Querying Ontologies: Retrieving Knowledge from Semantic Web Documents

Gerasimos Tzoganis, Dimitrios Koutsomitropoulos and Theodore S. Papatheodorou

Computer Engineering and Informatics Dpt, School of Engineering, University of Patras
{tzoganis, kotsomit, ptheodor}@ceid.upatras.gr

Abstract

OWL, and mainly its most recent version OWL 2, is currently being established as the language used to formalize the information existing on the Web, in a way that this information becomes machine understandable and machine processible. OWL has become a W3C recommendation as an ontology language. An OWL document, or simply an ontology, is more capable of being processed and interpreted by algorithms that can be implemented by a computer (in accordance with the vision of the Semantic Web), than an ontology written in XML or in RDF. But there is still open the problem of knowledge retrieval in OWL 2 via querying an ontology. An approach to this issue is attempted in this paper. We aim to build a user friendly (the user being human or a machine) application for querying ontologies. The queries are written in SPARQL-DL, an extension of SPARQL, which is the W3C recommendation as a query language for RDF.

Keywords: SPARQL, SPARQL-DL, OWL, ontology, ontology querying, semantic web.

1. Introduction

The Semantic Web is an evolving development of the World Wide Web in which the semantics of information and services on the web are defined, making it possible for the web to understand and satisfy the requests of people and machines to use the web content. In order to accomplish this goal there is the need of a formal description of concepts, terms, and relationships. The Web Ontology Language OWL is the W3C recommendation for this. In this context, we also need to have the ability to query an OWL document (or simply an ontology) in order to retrieve knowledge. This knowledge can concern general “classes” of concepts and properties between these classes (that would be a TBox query) or “instances” of classes (ABox query). SPARQL is a query language for querying RDF documents, and it has become a standard by W3C. SPARQL-DL was introduced in [2] as a query language for OWL-DL (a computationally efficient sublanguage of OWL) ontologies and poses only a few restrictions regarding compatibility with and making queries to OWL 2 knowledge bases (OWL 2 is the current version of OWL).

In this paper we attempt to build a user friendly environment, where the user can easily load an existing ontology and submit a query, formed in SPARQL-DL. We make use of the built-in query engine of the reasoner Pellet. A reasoner is a collection of software that allows understanding an ontology and inferring about it. In order to interact with Pellet, we make use of the Jena API, which is an API for managing ontologies and also has a built-in SPARQL query engine, but does not support SPARQL-DL. Pellet's query engine handles ABox queries. All of this software, as well as our application is written in JAVA. There is currently no other API with SPARQL support, so we chose Jena for our application in order to follow the recommendations of W3C (OWL 2 and SPARQL) and make benefit of the advanced capabilities for query answering of Pellet.

We also attempt to investigate the capabilities that such an approach offers for knowledge retrieving, as well as the restriction it poses. We experiment with several example queries, and examine the effectiveness of our application and the correctness of the results.

2. Background

At this time, Pellet is the only piece of software that supports the SPARQL-DL format for querying OWL DL and OWL 2 ontologies. Pellet can be accessed via three different APIs: the internal Pellet API, the Manchester OWL API and the Jena API. The internal API is designed for efficiency but is yet missing features and has low usability. The Manchester OWL API is an OWL-centric API with a wide range of features for managing ontologies but does not support SPARQL. Jena is a widespread and stable API but lacks specific OWL 2 support. However, it is the only one supporting SPARQL and it also has a built in SPARQL query engine as mentioned before. Therefore, it seems that the only way to retrieve knowledge from OWL 2 documents and stay in accordance with the SPARQL standard as well is by utilizing Pellet's query engine through the Jena API.

A complete overview of OWL is available in [1]. In a synopsis, we would describe an OWL-DL vocabulary as a set of classes, object properties, datatype properties, annotation properties, individuals, datatypes, and literals. There are also axioms including class axioms (subclass of, equivalentTo, …), property axioms (subPropertyOf, inverseOf, …), property characteristics (functional, transitive, …) and individual assertions (sameAs, differentFrom). There are various syntaxes for OWL, but the primary syntax follows RDF's XML based syntax. It is finally worth mentioning that the predecessor of OWL is DAML+OIL, which is the first effort to create an ontology markup more expressive than RDF or XML.

SPARQL is the standard RDF query language and its complete overview can be found in [6]. The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern, which corresponds to a RDF triple (subject

– predicate - object triple), but with a possibility of a variable instead of an RDF term in one of these three positions. Several triple patterns combined give a basic graph pattern, and an exact match to a RDF graph is needed to fulfill the pattern, and thus answer the query.

SPARQL, just like other RDF-based query languages (RDQL, SeRQL, SPARQL) are quite hard to give a semantics w.r.t OWL-DL and their expressivity is more powerful than OWL-DL reasoners can cope with. On the other hand, query languages based on Description Logics (DIG, nRQL) have clearer semantics but are not powerful enough. SPARQL-DL was introduced in [2] as a substantial subset of SPARQL with clear OWL-DL based semantics and claimed to be significantly more expressive than existing DL query languages and still able to be implemented on top of existing OWL-DL reasoners like Pellet. SPARQL-DL's designers took advantage of the fact that SPARQL is designed so that its graph matching semantics are extensible to any arbitrary entailment regime and the fact that there is a well-defined entailment relation between valid OWL-DL ontologies, as stated in [4]. They first define an abstract syntax for SPARQL-DL queries, define its semantics and then provide a transformation from the abstract syntax to the triple patterns of SPARQL aiming to define an entailment regime for SPARQL based on OWL-DL which can be implemented in a straightforward way. On Table 1 we show examples of SPARQL and SPARQL-DL queries asked against an OWL-DL ontology so that the differences become more clear.

Table 1. *A SPARQL and a SPARQL-DL query*

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ub:   <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X ?C            WHERE {
                                  ?X rdf:type ?C .
                                  ?C rdfs:subClassOf ub:Employee .
                              }
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ub:   <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX sparqldl: <http://pellet.owldl.com/ns/sdle#>
SELECT ?X ?C            WHERE {
                        ?X sparqldl:directType ?C .
                        ?C rdfs:subClassOf ub:Employee .
}
```

We use the namespace mechanism to define prefixes for namespaces in order to make queries more readable. Both queries return the same values. The difference between the two queries is "sparqldl:directType", which is necessary because of the OWL-DL semantics, according to which rdfs:subClassOf is a transitive property and the query atom ?X rdf:type ?C would also match all subclasses of a class C.

3. Implementation

3.1 Technologies used

Our application is a web application that runs in any major web browser. It was developed with Google Web Toolkit [7] and uses gwt-ext [11] widget library. Google Web Toolkit (GWT) allows the developer to write an AJAX front-end in the JAVA programming language, and then cross-compiles this JAVA source code into JavaScript. It can easily be combined with several JAVA development tools like Eclipse and offers an easy to handle development environment that also makes it easy to debug the JAVA code. Eventually it produces highly optimized JavaScript code by performing various optimization techniques that additionally runs automatically to any of the major web-browsers. Finally, GWT offers a really simple Remote Procedure Call mechanism. We took advantage of that when putting the actual implementation of our application on the server side, leaving the application's interface for the client side. Thus, we put on the server side the JAVA code that loads an ontology, parses and processes a SPARQL query utilizing, as foresaid, Jena API in combination with Pellet's classes.

GWT provides several widgets to build a really user-friendly interface, but we also used to a large extent the gwt-ext widget library. Gwt-ext makes it easy to combine GWT's functionality and the widgets available in the ExtJS javascript library. It is also possible to use development tools like eclipse with gwt-ext. Most of the widgets of our application's interface are gwt-ext widgets.

3.2 Functionality

The functionality of our application becomes available through its interface. The interface is shown at Figure 1. It is quite simple and minimalistic. It consists of two main parts. The first one takes the user input, i.e. the ontology and the query. The second (titled "results") displays the answer to the user' s query. Below we present the application' s interface and its functionality in more detail.

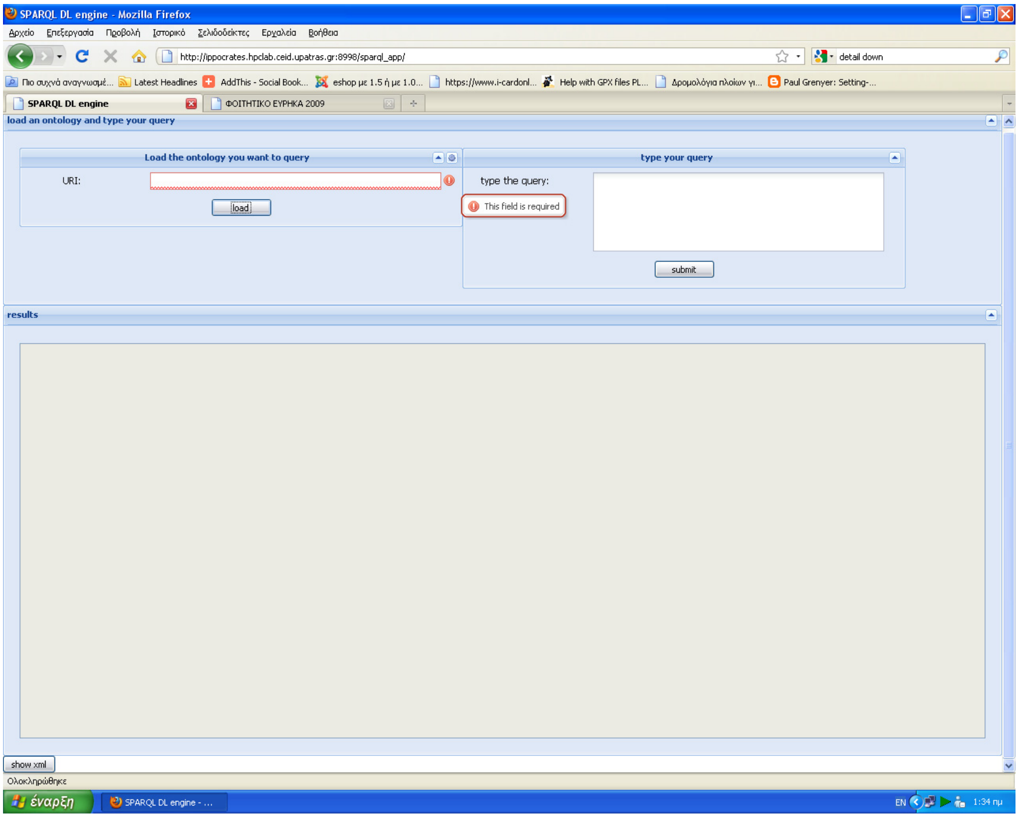
SPARQL DL engine - Mozilla Firefox
http://ippocrates.hpclab.ceid.upatras.gr:8998/sparql_app/
load an ontology and type your query
Load the ontology you want to query
URI:
load
type your query
type the query:
This field is required
submit
results
show xml

Figure 1: *The Application' s Interface*

The first part of the interface is a horizontal panel titled "load an ontology and type your query" that consists of two simple forms. The first form,titled "load the ontology you want to query", accepts from the user the URI of the ontology the user wishes to query and consists of a textfield where the user types the ontology URI and a load button. When this button gets pressed, if the URI provided by the user is valid, the ontology gets loaded. A success message is displayed on the second part of the application. The field of the URI is required, which is checked by AJAX code and the user gets notified when the field remains blank. This is also shown at Figure 1. In such a case, the textfield gets highlighted and an exclamation mark appears next to it. Finally, on the upper right corner of this form, the user will find a tool so that he can load and try querying one of some bookmark ontologies.

The second form is titled "type your query" and consists of a text area and a button. The user writes the query in the text area in SPARQL form, and then hits the button so that the query will be submitted to the ontology that was previously loaded. Typing the query is also required, and this is checked in similar fashion. If the query is not valid, an appropriate message is displayed at the second part of the interface. Otherwise, this part displays the results of the query.

The second part is a panel named “results” that contains a read-only text area and a button under it. Apart from displaying some useful log information to the user (like “ontology loaded”, etc.), mainly it serves the purpose of displaying the results of the query. These results appear by default in the form of URIs. But the button gives the user the opportunity to get the results in XML form. If the button is pressed, the text area contains the results in XML form.

3.3 Processing the query

In this section we describe in more detail how our application parses, processes and answers a SPARQL-DL query. We have used to some extent the code bundled with Pellet’s tutorial [3]. Note that the query must only be in SPARQL form and not in abstract form, for example. In any other case a parsing error will occur.

Initially, we have to load the ontology. When the “load ontology” button is pressed, we create a Jena ontology model backed by Pellet, using an instance of *ModelFactory* of Jena’s package *com.hp.hpl.jena.rdf.model*, and then read (using an instance of *Model* of the same package) the data from the ontology file specified by the URI the user has given, into the ontology model. Because all of the methods for loading an ontology and processing the query exist on the server, the URI is passed to the server side method for loading an ontology as a String.

After the application loads the ontology, the user can submit his query, which also passes to the server side methods as a String. There are two methods for processing a query, one that returns the results in a form of a URI set and one that returns the results in xml form. The main steps for processing the query are identical in both methods: First, the query String is read into a query object using an instance of Jena’s *com.hp.hpl.jena.query.QueryFactory*. Then, we use *com.clarkparsia.pellet.sparqldl.jena.SparqlDLExecutionFactory* to create a SPARQL-DL query execution for the given query and ontology model. We then execute the query using an instance of *QueryExecution* of Jena’s same package. Finally we call an instance of *ResultSetFormatter* of the same package to return the result to client side either as a set of URIs or as an XML tree.

Now we will describe from a high level of abstraction the way Pellet evaluates a query. This is described in more detail in [4]. First Pellet’s SPARQL-DL engine (built upon existing Pellet’s query engine) performs preprocessing of the query. This includes removing redundant atoms, and getting rid of “Same As” atoms with bnodes (a blank node or anonymous resource or bnode is a node in an RDF graph which is neither identified by a URI nor is it a literal). The next preprocessing step is removing trivially satisfied atoms, for example SubClassOf(C, C) and SubClassOf(C, owl:Thing). A final preprocessing step is to split the query into connected components in order to avoid computing cross-products of their results. After these preprocessing steps, the SPARQL-DL query gets partitioned into two parts. The first part (Q_c) contains all Type and PropertyValue atoms from Q, and intuitively represents an ABox query. The other part (Q_s) contains all other atoms from Q. Q_s gets augmented with trivial atoms so that it will contain all class and property variables mentioned in the query. All atoms in Q_s are evaluated, generating a collection R of result bindings. Each B that belongs in R is then applied to Q_c, resulting in query Q_c. This query is then evaluated using the underlying conjunctive ABox query engine.

4. Experiments

In this chapter we try some example queries in order to investigate and point out the capabilities and the shortcomings of SPARQL-DL. We will use an ontology describing the domain of a university. This ontology is found on [8] and we used the OWL 2 Validator [9] to make sure it is a valid OWL 2 ontology.

First of all, we want SPARQL-DL to capture the semantics of OWL 2. We execute a simple query that includes an OWL-DL atom. We use the simple query shown on Table 2.

Table 2. *Simple DL query*

```
# Retrieve all transitive properties

PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
SELECT ?X WHERE {
      ?X rdf:type owl:TransitiveProperty.
 }
```

This works with no problems, returning the URI of the property *subOrganizationOf.* Next we will pose a mixed ABox/Tbox query to see if Pellet's SPARQL-DL query engine handles well with it (this engine as foresaid is built on top of Pellet's ABox SPARQL query engine). We will use one of the queries available in [10]. This is an extension to the LUBM dataset made by the authors of [4]. The query is shown on Table 3. This query also contains a cycle through distinguished variables (?C). Pellet is supposed to be able to handle with cycles through distinguished variables and also perform several optimizations. Our application executed this query very fast and returned the correct results, confirming the above.

Finally, we will execute queries with undistinguished variables**.** Originally, Pellet's query engine would not support queries with cycles between undistinguished variables. It used the rolling-up technique to handle undistinguished variables (bnodes). A new, more effective and sophisticated technique was presented in [5].

Table 3. *Mixed ABox/TBox query*

```
# Give me all people that are members of the Department0 and
# tell me which kind of membership it is.

PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ub:   <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>

SELECT ?X ?C
WHERE {
      ?X rdf:type ub:Student .
      ?X rdf:type ?C .
      ?C rdfs:subClassOf ub:Employee .
}
```

Table 4. *Query with undistinguished variables*

```
# Give me all teaching graduate students (?X) that take a course (?Y)
# taught by their advisor.
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ub:   <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
SELECT ?X ?Y
WHERE {  ?X rdf:type ub:GraduateStudent .
         ?X ub:takesCourse ?Y.
          _:a ub:teacherOf ?Y.
        ?X ub:advisor _:a.
        ?X ub:teachingAssistantOf _:b.}
```

Table 5. *Query with cycle between undistinguished variables*

```
# For all people (?X) leading a department tell me at which #institution (?Z)
# they  have obtained their degrees.

PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ub:   <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>

SELECT ?X ?Z WHERE {
        ?X ub:headOf _:a .
        _:a rdf:type ub:Department .
        ?X ub:degreeFrom ?Z .
        }
```

We tried to test our application with the two sample queries proposed in this presentation, which are shown on Table 4 and Table 5. Both queries were answered fast, despite the second one containing a cycle through undistinguished variables. So we assume Pellet right now uses the newer technique to handle with bnodes.

5. Conclusions and Possible Extensions

Our application provides a handy tool so that one can retrieve knowledge from an OWL 2 ontology. Designed with gwt, the client side code (interface) runs faster and makes the procedure of querying a knowledge base sufficiently user friendly. By using SPARQL-DL as the query language, it complies with the W3C's recommendations and takes advantage of the advanced query capabilities of Pellet. SPARQL-DL proves to be a query language that can support the advanced expressivity of OWL 2. Pellet itself proves to be quite effective in supporting this expressivity. Thus, the whole application performs knowledge retrieval effectively.

As a future extension we would like our application to make the results of a query accessible over the web on a unique URI, so that the user can find the results there. Then, there could also be the possibility of an application using our interface, without manual intervention. It would be also useful supporting other syntaxes for writing SPARQL-DL queries, like the abstract form.

References

1. Deborah L. McGuinness, Frank van Harmelen: OWL Web Ontology Language Overview http://www.w3.org/TR/owl-features/ (2004).
2. Evren Sirin, Bijan Parsia. SPARQL-DL**:** SPARQL Query for OWL-DL, *3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.
3. Evren Sirin, Mike Smith: *A Programmer's Introduction to Pellet: How to Build Ontology-based Semantic Applications*
4. Petr Kremen, Evren Sirin. SPARQL-DL Implementation Experience, *4th OWL Experiences and Directions Workshop (OWLED-2008 DC)*, 2008.
5. Petr Kremen: On Queries with Undistinguished Variables, http://clarkparsia.com/weblog/2007/12/07/on-queries-with-undistinguished-variables/ (2007)

6. Prud’hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Working draft http://www.w3.org/TR/rdf-sparql-query/ (2006).
7. http://code.google.com/intl/el-GR/webtoolkit/
8. http://www.lehigh.edu/~zhp2/2004/0401/University0_0.owl
9. http://owl.cs.manchester.ac.uk/validator/
10. http://svn.versiondude.net/clark-parsia/datasets/lubm/query-owled/
11. http://gwt-ext.com/