Using SPARQL to Query BioPortal Ontologies and Metadata

Manuel Salvadores, Matthew Horridge, Paul R. Alexander,
Ray W. Fergerson, Mark A. Musen, and Natalya F. Noy

Stanford Center for Biomedical Informatics Research
Stanford University, US
{manuelso,matthew.horridge,palexander,
ray.fergerson,musen,noy}@stanford.edu

Abstract. BioPortal is a repository of biomedical ontologies—the largest such repository, with more than 300 ontologies to date. This set includes ontologies that were developed in OWL, OBO and other languages, as well as a large number of medical terminologies that the US National Library of Medicine distributes in its own proprietary format. We have published the RDF based serializations of all these ontologies and their metadata at `sparql.bioontology.org`. This dataset contains 203M triples, representing both content and metadata for the 300+ ontologies; and 9M mappings between terms. This endpoint can be queried with SPARQL which opens new usage scenarios for the biomedical domain. This paper presents lessons learned from having redesigned several applications that today use this SPARQL endpoint to consume ontological data.

Keywords: Ontologies, SPARQL, RDF, Biomedical, Linked Data.

1 SPARQL In Use In BioPortal: Overview of Opportunities and Challenges

Ontology repositories act as a gateway for users who need to find ontologies for their applications. Ontology developers submit their ontologies to these repositories in order to promote their vocabularies and to encourage inter-operation. In biomedicine, cultural heritage, and other domains, many of the ontologies and vocabularies are extremely large, with tens of thousands of classes.

In our laboratory, we have developed BioPortal, a community-based ontology repository for biomedical ontologies [11]. Users can publish their ontologies to BioPortal, submit new versions, browse the ontologies, and access the ontologies and their components through a set of REST services. BioPortal provides search across all ontologies in its collection, a repository of automatically and manually generated mappings between classes in different ontologies, ontology reviews, new term requests, and discussions generated by the ontology users in the community. BioPortal contains metadata about each ontology and its versions as well as mappings between terms in different ontologies.

We had numerous requests from users to open a public SPARQL endpoint, which would enable them to query and analyze the data and metadata in much more fine-ground and application-specific ways than our set of REST APIs allowed. In December 2011, we released `sparql.biooontology.org` to provide direct access to ontology content, metadata and mappings. We describe the details of the structure of the dataset and how it implements the Linked Data principles elsewhere [17] and provide a short overview here in Section 2.

In the first months of the endpoint deployment, we have received valuable feedback from our user community and were able to identify some points of continuous debate around the use of `sparql.bioontology.org`. In this paper, we address the following three points that both proved challenging and provided a clear use case for the advantages of SPARQL:

– Retrieval of common attributes from multiple ontologies: BioPortal's ontologies, with 300 ontologies and growing, have been developed by different institutions and groups. Even though there are standard vocabularies and best practices, the flexibility of ontology languages allow ontology authors to use different techniques and patterns. If a user needs to query for information across several ontologies (e.g., looking for a string in a textual definition), she must know how each ontology developer modeled the definitions. Thus querying across multiple ontologies for common properties becomes cumbersome and error-prone. We try to alleviate this issue by providing simple reasoning (Section 3).
– Best practices in using a shared SPARQL endpoint: We have faced challenges in scaling on two different aspects: (1) the processing of complex queries and (2) client applications processing large outputs as result of a query. To develop more robust and fast applications and to promote a fair usage of our resources, we adopted simple best practices. Some of the best practices are SPARQL query constructions, others are design recommendations. We discuss these best practices in Section 4.
– Complex Query Articulation: The non-trivial mapping of complex OWL objects to RDF graphs can make queries verbose and difficult to articulate. A W3C recommendation [13] describes how OWL 2 maps to RDF graphs. Most libraries that transform OWL into RDF conform to this recommendation. To load OWL ontologies in our triple store we use the OWL-API that follows these recommendation [6]. We discuss the query articulation for OWL ontologies that are stored in a triplestore in Section 5.

In this paper, we do not try to solve these issues from a research point of view but rather we describe them so that other Semantic Web developers can plan ahead with a sense of what they will encounter. For each of these points, we present our pragmatic solutions that at least alleviated these issues. We discuss these points from a developer's point of view. When possible, we link the discussion to the current state of Semantic Web standards and technology in terms of solving a particular issue.

2 Background: Dataset Description

Researchers and practitioners in the Semantic Web normally deal with two types of information: (1) ontologies or TBoxes; and (2) instance data or simply *data*. BioPortal's content is almost exclusively ontologies and related artifacts. Other popular datasets of the Linked Data Cloud focus on *instance data* and ontologies and schemas play only a small role there. In the biomedical domain, ontologies play a very active and important role and many ontologies and vocabularies are extremely large, with tens of thousands of classes and complex expressions. For example, SNOMED CT, one of the key terminologies in biomedicine, has almost 400,000 classes [14]. The Gene Ontology (GO) has 34,000 classes [4]. These ontologies and terminologies are updated on a regular basis, some very frequently. For example, a new version of GO is published daily.

To host BioPortal's RDF content we use 4store as SPARQL server [5]. The best practices and opportunities that we describe in this paper not only apply to a particular RDF database and can be extrapolated to other deployments.

Ontology Content. The core of the BioPortal dataset is the content of each ontology that users have submitted to BioPortal. The BioPortal repository keeps multiple versions of each ontology. However, at the moment, `sparql.bioontology.org` exposes only the latest version of each. There are three main ontology formats in BioPortal:

– **OBO format** is a format that many developers of biomedical ontologies prefer because of its simplicity. OBO Editor, a tool that many ontology developers in biomedicine use, produces ontologies in this format [3]. The OWL API now provides a translation from OBO syntax into OWL syntax [22].
– The **Rich Release Format (RRF)** is primarily used by the US National Library of Medicine to distribute the vocabularies that constitute the Unified Medical Language System (UMLS) [8].
– **OWL** is a W3C recommendation for representing ontologies on the Semantic Web [9].

For OBO and OWL ontologies, the content in the triple store is the ontology that includes the closure of the *owl:imports* statements [18]. Prior to our recent quad store implementation, our data had not been stored as triples in our backend systems and therefore we need to follow a different workflow for each format to expose the existing content as RDF triples. To handle the RRF syntax we have developed the UMLS2RDF project [16]. UMLS2RDF is a set of scripts that connect to the UMLS MySQL release and transforms its content into RDF triples. To process OBO and OWL ontologies, we use the OWL-API [6]. The OWL-API can read the OBO syntax and all the OWL syntaxes (e.g: OWL/XML, Manchester, RDF and Manchester syntax). We also use the OWL-API to extract the import closure. We fetch imports from the web and materialize them, saving the whole materialized ontology in the data store.

Ontology Metadata. In addition to ontology content, we track metadata related to each ontology in the system. We represent the metadata using an OWL ontology that we developed for this purpose, the BioPortal Metadata Ontology, which extends the Ontology Metadata Vocabulary (OMV) [15]. The metadata is a set of instances in this OWL ontology. The two main entities in the metadata are *meta:VirtualOntology* and *omv:Ontology*. *meta:VirtualOntology* represents a container for all versions of an ontology; an *omv:Ontology* represents a particular ontology version (Figure 1).

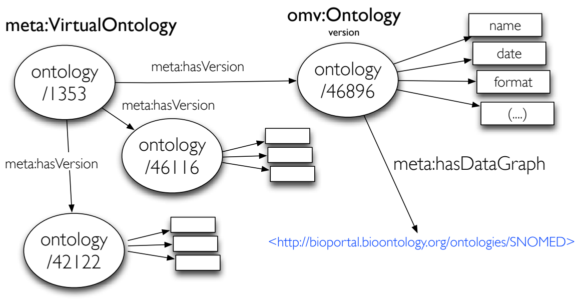


Fig. 1. Metadata: Virtual Ontologies and Version Ontologies

Mapping Data. The third type of data are the mappings between terms in different ontologies, which constitute an important part of the BioPortal repository [12]. Users can submit mappings to BioPortal through the Web interface or the REST APIs. In addition, the BioPortal team runs a series of processes to generate mappings automatically. A mapping in BioPortal connects two terms from different ontologies. It may also connect one term to many terms. We abstract the mappings into entities that record the provenance information of the mapping: the process that generated the mapping, when and how it was produced, the user who submitted it, the type of relation between classes, and so on. This information is represented in two sets of triples (a) the mapping itself and (b) the process information, which is referenced by all the mappings that the process generated.

3 Retrieval of Common Attributes from Multiple Ontologies

Ontologies in BioPortal vary in their content and structure. There are very rich representations, such as those found in the NCI Thesaurus, which has 111K *rdfs:subClassOf* relations. There are also terminologies, with no single transitive taxonomic relation, such as Medical Subject Headings (MeSH).

Ontology authors use different properties to represent common relations and attributes. The ontologies in BioPortal use 17 different properties to represent a preferred label of a term, and 28 different properties to store synonyms—even though standards, such as SKOS, provide recommendations for the properties to use in these cases. The two types of queries that we observe far more frequently than the rest are queries to (a) browse a taxonomy or (b) extract labels, synonyms and definitions. These type of queries are of particular interest for visualization and for building annotation tools. Both type of applications are very popular in the biomedical domain.

3.1 Retrieval of Preferred Names, Synonyms and Definitions

Preferred names, synonyms and definitions provide valuable term characterizations, often used in biomedical applications like annotation of clinical and scientific documents. For example, the tools developed by the BioPortal group include the NCBO Annotator, which allowed users to annotate their documents with ontology terms [21] and the NCBO Resource Index, which allows users to query an already annotated large collection of biomedical resources [7].

These types of resources must access lexical information in the ontologies such as preferred names and synonyms of terms. However, because different ontologies use different predicates to record each of this elements, it is difficult to developers to make their tools flexible enough to use any ontology in the repository. In order to provide the users of the BioPortal dataset with a uniform access to these properties, we link these different properties to the standard SKOS properties using *rdfs:subPropertyOf* relation. When ontology authors upload ontologies into BioPortal they have to choose what are the predicates that represent these attributes.

For example, properties that individual ontologies use for preferred labels all become subproperties of *skos:prefLabel* in a "globals" graph; properties that individual ontology authors chose to represent synonyms all become subproperties of *skos:altLabel*. As the result, we have a set of common predicates to query on lexical annotations across ontologies. The globals graph contains a hierarchy of properties that maps each custom attribute property to one of the standard predicates. We use this hierarchy of predicates to rewrite internally the SPARQL query using backward-chaining reasoning. Figure 2 shows an example of a SPARQL query for an ontology that uses a custom predicate to record preferred labels. In this case, the user does not need to know the specific predicate and she can query on the standard *skos:prefLabel*.

Internally our triple store rewrites the SPARQL query and not only binds the property *skos:prefLabel* but also the *rdfs:subPropertyOf* closure. Thus, the query in Figure 2 will also contain `http://NIF-RTH.owl#core_prefLabel`. BioPortal maintains a mirror of the 4store database where this backward-chained reasoning is implemented.[1] The entailment regime that we implemented follows the Minimal RDFS Semantics [10,19].

[1] `https://github.com/ncbo/4store`

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?termURI ?prefLabel
   FROM <http://bioportal.bioontology.org/ontologies/NIF-RTH>
   FROM <http://bioportal.bioontology.org/ontologies/globals>
WHERE {
   ?termURI a owl:Class;  skos:prefLabel ?prefLabel .
}
```

```
<http://NIF-RTH.owl/#nif_resourcep17:birnlex_2353> "Commercial license"
<http://NIF-RTH.owl/#nif_resourcenif_resource:nlx_res_20090414> "Biomaterial supply resource"
<http://NIF-RTH.owl/#nif_resourcep17:birnlex_2218> "3D Time-series analysis software"
<http://NIF-RTH.owl/#nif_resourcenif_resource:nlx_res_090904>  "Reference atlas"
                                    ( 150 solutions omitted)
```

Fig. 2. SPARQL Query on a standard preferred label property. The query result returns preferred labels for an ontology even though the authors used a nonstandard property for this attribute. The predicate used in this case is `http://NIF-RTH.owl#core_prefLabel`

The use *rdfs:subPropertyOf* reasoning was successful for our use case. Most applications consuming labels and definitions care only about the default predicates we provide as root elements of each property hierarchy. We documented this technique in our Wiki and we have noticed that users tend to rework the examples keeping the "globals" graph to use the standard predicates.[2]

3.2 Hierarchy Retrieval

Historically, browsing a taxonomic hierarchy is one of the most common ways to browse ontologies. The way in which ontologies represent their taxonomic hierarchy does not differ greatly among ontologies. By far, the predominant predicate for this purpose is *rdfs:subClassOf*. There are 6M triples with this predicate in the BioPortal triple store.

The fact that in modern triples stores we can load 300+ ontologies is of particular interest to applications that need to navigate multiple ontologies. Having this kind of remote service allows for incremental browsing. Applications need to know only a class IRI to start the navigation, such as root nodes. Then, with simple SPARQL queries the application would be able to browse the hierarchy. A SPARQL query like the one in Figure 3 would retrieve, by default, only direct asserted subclasses with their labels.

In `sparql.bioontology.org`, the default behaviour for queries like the one in Figure 3 is to retrieve only direct asserted subclasses; or direct superclasses if we invert the *rdfs:subClassOf* pattern. However, many times our users want to retrieve the subclass closure of a given node. If a triple store does not implement any reasoning, getting the closure requires us to query recursively until we reach all nodes. We argue in Section 5 that property paths are also not an option

[2] `http://www.bioontology.org/wiki/index.php/SPARQL_BioPortal`

```
SELECT ?subClass ?prefLabel
FROM <http://bioportal.bioontology.org/ontologies/SNOMED>
FROM <http://bioportal.bioontology.org/ontologies/globals>
WHERE {
        ?subClass  rdfs:subClassOf  <http://purl.bioontology.org/ontology/SNOMEDCT/12738006>;
                   skos:prefLabel ?prefLabel .}
```

Fig. 3. SPARQL query to retrieve all subclass elements from a given class and their labels. The IRI in the example is from the SNOMED CT ontology and represents the term "Brain Structure." This example combines taxonomy browsing with preferred name retrieval. This query outputs five results in the current SNOMED CT ontology.

to return hierarchy closures efficiently. To help users with querying hierarchy closures we again use Minimal RDFS reasoning and 4store's backward chained reasoner [19]. Backward-chain reasoning works very well in this case because it allows us to provide switchable structural *rdfs:subClassOf* reasoning. Sometimes, users only care about direct asserted subclasses and superclasses, like when visualizing the hierarchy tree. In this case, they can use an CGI parameter indicating that the reasoning should be switched off. In other cases, when users need the full closure, they can just switch it on.[3]

Some applications need to traverse the hierarchy for a fixed number of steps. For instance, the NCBO Annotator [21] includes an option to annotate text including ancestors of the terms that appear directly in the text up to a certain level. Neither direct superclasses nor structural subclass reasoning can help with this issue. But, with the SPARQL union operator it is possible to formulate queries for this purpose. The query in Figure 4 uses the UNIONS together with SPARQL 1.1 BIND operator to provide this functionality. The BIND operator allow us to identify the provenance of each resulting solution. We show two solutions at the bottom, "Parasite identification" is the direct superclass and "Identification procedure for living organism" is the superclass in distance 2. We can apply ORDER BY to the *nstep* variable to rank the results. In this case, classes that are closer to the query term are ranked higher.

The performance of queries like the one in Figure 4 is critical to the NCBO Annotator. Recently, we have measure how these queries perform as we add more UNION/BIND blocks into the query structure. Figure 5 shows the performance of queries for SNOMED CT. SNOMED CT contains a rich taxonomy with 539K subclass axioms, 395K classes and up to 32 levels in the hierarchy. We studied groups of queries that request terms within 1, 5, 10, 15, and 20 hierarchical levels from a given term. The average performance of these 5 groups that we studied remains below 0.11 seconds. Average performance for getting 5 and 10 levels—common choices by BioPortal users—are 0.019 and 0.13 seconds respectively. For NCBO Annotator this performance is within an expected range of acceptable performance. We did not see a drastic performance degradation as we added more UNION blocks into the queries.

[3] `https://github.com/msalvadores/4sr/wiki/4sr-reasoning-and-queries`

```
SELECT DISTINCT ?ss0 ?nstep ?label {
  { GRAPH <http://bioportal.bioontology.org/ontologies/SNOMEDCT> {
      <http://purl.bioontology.org/ontology/SNOMEDCT/122040007> rdfs:subClassOf ?ss1 .
      ?ss1 rdfs:subClassOf ?ss0 .
      ?ss0 skos:prefLabel ?label
      BIND ('2' AS ?nstep) .
  } }
  UNION { GRAPH <http://bioportal.bioontology.org/ontologies/SNOMEDCT> {
      <http://purl.bioontology.org/ontology/SNOMEDCT/122040007> rdfs:subClassOf ?ss0 .
      ?ss0 skos:prefLabel ?label
      BIND ('1' AS ?nstep) .
  } }
} ORDER BY ?nstep

<http://purl.bioontology.org/ontology/SNOMEDCT/122069003> "1" "Parasite identification"@EN
<http://purl.bioontology.org/ontology/SNOMEDCT/108266002> "2" "Identification procedure for living organism"@EN
```

Fig. 4. SPARQL query to retrieve two ancestors of a given element. The first GRAPH block retrieves ancestors in distance 2 and the second GRAPH block ancestors in distance 1.

BioPortal allows users to browse hierarchies that are interconnected by mappings. For instance, if we try to find subclasses of term A to extract their labels and we do not find any, we can retrieve mappings for the term A and look for subclasses of a related term in a different ontology. Mappings provide connection paths that can be crossed depending on the application needs. For instance, when looking at the term "malignant hyperthermia" from the Human Disease Ontology we do not see any subclasses. But this term in BioPortal has 24 mappings. Figure 6 shows the query that would find subclasses from other ontologies where a term is mapped to "malignant hyperthermia." The query in Figure 6 retrieves 13 solutions with terms and labels from 3 other ontologies (SNOMED CT, MESH and CTV3). We discussed the details of the RDF structure of BioPortal mappings elsewhere [17].

4 Best Practices in Using a Shared SPARQL Endpoint

Our RDF data store contains 203M triples with more than 2,000 different predicates. SPARQL allows the construction of very complex queries. One typical problem are complex joins that generate very large intermediate results. The size of our database and some of the queries that our users run make this issue a recurring one. SPARQL engines are still to improve in terms of query planning optimisations and we have identified a few best practices that can be beneficial to develop applications that use shared SPARQL endpoints.

Queries containing non-selective graph patterns tend to generate large intermediate results in SPARQL engines. This issue is particularly problematic in an open SPARQL endpoint. An open SPARQL endpoint is normally a shared resource. From the perspective of the publisher, we can optimise resources if we delegate some of the query processing to the client's application. Thus, we encourage our users to query an open SPARQL endpoint iteratively with selective queries, which overall provide the functionality of a non-selective query.

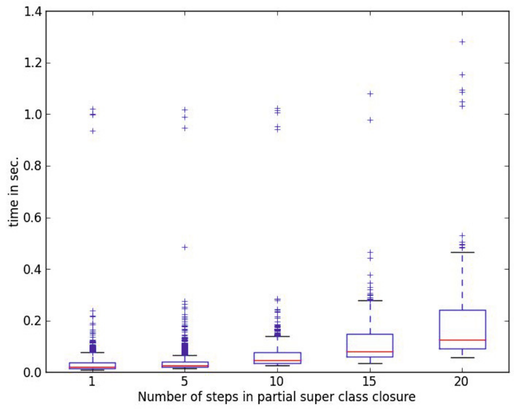


Fig. 5. Performance analysis of queries that retrieve N step superclass elements together with their labels. X-axis represents the number of steps. We have measured 1, 5, 10, 15 and 20 steps. We processed 2,800 SPARQL queries for each number of steps. We used 2,800 random leaf classes from SNOMED CT to construct queries like the one shown in Figure 4.

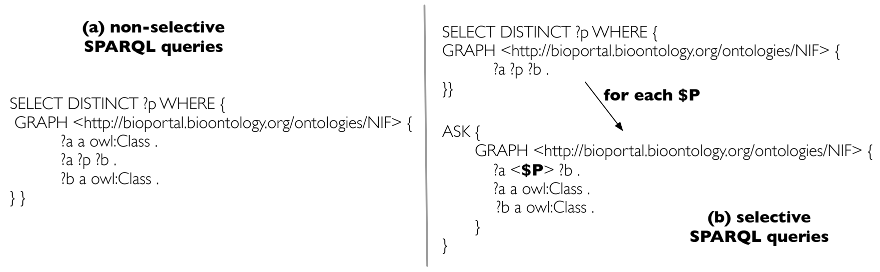
Figure 7 shows two different approaches to retrieving the same information from our SPARQL endpoint. Figure 7(a) uses a query that contains one non-selective triple pattern "?a ?p ?b" joined with two other triple patterns. The combination of these three triple patterns generates large intermediate results and computationally expensive joins. Queries in Figure 7(b) release the SPARQL server from some of the processing. Queries like the first one in Figure 7(b) are optimised in most SPARQL engines. The second query, that is part of an inner loop, becomes more selective. Here we do not argue that one is faster than the other. Our point is that the second approach, with multiple selective queries, is more likely to succeed in the open Web. Most triple stores implement mechanisms to restrict resources used by queries. These restrictions are often implemented in terms of "query timeouts" or "join space restrictions." `sparql.bioontology.org` uses two 4store mechanisms to limit execution of expensive queries. These are soft limits and join space restrictions.[4] Wherever non-selective queries are likely to fail, there is usually another approach with selective queries that is likely to succeed.

For the same reasons, we recommend the use of OFFSET and LIMIT to paginate over results. Queries that return all preferred names from an ontology (cf. Section 2) are likely to hit resource limits and fail on some of the largest ontologies in BioPortal, such as NCBITaxon with more than 500K terms and 4.6M triples. In these cases, paginating the results with OFFSET and LIMIT will help to return all the solutions. Most triple stores will produce a consistent

[4] `http://4store.org/trac/wiki/SparqlServer`

```
SELECT DISTINCT ?subClassOf ?label WHERE {
            ?s maps:source <http://purl.obolibrary.org/obo/DOID_8545>;
               maps:target ?target .
            ?subClassOf rdfs:subClassOf ?target .
            ?subClassOf skos:prefLabel ?label .
}
               <http://purl.bioontology.org/ontology/SNOMEDCT/213026003> "Malignant hyperpyrexia due to anesthetic"
               <http://bioonto.de/mesh.owl#C531737>   "Malignant fever"
               <http://bioonto.de/mesh.owl#C535695>   "Malignant hyperthermia susceptibility type 2"
               <http://bioonto.de/mesh.owl#C538343>   "Native American myopathy"
                                        ( 8 results omitted)
```

Fig. 6. SPARQL query that uses mappings for the term `http://purl.obolibrary.org/obo/DOID_8545` to reach other ontology term and retrieve their labels. The query returns 13 solutions, in the figure only 4 are shown the rest are omitted.


(a) non-selective SPARQL queries
for each $P
(b) selective SPARQL queries

```
SELECT DISTINCT ?p WHERE {
 GRAPH <http://bioportal.bioontology.org/ontologies/NIF> {
        ?a a owl:Class .
        ?a ?p ?b .
        ?b a owl:Class .
} }
```

```
SELECT DISTINCT ?p WHERE {
GRAPH <http://bioportal.bioontology.org/ontologies/NIF> {
        ?a ?p ?b .
}}

ASK {
    GRAPH <http://bioportal.bioontology.org/ontologies/NIF> {
        ?a <$P> ?b .
        ?a a owl:Class .
        ?b a owl:Class .
    }
}
```

Fig. 7. (a) shows a SPARQL query to retrieve all the predicates that connect two resources that are instances of *owl:Class*. (b) shows two queries, the top query gets all the distinct predicates in the graph; the bottom query uses ASK to see if a predicate P connects one pair of instances of *owl:Class*. Both (a) and (b) are restricted to the NIF ontology graph.

output when using OFFSET and LIMIT without ORDER BY. The sequence of the solution will, in most cases, follow the output of the last join operation.

The use of ORDER BY and GROUP BY do not play very well with resource contention. To compute these two operators, the SPARQL query engine needs to aggregate or sort all query solutions. Thus, the output needs to be kept in memory for a longer period of time and disqualifies the engine from streaming out results. If the solution space is large enough, the query is likely to hit timeouts.

Frequently, very simple queries produce an extremely large resultset. These resultsets have to be moved from our endpoint to the client application. Even though sometimes it looks like the issued query is taking a long time to be processed, in reality, significant portion of this time is spent transferring the result and parsing the result on the client side. For instance, the retrieval of all preferred names from NCBITaxon—500K solutions—generates a JSON or XML output of 97MB or 121MB, respectively. On average, this query takes 7.05 seconds in the query engine. Parsing the JSON output takes 55 seconds using

Python 2.6 and the built-in JSON parser and 15 seconds using the python-cjson library.[5] With Java, Jena's ARQ SPARQL client library processes XML SPARQL results as a stream of solutions adding little memory overhead to the client. For the previous example, parsing the result adds only 1.6 seconds of extra processing time.[6] Sesame, also for Java, does the same trick and adds 2.4 seconds of processing time when parsing the SPARQL XML result. One can see that times can differ greatly depending on the mechanism we use to process the SPARQL results. It is important that we choose the library that performs best on a given platform; and often a library that will parse the SPARQL resultset on-demand as rows are read by the application.

We encourage users to implement caching between their applications and our SPARQL endpoint. They cannot expect that they will always get SPARQL answers within a small response time range. Like many other SPARQL endpoints in the Linked Data Cloud, `sparql.bioontology.org` is a shared resource and its performance does not depend only on the queries submitted by one user. The overall load of the server affects all users. In that sense, we do not allow singles IPs to have more than 6 threads running simultaneously. We require API keys to grant access to private graphs [17]. Though we do not yet use API keys to queue user queries and implement resource contention, this is something that we have considered for future work.

5 Complex Query Articulation

The normative exchange syntax for OWL 2 ontologies is RDF/XML. The OWL 2 specification contains a document which describes how to map OWL 2 constructs into triples that form an RDF graph [13]. For all ontologies that users submit to BioPortal in OWL format, we parse and translate them to their RDF graph representations using this mapping. In the case of OBO ontologies these are first translated to OWL using the mapping at [22]. In some cases, such as when OWL axioms contain only class, property or individual names, the mapping is straightforward. For example if :A and :B refer to class names, SubClassOf(:A :B) is mapped to one triple :A rdfs:subClassOf :B. Similarly, ObjectPropertyDomain(:R :A) is mapped to :R rdfs:domain :A. However, when ontology contains complex class expressions or OWL 2 nary axioms, such as disjoint classes axioms with more than 2 classes, a single OWL axiom may be mapped into multiple triples. These triples form tree shaped graphs connected with RDF blank nodes. For example, consider the axiom SubClassOf(:A ObjectSomeValuesFrom(:hasPart :B)) which states that all instances of :A have hasPart relationships to instances of :B (an extremely common form of axiom in biomedical ontologies, particularly OBO ontologies). We map this axiom into the following RDF Graph:

[5] `http://pypi.python.org/pypi/python-cjson`
[6] `http://jena.apache.org/`

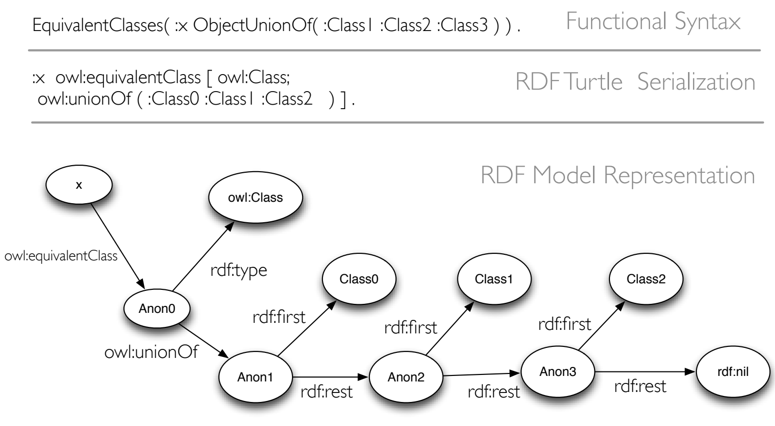


Fig. 8. Example of an equivalent class definition as a union of three classes using the OWL language. The top part of the figure shows the functional syntax often used in OWL documents. The next representation is in RDF/Turtle, a readable RDF serialization. The bottom part of the figure is the representation of the same OWL object very much as it would look like in triples.

```
:A rdfs:subClassOf _:x .
_:x rdf:type owl:Restriction .
_:x owl:onProperty :hasPart .
_:x owl:someValuesFrom :B .
```

This single axiom requires four triples. Also notice the use of blank nodes to tie everything together. For axioms that contain sets of objects, it is common to RDF lists are used to serialize these sets. For example, EquivalentClasses(:A ObjectUnionOf(:B :C :D)) gets mapped to the graph shown in Figure 8.

Ultimately, the mapping of axioms to RDF graphs is non-trivial. Querying these graphs to explore the structure of axioms requires special knowledge of this mapping, and understanding of how special RDF constructions such as RDF lists work, and multiple calls to retrieve the subgraph associated with a single axiom. From a modeler's perspective, these constructs look simple when presented in Functional or Manchester Syntax but appear overly complicated when represented as an RDF graph. In summary, the triple-based representation is not an end-user facing representation or presentation syntax.

Figure 9 shows two examples taken from BioPortal ontologies. The left-hand side is the definition of the term "Vaccine" from the Vaccine Ontology. "Vaccine" is characterized with several data type properties that give different descriptions of the term "Vaccine." "Vaccine" is a subclass of a class name (OBI_0000047[7]) but also an equivalent class of an intersection of three classes. Two of these classes are themselves complex class expressions. The RDF serialization of the

[7] OBI_0000047: Is a material entity that is created or changed during material processing.

```
obo:VO_0000001
   a owl:Class ;
   rdfs:label "vaccine" ;
   rdfs:seeAlso "MeSH: D014612" ;
   obo:IAO_0000115 "A vaccine is a processed (...) " ;
   obo:IAO_0000116 "Many vaccines are developed (...) " ;
   obo:IAO_0000117 "YH, BP, BS, MC, LC, XZ, RS" ;
   rdfs:subClassOf obo:OBI_0000047 ;
   owl:equivalentClass [
      a owl:Class ;
      owl:intersectionOf (obo:OBI_0000047
         [
            a owl:Restriction ;
            owl:onProperty obo:BFO_0000085 ;
            owl:someValuesFrom [
               a owl:Class ;
               owl:intersectionOf (obo:VO_0000278
                  [
                     a owl:Restriction ;
                     owl:onProperty obo:BFO_0000054 ;
                     owl:someValuesFrom obo:VO_0000494
                  ]
               )
            ]
         ]
         [
            a owl:Restriction ;
            owl:onProperty obo:OBI_0000312 ;
            owl:someValuesFrom obo:VO_0000590
         ]
      )
   ] .
```

```
fma:AAL
   fma:FMAID "276388"^^xsd:string ;
   a owl:FunctionalProperty,
     owl:ObjectProperty ;
   rdfs:domain [
      a owl:Class ;
      owl:unionOf (
         fma:Segment_of_telencephalon
         fma:Putamen
         fma:Amygdala
         fma:Region_of_cerebral_cortex
         fma:Organ_component_of_neuraxis
         fma:Neuraxis
         fma:Dura_mater
         fma:Segment_of_neural_tree_organ
         fma:Globus_pallidus
         fma:Lobule_of_cerebral_hemisphere
         fma:Set_of_neuraxis_structures
         fma:Caudate_nucleus
      )
   ] ;
   rdfs:range fma:AAL_term .
```

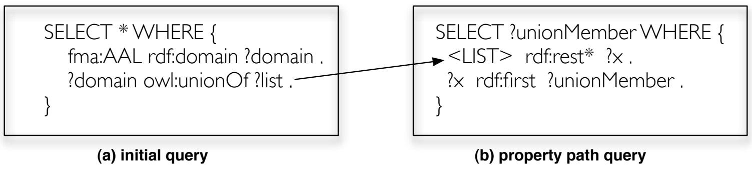
Fig. 9. Examples of relatively complex OWL constructions formatted in RDF/Turtle. Left side is an example taken from the Vaccine Ontology. Right side is an example from the Foundational Model of Anatomy.

term "Vaccine" generates 10 blank nodes and 10 SPARQL queries are required to browse this graph.[8] The right-hand side of Figure 9 shows the construction of an object property where the domain is represented as the union of a collection of 12 classes.

Users using `sparql.bioontology.org` often ask how to retrieve ontology elements like the ones that we show in Figure 9. It is somehow challenging for users sitting in front of a SPARQL query editor to articulate queries that will extract the triples that construct these OWL objects. They need libraries that recursively browse the RDF graph, libraries that understand the OWL mapping to RDF graphs by means of sets of SPARQL queries that are connected properly. To the best of our knowledge, tools that offer this functionality are not yet available in the open source community. There are tools that can parse ontologies in RDF but there are no tools that can extract parts of an ontology using SPARQL.

Browsing recursively an RDF graph with SPARQL can be problematic. Some SPARQL parsers and triple store databases do not allow the use of blank nodes as IRI literals in SPARQL queries. In this case, we need to re-articulate the query that led us to the blank node to retrieve any out-going elements from the blank node. Because this issue is such a major one, most SPARQL engines have addressed it with out-of-specification implementations. RDF 1.1 is in the process of specifying an official solution. The early RDF 1.1 draft says that systems are allowed to replace blank nodes with IRIs if they follow an IRI pattern that will

[8] Assuming that we use only SPARQL 1.0 and the SPARQL 1.1 property path specification is not available.



```
SELECT * WHERE {
    fma:AAL rdf:domain ?domain .
    ?domain owl:unionOf ?list .
}
```

```
SELECT ?unionMember WHERE {
  <LIST>  rdf:rest*  ?x .
  ?x  rdf:first  ?unionMember .
}
```

Fig. 10. Example of two queries that would retrieve the definition in Figure 9–right side. The initial query instantiates the beginning of the list. The property path query (b) uses gets as input the beginning of the from (a) and using the * property path operator traverses all the RDF list blank nodes binding the list elements in the variable *?unionMember*

be IETF registered. This technique is also known as Skolem IRIs [2]. The triple store used in `sparql.bioontology.org` has already implemented Skolem IRI replacement and our users can safely traverse RDF graphs with blank nodes [5].

The problem of retrieving RDF lists with SPARQL is, in theory, mitigated in the SPARQL 1.1 specification. In SPARQL 1.1, the property path specification defines enhanced navigational functionalities [20]. Property paths are defined as regular expressions that help to traverse RDF graphs. A property path query retrieves pairs of connecting nodes where the paths that link those nodes satisfy a path defined with a regular expression. Figure 10 shows the use of property paths to retrieve the definition in the right-hand side of Figure 9. Even though property paths provide a convenient way to query structures such as RDF lists with SPARQL, many current implementations of property paths—as of November 2011—have poor performance. Arenas and colleagues argued that the poor performance is not the result of particularly bad implementations but rather is due to the complexity of the specification itself [1]. Furthermore, according to the current specification, a property path query using rdf:rest*/rdf:first does not have to return the elements in the order they occur. The preamble of the property paths specification acknowledges this issue but contends that adding order to property paths would add significant complexity. In OWL, even though the language uses RDF lists to implement sequences of class expressions, the order of the elements does not change the "meaning." A union of classes A and B is the same as the union of B and A. Thus, many applications could potentially use property paths without order. However, many ontology visualization and editing tools like to maintain a fixed order of elements because humans tend to remember the position of UI components.

In summary, our experience shows that with or without property paths, retrieving OWL objects from a SPARQL endpoint can be challenging. The discussion that we presented in this section has described how OWL, RDF and SPARQL converge to solve parts of the problem and which parts are still matter of discussion in the Semantic Web community.

6 Conclusions

In general, RDF stores work extremely well as backend technology for quering ontology repositories due to their schema-less nature. In recent years, triple store technology has improved dramatically. Our deployment shows that it is feasible to publish 300+ ontologies—some with hundreds of thousands of classes and millions of axioms—in a public shared SPARQL endpoint. Triple stores have also become an important component in the Web of Data due to the standarization and adoption of RDF and SPARQL.

The BioPortal community had demanded access to our data via the SPARQL query language and in this paper we describe some of the design issues behind the implementation and deployment of `sparql.bioontology.org`. Our use of SPARQL is different from many other use cases because our data are primarily ontologies themselves and not data about individuals. Our experience shows that SPARQL and a small amount of reasoning can be particularly powerful in providing easy access to common attributes from our dataset, such as preferred names, synonyms, definitions and taxonomies—even though ontology authors use different RDF properties to represent these attributes. However, our experience also highlighted challenges in running a shared open SPARQL endpoint. We can overcome these challenges if we encourage developers to conform to a set of simple best practices. Finally, because our dataset includes OWL ontologies, we need to use `sparql.bioontology.org` to query the structure of these ontologies. Our experience shows that exposing OWL through a SPARQL endpoint poses a number of challenges. In future work, we plan to develop a set of SPARQL query templates to make it easier for others to explore the structure of these ontologies through `sparql.bioontology.org`.

Acknowledgments. This work was supported by the National Center for Biomedical Ontology, under grant U54 HG004028 from the National Institutes of Health.

References

1. Arenas, M., Conca, S., Pérez, J.: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In: WWW, pp. 629–638 (2012)
2. Cyganiak, R., Wood, D.: RDF 1.1 concepts and abstract syntax, `http://www.w3.org/TR/2012/WD-rdf11-concepts-20120605/`
3. Day-Richter, J., Harris, M.A., Haendel, M.: Gene Ontology OBO-Edit Working Group, Lewis, S.: OBO-Edit–an ontology editor for biologists. Bioinformatics 23(16), 2198–2200 (2007), `http://dx.doi.org/10.1093/bioinformatics/btm112`
4. Gene Ontology Consortium: The Gene Ontology (GO) project. Nucleic Acids Research 34(suppl. 1), D322–D326 (2006)
5. Harris, S., Lamb, N., Shadbolt, N.: 4store: The Design and Implementation of a Clustered RDF Store. In: Scalable Semantic Web Knowledge Base Systems, SSWS 2009, pp. 94–109 (2009)

6. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web 2(1), 11–21 (2011)
7. Jonquet, C., LePendu, P., Falconer, S.M., Coulet, A., Noy, N.F., Musen, M.A., Shah, N.H.: NCBO Resource Index: Ontology-based search and mining of biomedical resources. Journal of Web Semantics (JWS) 9(3) (2011)
8. Lindberg, C.: The Unified Medical Language System (UMLS) of the National Library of Medicine. Journal of American Medical Record Association 61(5), 40–42 (1990)
9. Motik, B., Patel-Schneider, P.F., Parsia, B.: Owl 2 web ontology language structural specification and functional-style syntax, `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`
10. Muñoz, S., Pérez, J., Gutierrez, C.: Simple and efficient minimal RDFS. Web Semant 7, 220–234 (2009)
11. Noy, N.F., Shah, N.H., Whetzel, P.L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D.L., Storey, M.A., Chute, C.G., Musen, M.A.: BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research 37(suppl. 2), W170–W173 (2009), `http://dx.doi.org/10.1093/nar/gkp440`
12. Noy, N.F., Griffith, N., Musen, M.A.: Collecting Community-Based Mappings in an Ontology Repository. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 371–386. Springer, Heidelberg (2008)
13. Patel-Schneider, P.F., Motik, B.: OWL 2 Web Ontology Language Mapping to RDF Graphs, `http://www.w3.org/TR/owl2-mapping-to-rdf/`
14. Price, C., Spackman, K.: SNOMED clinical terms. British Journal of Healthcare Computing & Information Management 17(3), 27–31 (2000)
15. Project, N.: The BioPortal Metadata Ontology (2012), `http://purl.bioontology.org/ontology/BPMetadata`
16. Salvadores, M.: UMLS2RDF Unified Medical Language System (UMLS) into RDF (2012), `https://github.com/ncbo/umls2rdf`
17. Salvadores, M., Alexander, P.R., Musen, M.A., Noy, N.F.: Bioportal as a dataset of linked biomedical ontologies and terminologies in RDF. Semantic Web Journal. IOS Press Journal (under review)
18. Salvadores, M., Alexander, P.R., Musen, M.A., Noy, N.F.: The quad economy of a semantic web ontology repository. In: The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems, SSWS 2011 (2011)
19. Salvadores, M., Correndo, G., Harris, S., Gibbins, N., Shadbolt, N.: The Design and Implementation of Minimal RDFS Backward Reasoning in 4store. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part II. LNCS, vol. 6644, pp. 139–153. Springer, Heidelberg (2011)
20. Seaborne, A.: SPARQL 1.1 property paths, `http://www.w3.org/TR/2010/WD-sparql11-property-paths-20100126/`
21. Shah, N.H., Jonquet, C., Chiang, A.P., Butte, A.J., Chen, R., Musen, M.A.: Ontology-driven indexing of public datasets for translational bioinformatics. BMC Bioinformatics 10 (suppl. 2), S1 (2009)
22. Tirmizi, S.H., Aitken, S., Moreira, D.A., Mungall, C., Sequeda, J., Shah, N.H., Miranker, D.P.: OBO & OWL: Roundtrip ontology transformations. In: SWAT4LS (2009)