among others. It also contains business information such as license areas and companies. The schema is organized in such a way that the direct data access by engineers (and geologists in particular) often becomes challenging or even impossible. The main problem lies not only in the size of the schema and the data but also in the complex structure of this legacy database. The solution currently adopted by Statoil relies on tools that provide domain experts with a few different predefined queries. However, these pre-defined queries are often too specific, or too general, and cannot be easily combined to obtain the desired results.

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 a number of 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 generated every day, resulting in a total of 100 TB of timestamped data. One of the main tasks for service engineers monitoring these devices is to promptly solve issues detected by gathering the relevant sensor data and analyzing it.

The data gathering phase is often the bottleneck of the process because it takes about 80% of the engineers' time. This is partly due to the complexity and quantity of the data. Ideally, the engineers should be able to access the data directly, by creating and combining queries in an intuitive way that is compatible with their knowledge. However, the data is often organized to better serve the applications rather than the domain experts.

In scenarios such as at Statoil and Siemens, the OBDA approach to solving these problems consists in enriching the legacy databases with an ontological layer that uses a terminology familiar to the engineers.

[8] `http://factpages.npd.no/factpages`
[9] Reported by SourceForge for the period May–December, 2015.
[10] `https://github.com/ontop/ontop/wiki/UseCases`
[11] `http://www.roman-ep.net`
[12] `http://www.fluidops.com/en`

The ontology helps the engineers in formulating their own queries autonomously using the domain vocabulary [22,31], thus effectively mediating between the engineers and the data. The role of *Ontop* (and *Optique*) is to make the OBDA approach feasible, by automating the process of translating the queries that the engineers pose over the ontology into queries over the legacy databases that can be executed efficiently.

6. Related SPARQL Query Answering Systems

We now briefly review the most popular SPARQL query answering systems, which can be categorized into two major types: triplestores and OBDA systems. Table 1 summarizes their main features.

Triplestores provide a flexible generic logical model for storing 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 is required to transfer data between these external sources and the triplestore. The ETL process can be expensive, especially when data sources are frequently updated.

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 (thus, the additional ETL process is not 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 certain drawbacks: inferences can be costly in terms of both time and space; moreover, updates and deletions of triples require additional bookkeeping for incremental reasoning. Also, this approach cannot be adopted without sacrificing completeness of query answering when the ontology language (such as OWL 2 QL) is capable of inferring new individuals in the data.

In contrast to triplestores, the most common strategy for OBDA systems is query rewriting, and so OWL 2 QL is the OWL 2 profile most suitable in this setting. 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 of the two types.

6.1. Triplestores

Virtuoso Universal Server[13] is a hybrid system 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 used mostly as a triplestore. It supports SPARQL 1.1 and, in this mode, it offers some backward- (by default) and forward-chaining capabilities for limited subsets of RDFS and OWL. When Virtuoso is used as a regular DBMS, it can be turned into an OBDA system by setting up mappings in its own mapping language. However, its OBDA mode has several limitations: no reasoning capabilities are available and only a small fragment of R2RML is supported. Virtuoso can be accessed through the Sesame and Jena APIs.

GraphDB,[14] previously known as OWLIM [5], is a commercial triplestore developed by Ontotext. It fully supports SPARQL 1.1. OWL reasoning is based on the forward-chaining materialization approach. This strategy naturally fits with the OWL 2 RL profile but is incomplete for OWL 2 QL [4]. GraphDB is accessible through the Sesame API.

Stardog[15] is a commercial triplestore developed by Complexible Inc[16]. 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 the latter is guaranteed only for schema reasoning). Stardog avoids eager materialization and its reasoning engine is partly based on query rewriting (in fact, the reasoning level can be chosen by the user at query time). Stardog can be accessed through the Sesame API. Since version 4 released in November 2015, Stardog has integrated *Ontop* code to support SPARQL queries over virtual RDF graphs. Therefore, it can now be classified also as an OBDA system.

RDFox[17] is an in-memory triplestore developed at the University of Oxford. It implements a novel shared-memory parallel Datalog reasoning algorithm and sup-

[13] http://virtuoso.openlinksw.com
[14] http://ontotext.com/products/graphdb
[15] http://stardog.com
[16] http://complexible.com
[17] http://www.cs.ox.ac.uk/isg/tools/RDFox

Table 1
Feature matrix of SPARQL query answering systems

Type	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*	Native, R2RML	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

(* indicates limited support)

ports OWL 2 RL reasoning by materialization [40]. The system is a cross-platform software written in C++ and comes with a Java wrapper supporting OWL API.

6.2. OBDA Systems

D2RQ[18] is one of the pioneering OBDA systems, developed at the Free University of Berlin and DERI. This query rewriting system implements some query optimizations but these have often been reported as insufficient: for instance, the generated SQL queries can contain an excessive number of joins [44]. 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[19] 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 conjunctive queries. Mastro is available only for demonstration, testing, and evaluation purposes.

Ultrawrap[20] is an OBDA system commercialized by Capsenta. It was recently extended to support inference over an extension of RDFS with inverse and transitive properties [56]. Ultrawrap uses an analogue of 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,[21] 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 self-join elimination [44]. However, it has no inference capability.

7. A Retrospective

Ontop has its roots in our early work on QuOnto and Mastro [1,10]. QuOnto is a reasoner for the description logic *DL-Lite* with plain conjunctive query (CQ) answering and Mastro is its extension with GAV (global as view) mappings for relational databases [42] (both systems are maintained by the Sapienza University of Rome). Our work enabled the use of these systems through the ontology editor Protégé 3 [43] and the DIG reasoner API [48].

Using these tools we interacted with third parties to develop several OBDA applications [13,10,29,52,47] (for a full list, see [47]). In the process we tested both the performance of the state-of-the-art query rewriting techniques and the feasibility of this technology for data integration and data access. We obtained insights on techniques and optimizations on the one hand, and on APIs and required features on the other hand. These two strands of development characterized our work from then on. We now briefly elaborate on them.

Reasoning, Optimization, and Performance. The main issue initially was the large number of CQs produced by the rewriting algorithm (PerfectRef [11]) implemented in QuOnto, which often returned hun-

[18] http://d2rq.org
[19] http://www.dis.uniroma1.it/~mastro
[20] http://capsenta.com
[21] https://github.com/oeg-upm/morph-rdb

dreds of thousands of CQs (even for simple ontologies and mappings). And although database systems do perform very well in general, commercial and non-commercial engines alike have problems with large generated queries. To deal with the issue, we extended PerfectRef by a Semantic Query Optimization (SQO) component, which removes redundant CQs and eliminates redundant self-joins using database integrity constraints (foreign and primary keys) [47].

The work in this direction materialized in the first version of *Ontop* (2010), which was called Quest (the name now refers only to the query processing engine). Quest can work in *(i)* the virtual mode that supports virtual RDF graphs via mappings, and *(ii)* the triplestore mode that stores RDF triples directly in a relational database. We developed the theory of T-mappings to improve performance in the virtual mode [49,35] (cf. Section 4.1) and the Semantic Index for the triplestore mode [50]. Then, the tree-witness query rewriting algorithm [32] replaced PerfectRef to drastically reduce the size of rewritings and take advantage of T-mappings and the Semantic Index. We also observed [47] that the generic database-centric SQO is insufficient in the OBDA setting and proposed novel techniques: e.g., simplification of join conditions by de-IRIing, cf. Section 4.2.2.

More recent lines of research on *Ontop* include *(i)* the formalization of SPARQL in the context of OBDA [53,35], *(ii)* the OWL 2 QL entailment regime [35], *(iii)* the SWRL rule language with a limited form of recursion handled by SQL Common Table Expressions [61], *(iv)* `owl:sameAs` for cross-linked datasets [12], and *(v)* expressive ontologies beyond OWL 2 QL by rewriting and approximation with the help of the mapping layer [6].

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 the level of compliance with the RDF, RDFS, OWL 2 QL, SPARQL and R2RML standards. To support the OWL community, we include the OWL API and Protégé 4 (and more recently Protégé 5) interfaces for *Ontop*. To support the Linked Data community, we provide the Sesame API interface for *Ontop*, as well as an HTTP SPARQL endpoint.

Ontop was initially released under a non-commercial use license before adopting the permissive Apache 2.0 license in 2013. The project is now hosted in GitHub so that anybody can download it and contribute to the code. 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 (∼2000) are automatically run with Travis-CI whenever new changes are pushed to GitHub.

8. Conclusion

We presented *Ontop*, a mature open-source OBDA system, which 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 numerous 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 develop *Ontop* in the following directions.

- In order to further improve performance, we will investigate data-dependent optimizations.
- We plan to support larger fragments of SPARQL (e.g., aggregation, negation, and path queries) and R2RML (e.g., named graphs).
- For end-users, we will improve the GUI and extend utilities to make *Ontop* even more user-friendly.
- We plan to go beyond relational databases and support other kinds of data sources (e.g., graph and document 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. AAAI Press, 2005.

[2] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. ANAPSID: An adaptive query processing engine for SPARQL endpoints. In *Proc. of the 10th Int. Semantic Web Conf. (ISWC)*, volume 7031 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2011. doi:10.1007/978-3-642-25073-6_2.

[3] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The *DL-Lite* family and relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009. doi:10.1613/jair.2820.

[4] 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*. CEUR-WS.org, 2011.

[5] 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. doi:10.3233/SW-2011-0026.

[6] Elena Botoeva, Diego Calvanese, Valerio Santarelli, Domenico Fabio Savo, Alessandro Solimando, and Guohui Xiao. Beyond OWL 2 QL in OBDA: Rewritings and approximations. In *Proc. of the 30th AAAI Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 2016.

[7] 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. doi:10.1145/2628194.2628218.

[8] 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/`.

[9] 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. doi:10.1007/3-540-48005-6_7.

[10] 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.

[11] 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. doi:10.1007/s10817-007-9078-x.

[12] Diego Calvanese, Martin Giese, Dag Hovland, and Martin Rezk. Ontology-based integration of cross-linked datasets. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC)*, volume 9366 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 2015. doi:10.1007/978-3-319-25007-6_12.

[13] 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. ACM Press, 2010. doi:10.1145/1774088.1774384.

[14] Diego Calvanese, Alessandro Mosca, Jose Remesal, Martin Rezk, and Guillem Rull. A 'historical case' of ontology-based data access. In *Proc. of Digital Heritage 2015 (DH 2015)*. IEEE Computer Society Press, 2015.

[15] Tiziana Catarci, Paolo Dongilli, Tania Di Mascio, Enrico Franconi, Giuseppe Santucci, and Sergio Tessaris. An ontology based visual tool for query formulation support. In *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI)*, pages 308–312. IOS Press, 2004. doi:10.1007/978-3-540-39962-9_15.

[16] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Trans. on Database Systems*, 15(2):162–207, 1990. doi:10.1145/78922.78924.

[17] 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/`.

[18] AnHai Doan, Pedro Domingos, and Alon Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003. doi:10.1023/A:1021765902788.

[19] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12(4):303–319, 2003. doi:10.1007/s00778-003-0104-2.

[20] Ronald Fagin, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*, volume 5600 of *Lecture Notes in Computer Science*, pages 198–236, 2009. doi:10.1007/978-3-642-02463-4_12.

[21] Luciano Frontino de Medeiros, Freddy Priyatna, and Oscar Corcho. MIRROR: Automatic R2RML mapping generation from relational databases. In *Proc. of the 15th Int. Conf. on Web Engineering (ICWE)*, volume 9114 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2015. doi:10.1007/978-3-319-19890-3_21.

[22] Martin Giese, Ahmet Soylu, Guillermo Vega-Gorgojo, Arild Waaler, Peter Haase, Ernesto Jiménez-Ruiz, Davide Lanti, Martín Rezk, Guohui Xiao, Özgür L. Özçep, and Riccardo Rosati. Optique – zooming in on big data access. *IEEE Computer*, 48(3):60–67, 2015. doi:10.1109/MC.2015.82.

[23] 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. doi:10.1016/j.artint.2014.04.004.

[24] Florian Haag, Steffen Lohmann, Stephan Siek, and Thomas Ertl. QueryVOWL: A visual query notation for linked data. In *Proc. of the 3rd Int. Workshop on Human Semantic Web Interaction (HSWI). ESWC 2015 Satellite Events*, volume 9341 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2015. doi:10.1007/978-3-319-25639-9_51.

[25] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web J.*, 2(1):11–21, 2011. doi:10.3233/SW-2011-0025.

[26] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Yujiao Zhou, and Ian Horrocks. Large-scale interactive ontology matching: Algorithms and implementation. In *Proc. of the 20th European Conf. on Artificial Intelligence (ECAI)*, volume 242, pages 444–449. IOS Press, 2012.

[27] Ernesto Jimenez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G.

Skjæveland, Evgenij Thorstensen, and Jose Mora. BootOX: Practical mapping of RDBs to OWL 2. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC)*, volume 9367 of *Lecture Notes in Computer Science*, pages 113–132. Springer, 2015. doi:10.1007/978-3-319-25010-6_7.

[28] 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`.

[29] 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*. CEUR-WS.org, 2008.

[30] Evgeny Kharlamov, Dag Hovland, Ernesto Jimenez-Ruiz, Davide Lanti, Hallstein Lie, Christoph Pinkel, Martin Rezk, Martin G. Skjæveland, Evgenij Thorstensen, Guohui Xiao, Dmitriy Zheleznyakov, and Ian Horrocks. Ontology based access to exploration data at Statoil. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC)*, volume 9367 of *Lecture Notes in Computer Science*, pages 93–112. Springer, 2015. doi:10.1007/978-3-319-25010-6_6.

[31] 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. doi:10.1007/978-3-319-11964-9_38.

[32] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pages 275–285. AAAI Press, 2012.

[33] 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.

[34] Craig A. Knoblock, Pedro A. Szekely, José Luis Ambite, Aman Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, Mohsen Taheriyan, and Parag Mallick. Semi-automatically mapping structured sources into the Semantic Web. In *Proc. of the 9th Extended Semantic Web Conf. (ESWC)*, volume 7295 of *Lecture Notes in Computer Science*, pages 375–390. Springer, 2012. doi:10.1007/978-3-642-30284-8_32.

[35] 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. doi:10.1007/978-3-319-11964-9_35.

[36] 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)*, pages 617–628. OpenProceedings.org, 2015. doi:10.5441/002/edbt.2015.62.

[37] 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. doi:978-3-642-38288-8_22.

[38] Vanessa Lopez, Martin Stephenson, Spyros Kotoulas, and Pierpaolo Tommasi. Data access linking and integration with DALI: building a safety net for an ocean of city data. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC), Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 186–202. Springer, 2015. doi:10.1007/978-3-319-25010-6_11.

[39] 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/`.

[40] 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.

[41] 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. doi:10.1007/978-3-319-10587-1_7.

[42] 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. doi:10.1007/978-3-540-77688-8_5.

[43] 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.

[44] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. Formalisation 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. ACM, 2014. doi:10.1145/2566486.2567981.

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

[46] 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. of Medical Informatics*, 83(10):768–778, 2014. doi:10.1016/j.ijmedinf.2014.06.002.

[47] Mariano Rodriguez-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. doi:10.1007/978-3-642-39784-4_5.

[48] 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*, 2008.

[49] 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*, 2011.

[50] Mariano Rodriguez-Muro and Diego Calvanese. High perfor-

mance query answering over *DL-Lite* ontologies. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pages 308–318. AAAI Press, 2012.

[51] 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. doi:10.1007/978-3-642-41335-3_35.

[52] Mariano Rodriguez-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)*, pages 286–289. IEEE Computer Society Press, 2008. doi:10.1109/ICDEW.2008.4498333.

[53] Mariano Rodriguez-Muro and Martin Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *J. of Web Semantics*, 33:141–169, 2015. doi:10.1016/j.websem.2015.03.001.

[54] 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*, pages 94–100, 2013.

[55] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: A federation layer for distributed query processing on linked open data. In *Proc. of the 8th Extended Semantic Web Conf. (ESWC)*, volume 6644 of *Lecture Notes in Computer Science*, pages 481–486. Springer, 2011. doi:10.1007/978-3-642-21064-8_39.

[56] 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. doi:10.1007/978-3-319-11964-9_34.

[57] 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.

[58] Ahmet Soylu, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jimenez-Ruiz, Martin Giese, and Ian Horrocks. Ontology-based visual query formulation: An industry experience. In *Proc. of the 11th Int. Symposium on Visual Computing (ISVC)*, volume 9474 of *Lecture Notes in Computer Science*, pages 842–854. Springer, 2015. doi:10.1007/978-3-319-27857-5_75.

[59] 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.

[60] Daniel Tunkelang. *Faceted Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009. doi:10.2200/S00190ED1V01Y200904ICR005.

[61] 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. doi:10.1007/978-3-319-11113-1_11.