Publishing OWL ontologies with Presto

Alexander DeLeon[1] and Michel Dumontier[1,2]

[1] School of Computer Science
[2] Department of Biology
Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, K1S5B6 Canada
adlbatti@scs.carleton.ca, michel_dumontier@carleton.ca

Abstract. Publishing RDF/OWL ontologies on the Semantic Web typically starts by placing the document in a web accessible location and ends with redirects of ontological components (classes, properties, individuals) to that the document. Unfortunately, this is seldom sufficient for expressive OWL ontologies in which reasoning is essential to determine the full extent of the entity in question. Moreover, the ability to dynamically query expressive ontologies yields new applications over static publishing including the possibility that these queries may be equivalent to new ontological entities. Here, we describe the design of a new tool for publishing OWL ontologies in a dynamic manner such that the ontology and all of its entities are web resolvable and queryable, hence opening new avenues for knowledge management on the Semantic Web.

1 Introduction

A core objective of the Semantic Web is to add machine understandable descriptions to the current Web, in part, by publishing ontologies that describe and relate entities using formal, logic-based representations. An essential aspect of the Semantic Web is to ensure that the terminology defined in ontologies are web-accessible such that information about the ontological entity may be discovered and links with related entities explored. The Linked Data architecture [1] suggests that HTTP URIs may be used as resource names, whether they be electronic documents or conceptual representations (i.e. the class of Cat or the first author). Should HTTP URIs be web resolvable then web client may discover additional knowledge by following links between these web-accessible resources.

The W3C Semantic Web Deployment Working Group recently proposed best practices in the publishing of RDF vocabulary on the web [2] in which the resolution of named entities (classes, properties, individuals) is achieved by redirection to a single document in which axiomatic statements are made about it. This approach may be exceedingly cumbersome should the entity be defined in a very large document, which is generally the case in ontologies for the life sciences. Crucially, the proposed solution may not be appropriate for OWL ontologies as the knowledge about an entity depends on the cumulative sum of knowledge obtained from the import of OWL documents into the knowledge base. Thus, the description of some entity may differ from one knowledge base to another.

Dynamic generation of ontology documentation may be provided using the OWLDoc server[3] whose interactive user interface with reasoner-enabled query capabilities reveal the sophistication of OWL ontologies. While OWLDoc server generates project-specific HTML documentation with permanent links for sharing, it does not currently provide RDF/OWL descriptions for use on the Semantic Web.

In this paper, we describe Presto (Published RESTful Ontology), a tool for publishing and querying OWL ontologies on the Semantic Web. Presto allows any party to publish their OWL knowledge base, including any imported documents. For a given ontology, Presto provides the following:

1. A self-referential namespace for all ontological documents and entities, so as to follow linked knowledge as a static ontological snapshot.
2. A RESTful service for DL and SPARQL queries that are identified by permanent HTTP URIs.
3. Content-negotiation capabilities to retrieve dynamically generated HTML or RDF/XML.

Finally, we demonstrate Presto's value in i) maintaining ontological interoperability, ii) building new ontologies with terminology defined from queries of other ontologies and iii) ontology version control.

2 Overview

Figure 1 illustrates the general architecture of Presto, a Java based application whose central data model relies on the OWL API (version 2.1.1) [3]. The general operation is as follows. Publishers invoke Presto with an OWL document and a target publishable URI that should resolve to the machine where Presto is running. The *Presto Manager* is a mediator component that provides RESTful services by interacting with several system components including a *DL-Reasoner*, a lucene-based entity indexer [4], Manchester Syntax parser from Protege[4] and HTML rendering from OWLDoc.[5] The Restlet Framework[6] is used to create HTTP handlers and representations for the URIs of the ontology and its entities. For instance, if the request `Accept` header includes the mime-type `application/rdf+xml` the response is rendered using RDF/XML otherwise an HMTL document generated with OWLDoc is returned to the client. The HTTP handler of the ontology URI has a special behavior which is that it could act as a query endpoint. When a request is made to the ontology URI using the `query` parameter, the server executes the query on the ontology, and returns the results as OWL, XML or HTML depending on the `Accept` header and the query language used. This querying service is futher described in section 4.

[3] http://www.co-ode.org/downloads/owldoc-server/
[4] http://protege.stanford.edu
[5] http://www.co-ode.org/downloads/owldoc/
[6] http://www.restlet.org/

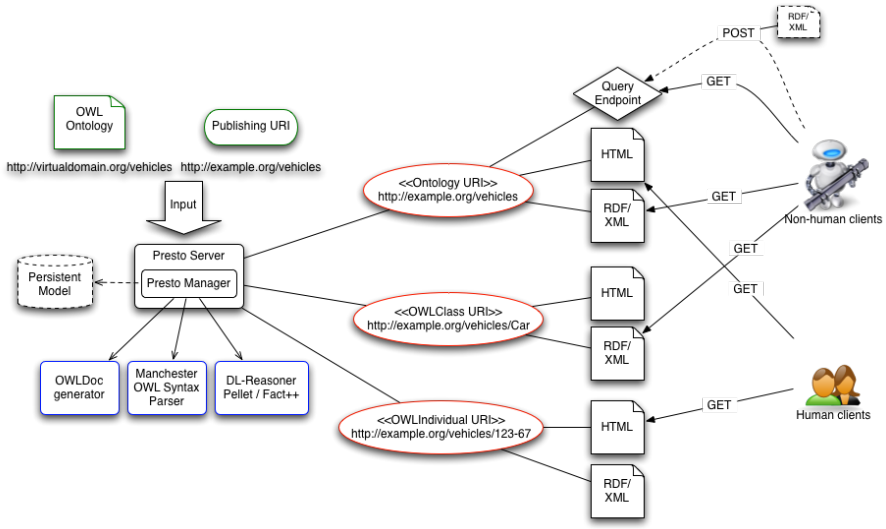

Fig. 1. Presto at a glance. Components with dotted-lines are envisioned as future work

3 Resolvable namespace for ontologies

The OWL 1.1 specification[5] establishes that the logical ontology URI need not be identical to its physical location (i.e. a resolvable HTTP URI). The lack of agreement causes problems for Semantic Web applications since they should check that the document at the physical location matches the logical URI specified. Hence, it is important that ontologies published on the Semantic Web have identical logical and physical names.

Presto gathers and publishes ontologies and their contents to a user-specified target namespace, which should be web-accessible. If the namespace of the document does not match the namespace of the ontology, it rewrites the document and entity URIs. To ensure interoperability, it optionally adds `owl:PriorVersion`, `owl:equivalentClass`, `owl:equivalentProperty`, and `owl:sameAs` axioms to maintain semantic equivalent relationships with the document, class, property and individual URIs, respectively. When a HTTP request is made to the URI of the ontology, the rewritten ontology is returned in its entire form. When the request is made to an entity URI, an OWL document is returned containing only those axioms (asserted and inferred) that directly describe the requested entity (i.e. the entity is the subject of the axiom or is the object and the axiom is symmetric, for instance owl:disjointWith). This document also imports the URIs of the ontologies where these axioms where drew from. In the future these imports will likely be replaced with a single import to an ontology module[6] extracted from these ontologies and the signature of the requested entity.

The format used in the HTTP response can be RDF/XML or HTML depending on the `Accept` header of the request. This allows human users to navigate

ontologies following HTML links in a web browser, while software agents can request machine understandable RDF/OWL.

3.1 Renaming entities

Presto replaces the URI of the input ontology, as well as, the URIs of all entities refered in the ontology. The URIs replacement is done using the following algorithm:

Let u be the resolvable URI where the ontology will be deployed.
Replace the ontology name URI by u.
For each entity URI e in the ontology, do the following:
  If e contains a fragment ("hashed names"), **replace** the hash (#) by a slash (/).
  Let n ("entity name") be the substring of e that starts from a character that immediately follows a slash and contains the minimum number of slashes necessary to guarantee uniqueness between all "entity names".
  Replace e by u/n.

Using the method described above, the *host* part of the original URI may be included in the new URI created by Presto. This is needed to avoid collitions between names. For example, if a given ontology contains the following two URIs: *http://foo.org/Car* and *http://bar.org/Car*, and this ontology is deploy under the name: *http://example.org/myOntology*, the resulting entity names would be: *http://example.org/myOntology/foo.org/Car* and *http://example.org/myOntology/bar.org/Car* respectively .

3.2 Reasoning enabled

Presto can be enabled to use an OWL-DL reasoner. By doing this, Presto is able to provide inferred axioms (those resulting from classification and realization of the ontology) with entity descriptions. The representation of an ontology entity obtained from an HTTP request to the entity URI will contain the axioms that were explicitly asserted by the ontology author complemented with those inferred from the ontology by the reasoner. Presto currently uses either Pellet 1.5 or Fact++ 1.1.9 depending on the user's preference.

Clients interested in consuming the ontology without intending to do any reasoning themselves can benefit from the inferences provided by Presto. These clients can obtain a larger amount of information from the ontology without the computational cost of reasoning. For instance, applications that process RDF information without interpretation of OWL semantics can be one of such clients. Another example would be browsers or interfaces that aid humans in the exploration of the ontology.

It is unknown to us what would be the impact of using an ontology together with its inferences in an application that intends to do additional DL-reasoning.

Intuitively it may appear that having these inferences would simplify future reasoning tasks. However, in practice, we suspect that most of current DL-reasoners will be negatively affected by receiving these inferences as inputs. The reason for our assumption is that the reasoner is unaware of the presents of these inference when executing its proving algorithm. For this reason we intend to extend Presto to allow clients to request the removal of inferences when requesting the representation of an ontology or one its entities. This feature will be supported by means of a HTTP parameters (e.g *http://example.org/ontology?noinferences*).

4 Query service

Presto is implemented as a RESTful service [**?**]. Representational State Transfer (REST) is centered around two basic principles: i) Resources as unique URLs and ii) Operations as HTTP methods (i.e. GET, POST). Depending on the HTTP `Accept` header of the Request, every URI in the resolvable namespace can be resolved to an HTML or RDF/OWL document. These URIs can be further parameterized. For instance, queries can be achieved by sending a HTTP GET request to the ontology URI with the mandatory `query` parameter whose value is string of the SPARQL or Manchester OWL DL query [7] along with the optional language parameter `queryLang`, whose value is either 'manchester' (default) or 'sparql'. To ask *what vehicles have a gas engine?* using the DL syntax for an ontology published at *http://www.example.org/vehicles*, one sends a GET request to the the following URL:

http://www.example.org/vehicles?query=Vehicle that hasPart some GasEngine [7]

Similarly, the URL to ask the same question using SPARQL is the following:

http://www.example.org/vehicles?query=DESCRIBE ?x WHERE {
?x rdf:type <http://www.example.org/vehicles/Vehicle> .
?x <http://www.example.org/vehicles/hasPart>
<http://www.example.org/vehicles/GasEngine> } [7]

The format in which results are returned from a query varies depending on the query language. When using the DL syntax, results are presented as an OWL document. This document contains the query expression conceptualized in a new *owl:Class* entity. This OWL class receives its name from the URI used to construct the query (i.e. the URI of the ontology + parameters). The remainder of the document contains those entities that are related to the query expression such as equivalent classes, subclasses and individuals that are classified into the class expression. The response document also includes an `owl:imports` axiom referring to the ontology containing the terminology. SPARQL queries are formatted to the "SPARQL Query Results XML Format" as proposed by the W3C recommendation [8].

[7] URL encoding has been omitted for readablity.

5 Discussion

Presto provides a convenient and intuitive mechanism for making an OWL ontology and its entities available as web-resolvable resources. In comparison with the method proposed by the W3C Semantic Web Deployment Working Group in [2], user may find Presto easier to use given the following reasons: (i) Presto does not require the installation and configuration of a conventional web server. (ii) There is no need for providing specific configuration of the web server for each ontology published. (iii) The URI rewriting feature of Presto allows to easily publish ontological content that were not original designed with HTTP resolvable URIs. (iv) Presto leverage the capabilities of OWLDoc server to dynamically generate HTML representations of ontology entities. In addition to this advantages, the are various use cases were the features of Presto can be found useful. We describe two of these use cases: building new ontologies with terminology defined from queries of other ontologies and ontology version control.

5.1 Identifiable Query Results

A DL query posted to an ontology published by Presto is identifiable by the URL used to construct the query. This URI identifies an OWL class which is equivalent to the class expression used to query the ontology. Moreover, this URI is stateless and therefore permanent. Because of its persistent URI, the results of a query can be imported into a new ontology. By doing this, the new ontology imports those individuals that are instances of the query class expression, as well as, those classes that are related to the query (i.e parent classes, subclasses and equivalent classes). The ontology that imports the query results can provide a name to the query class expression. To achieve this, an OWL class must be declared in the importing ontology and it must be made equivalent to the URI of the query. For example, let *http://www.example.org/vehicles* be a URI of published ontology. An HTTP GET to the query URI *http://www.example.org/vehicles?query=Vehicle and hasPart some ManualTransmission* returns the following:

```
<!DOCTYPE rdf:RDF [
 <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
 <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
 <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
 <!ENTITY query "http://www.example.org/vehicles/?query=" >
]>
<rdf:RDF
 xmlns="http://www.example.org/vehicles/"
 xmlns:query="http://www.example.org/vehicles/?query="
 xml:base="&query;Vehicle%20and%20hasPart%20some%20ManualTransmission">

<owl:Ontology rdf:about="">
 <owl:imports rdf:resource="http://www.example.org/vehicles" />
</owl:Ontology>
```

```
<!-- Query class expression -->
<owl:Class
 rdf:about="&query;Vehicle%20and%20hasPart%20some%20ManualTransmission" >
   <owl:intersectionOf rdf:parseType="Collection" >
    <rdf:Description rdf:about="Vehicle" />
    <owl:Restriction>
     <owl:onProperty rdf:resource="hasPart" />
     <owl:someValuesFrom rdf:resource="ManualTransmission" />
    </owl:Restriction>
   </owl:intersectionOf>
</owl:Class>

<!-- Individuals -->
<owl:Thing rdf:about="123-456" >
 <rdf:type
   rdf:resource="&query;Vehicle%20and%20hasPart%20some%20ManualTransmission" />
</owl:Thing>
</rdf:RDF>
```

The result of this query contains an OWL class with URI *http://www.example.org/vehicles/?query=Vehicle%20and%20hasPart%20some%20ManualTransmission* which correspond to the query expression. The resulting document also includes an OWL individual which is inferred to be an instance of the query. Note that this individual is asserted to have the type of the query class. One can define an ontology that uses the above results as follows:

```
<!DOCTYPE rdf:RDF [
 <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
 <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
 <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
 <!ENTITY query "http://www.example.org/vehicles/?query=" >
]>
<rdf:RDF
 xmlns="http://domain.org/myontology/"
 xmlns:query="http://www.example.org/vehicles/?query="
 xml:base="http://domain.org/myontology/">
<owl:Ontology rdf:about="">
 <owl:imports
  rdf:resource="&query;Vehicle%20and%20hasPart%20some%20ManualTransmission" />
</owl:Ontology

<owl:Class rdf:about="ManualVehicle" >
 <owl:equivalentClass
   rdf:about="&query;Vehicle%20and%20hasPart%20some%20ManualTransmission" />
</owl:Class>
</rdf:RDF>
```

The OWL ontology includes the query results by importing the URI *http://www.example.org/vehicles/?query=Vehicle%20and%20hasPart%20some%20ManualTransmission*. Additionally, the ontology defines the class *ManualVehicle* which is equivalent

to the query class expression identified by the URI
http://www.example.org/vehicles/?query=Vehicle%20and%20hasPart%20some%20ManualTransmission.
The advantage of this approach is that the reasoning tasks for classification and realization of the expression *Vehicle and hasPart ManualTransmission* is done before the import occurs. Essentially, this is distributing part of the reasoning work across the layers of imports.

5.2 Version Control

Presto can facilitate ontology version control and management. Each ontology is published on a namespace that includes the version number (e.g. *http://www.example.org/my-ontology-1.0.owl*) while the current version is published with an unversioned namespace (e.g. *http://www.example.org/my-ontology*). When a new ontology version is published, it replaces the previous version as the most current one. Using this pattern one can maintain version history and also provide prior versions in perpetuity for those web applications that depend on the stability afforded by a specific set of versioned documents. In contrast, applications interested in using the latest version need only link to the unversioned URI. Hence, one can leverage Presto's automatic URI rewriting feature and publishing capabilities to allow immediate and effortless deployment of new ontology versions.

6 Future Directions

The project aims to pursue three main goals in the near future. The first goal is to implement a procedure dynamically extract an ontology module that can capture the meaning of a given entity. A procedure which gives a practical approximation to the minimal module is described in [6] . Modules will be used as imports to entity descriptions such that clients can use the module to interpret the entity and reason about it. The second goal is to develop an augmented query answering model such that clients can provide additional knowledge when querying published ontologies. In practice, the query will be invoked by a HTTP POST to an ontology resource. The content of this POST will contain an OWL document which states the additional knowledge to be considered in the resolution of the client's query. The approach might also be valuable in composing and testing domain-specific hypotheses based on a set of user-specified premises for which the answer does not contain the empty set. The last goal aims to implement a persistent data model to replace the in-memory model for ontologies. This feature is considered a requirement for the system to scale when the size of the ontologies and/or the number of published ontologies are large. Using a similar approach to the one previously described in [4], we aim to optimize the task of answering DL queries over instances by translating them to SPARQL over an RDF triple store containing inferences.

References

[1] T. Berners-Lee. (2006, July) Linked data - design issues. [Online]. Available: http://www.w3.org/DesignIssues/LinkedData.html

[2] W3C. (2008, January) Best practice recipes for publishing rdf vocabularies. [Online]. Available: http://www.w3.org/TR/swbp-vocab-pub/

[3] M. Horridge, S. Bechhofer, and O. Noppens. Igniting the owl 1.1 touch paper: The owl api. [Online]. Available: http://owlapi.sourceforge.net/publications.html

[4] A. DeLeon-Battista, N. Villanueva-Rosales, M. Palenychka, and M. Dumontier, "Smart: A web-based, ontology-driven, semantic web query answering application," *Proceedings of the Semantic Web Challenge 2007*. [Online]. Available: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-295/paper17.pdf

[5] B. Motik, P. F. Patel-Schneider, and I. Horrocks, "Owl 1.1 web ontology language: Structural specification and functional-style syntax," *W3C Working Draft*, 8 January 2008.

[6] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "Just the right amount: Extracting modules from ontologies," in *Proceedings of WWW-2007: the 16th International World Wide Web Conference, Banff, Alberta, Canada, May 8–12, 2007*, 2007.

[7] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. H. Wang, "The manchester owl syntax."

[8] W3C. (2008, January) Sparql query results xml format. [Online]. Available: http://www.w3.org/TR/rdf-sparql-XMLres/