The Quad Economy of a Semantic Web Ontology Repository

Manuel Salvadores, Paul R Alexander,
Mark A. Musen, and Natalya F. Noy

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

Abstract. Quad stores have a number of features that make them very attractive for Semantic Web applications. Quad stores use Named Graphs to give contextual information to RDF graphs. Developers can use these contexts to incorporate meta-data such as provenance and versioning. The OWL language specification provides the *owl:imports* construct as one of the key ways to reuse the ontologies on the Semantic Web. When one ontology imports another through the *owl:imports* statement, all axioms from the imported ontology are brought to the source ontology. When implementing an online ontology repository, we have explored a number of different approaches to reflect these imports and contextual information in a quad store. These approaches have a direct impact on storage requirements, query articulation, and reusability.
This paper describes different models to represent contextual information, ontology imports and versioning. We have performed the experiments in the context of storing ontologies from the BioPortal ontology repository. This repository has 304 ontologies, with multiple versions for many of them. We extract metrics on the levels of imports, and the storage and performance requirements and discuss the trade-offs among query articulation, reusability and scalability.

Keywords: Ontology Repository, Scalability, Quad Stores, SPARQL

1 Introduction

Ontology repositories act as a gateway for users who need to find ontologies for their applications. Research institutions and companies 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. For example, SNOMED CT, one of the key terminologies in biomedicine, has almost 400,000 classes. The Gene Ontology (GO) has 34,000 classes. These ontologies and terminologies are updated on a regular basis, some very frequently. For example, a new version of the Gene Ontology is published daily.

In our laboratory, we have developed BioPortal, a community-based ontology repository for biomedical ontologies [1].[1] 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.

At the time of this writing, BioPortal has 304 ontologies, with 5.3 million classes among them. Many of these ontologies have multiple versions in the repository. Many of the ontologies import one another or import ontologies that are not in the BioPortal repository. Finally, the ontologies come in a number of different formats, and thus are persisted in different backends using various APIs. There is a single Lucene index of classes to enable search across ontologies and the BioPortal REST API provides uniform access to all the ontologies in the repository so that API users can be agnostic about the individual formats. For each ontology, we provide access to the latest version of the ontology, and partial access to the earlier versions.

We are currently working on creating a single database—a quad store—for all the ontologies and their associated metadata. As part of the design, we evaluated the cost, in terms of data size, of representing and materializing various aspects of the ontologies, including the import structure of the ontologies. In this paper, we report on our analysis and the trade-offs with respect to representing a set of ontologies and their versions in a quad store.

Specifically, this paper makes the following contributions:

- We analyze the anatomy of a large-scale ontology repository with respect to the frequency of updates and the structure of imports among the ontologies
- We compare the storage requirements for representing the ontologies and their versions in a quad store, analyzing the effect of materializing imports.

2 Background

In this section, we provide background on the quad store technology, and its relationship to the SPARQL query language. We discuss the use of these technologies to represent and query ontology metadata.

2.1 Quad Stores, Named-Graphs and RDF

The Resource Description Framework (RDF)[2] is the core standard for representing data on the Semantic Web. The Semantic Web community is promoting RDF stores (or *triple stores*) as the key data storage technology. In an RDF store, data are not bound to a schema. Furthermore, we can assert data into an

[1] `http://bioportal.bioontology.org`
[2] `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`

RDF store directly from RDF sources where it is represented in native Semantic Web formats (e. g. RDF/XML or Turtle files). SPARQL is a query language designed to express queries for data viewed or stored as RDF. SPARQL introduces the notion of Named Graphs [4, 5] to represent and encapsulate distinct RDF Datasets.[3]

The notion of Named Graphs gave rise to the development of *quad stores*. A quad, unlike a triple, has a fourth component that often is referred as "named-graph," "context" or "model." In this paper, for the sake of consistency, we will always refer to the fourth component as named-graph. Today, there are numerous RDF databases that in essence are quad stores due to their native support for named-graphs. Examples of these systems include 4store [8], Virtuoso [7], Jena TDB [11] and Sesame [3]. These tools, and many others, have adopted SPARQL as the language to query RDF graphs. SPARQL 1.0 defines a standard mechanism for querying named-graphs. This type of query can be achieved with either `FROM NAMED` or `GRAPH` clauses. There are two widely accepted formats to represent named-graphs in RDF: N-Quads [6] and TriG [2]. Most quad stores are compatible with these two formats. However, neither of these formats is an RDF standard. Indeed, graph identification in RDF was listed as one of the core working items in the RDF Next Steps Workshop and, possibly, it will be standardized as part of the next RDF specification.[4]

2.2 Metadata, Versioning, Named-Graphs and SPARQL

Named graphs can be used in two ways. First, we can use them to represent statements about the same resource that were made in a different context. Each named graph contains statements that were made in a specific context (e.g., by a particular information source). Second, we can use named graphs to collect metadata and attach it to a document, in our case to attach metadata to an ontology.

BioPortal keeps several versions of each ontology. For each version, it holds metadata about its content, authorship, creation and modification dates, projects, categories, and so on. It is important that our backend infrastructure enables us to combine queries that filter information based on the metadata about both the ontologies and the content of the ontologies themselves. Quad store technology is the clear option to implement this kind of capability.

In quad stores, IRIs are used to identify graphs such that other statements can "say" things about graphs. This auto-referenceable mechanism allows us to incorporate metadata. As an example, the quads below—in TriG format—show how we can say things about a graph. In this example, we use the IRI `GraphX` to identify the named-graph that holds the ontology containing the term "Aromatic Amino Acids." Similarly, `GraphXMetadata` is the IRI where triples that say things about `GraphX` are held. This is just a simple example and is not necessarily the model that we will use to represent metadata for ontologies.

[3] `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/#rdfDataset`
[4] `http://www.w3.org/2011/rdf-wg/wiki/TF-Graphs`

```
:GraphX {
  :AromaticAminoAcid rdf:type owl:Class;
                     rdfs:subClassOf :SpecificAminoAcid, :AminoAcid;
                     rdfs:label "Aromatic Amino Acid" .
    (... some other ontological axioms in RDF/Turtle ...) }

:GraphXMetadata {
  :GraphX rdf:type owl:Ontology;
          dct:created "2001-09-01"^^xsd:date;
          rdfs:label "Amino Acid";
          dct:format "OWL";
          foaf:primaryTopic <http://bioportal.bioontology.org/ontologies/1054>;
    (... some other metadata statements ...) }
```

This technique has become very popular for RDF graph versioning. We could state things such as `X` is replaced by `Y` or that `Z` is a version of `T`. The `data.gov.uk` project uses a similar versioning model using named graphs [14].

As you can see, named-graphs are first-class objects in a quad store and the IRIs used to identify graphs become part of the global graph and can thus be retrieved with SPARQL queries. The SPARQL 1.0 specification defines two query clauses to filter and/or bind specific named-graphs. These clauses are `FROM NAMED` and `GRAPH`. For instance, with the following query, we would retrieve all the ontologies `?ont` where a triple has a binding on subject or object to `:SpecificAminoAcid`. The query also selects the predicates `?p` that match such bindings.

```
SELECT DISTINCT ?ont ?p WHERE {
    GRAPH ?g {
        { ?x ?p :SpecificAminoAcid . }
        UNION
        { :SpecificAminoAcid ?p ?x . }
    }
    ?g rdf:type owl:Ontology .
}
```

The `GRAPH` clause can be used both with query variables or constant IRIs depending on whether or not we need to match a specific named-graph or just filter out named-graphs that do not satisfy other conditions. `FROM NAMED` is used only to declare a set of graph IRIs where a set of triple patterns is going to be satisfied. Combining `GRAPH` and `FROM NAMED` is possible, and we can constrain queries to a set of graphs and still identify the graph which is the source of information.

As an example, the following query is applied over an RDF dataset that contains one default graph and two named graphs. Moreover, in the `WHERE` clause we obtain the graph where things of type `:SpecificAminoAcid` are bound. This query reflects the case where we can combine queries that retrieve both metadata and data about the ontologies.

```
SELECT ?g ?label_g ?s ?label_s
FROM :GraphMetadata
FROM NAMED :GraphX
FROM NAMED :GraphY
WHERE {
   GRAPH ?g {
        ?s rdf:type :SpecificAminoAcid .
        ?s rdfs:label ?label_s .
   }
   ?g rdfs:label ?label_g .
}
```

The next section describes BioPortal's architecture, its limitations and some reasons to motivate the replacement of the current backend technology with a more scalable solution.

3 Motivation

Two different motifs drive this research:

1. Scalability: The BioPortal backend interacts with several databases and APIs (Section 3.1). This variety of non-compatible sources makes it extremely hard to implement a uniform query interface across ontologies.
2. Provenance and Versioning: Currently, BioPortal treats every ontology as a whole, including the materialization of all the `owl:imports`. With this approach, it is very difficult to track provenance and to keep several versions of ontologies at the term/class level.

3.1 BioPortal Architecture

BioPortal is an open library of biomedical ontologies that may be accessed using a Web-based user interface or RESTful Web services.[5] The Web-based user interface allows users to browse, search, and visualize ontologies and also facilitates community participation in the ontology lifecycle, including providing reviews of ontologies, mappings between terms, comments, and new term proposals.

This Web-based user interface is driven by a variety of RESTful services, including services that expose information about terms in ontologies, mappings, notes, and metadata about the ontologies themselves. This metadata includes the information on who uploaded each ontology into the system, which authorities support it, the size of the ontology, and other information that is specific to BioPortal. These RESTful services are accessed using standard HTTP verbs (POST, GET, PUT, DELETE) that are roughly equivalent to create, retrieve, update, and delete operations.

Users of these services are able to access all the ontology formats using a standard XML output, which is one of the major advantages of using BioPortal

[5] `http://bioportal.bioontology.org`

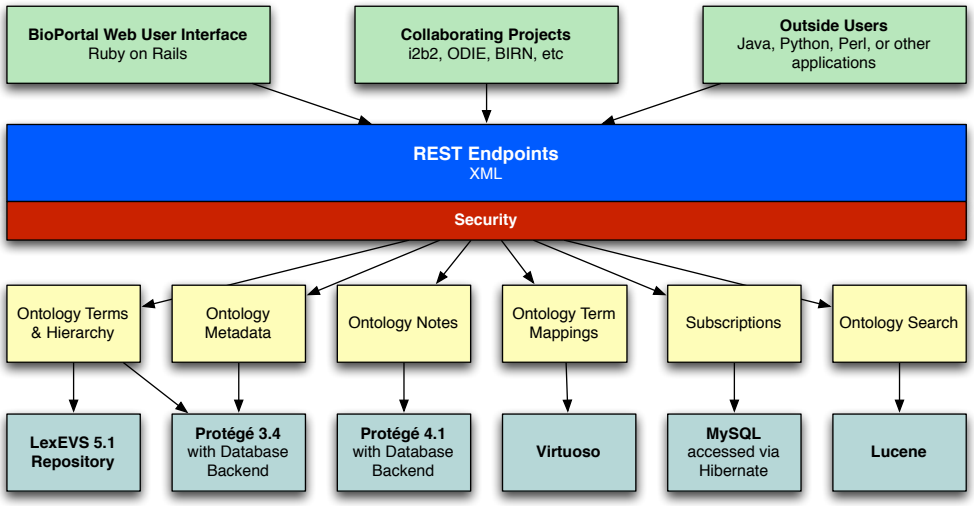

Fig. 1. BioPortal Architecture

over other systems that provide access to ontological content. Unfortunately, supporting the variety of ontology formats and related artifacts (mappings, notes, metadata) adds several layers of complexity over a traditional RDBMS solution.

In order to support multiple ontology formats, BioPortal currently utilizes two applications, LexEVS and Protégé. LexEVS is responsible for parsing and storing terminologies in formats that are primarily used in the biomedical domain: OBO Format and Rich Release Format (RRF). Protégé handles OWL, OWL2, and Protégé Frames (Figure 1). The systems use two entirely different models for storing information: Protégé relies on a RDBMS solution; LexEVS uses a combination of RDBMS and file-based storage.

In addition to ontology content, we also track a set of 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 [10]. The metadata itself are a set of instances in this OWL ontology. Thus, we ultimately use the Protégé database backend as a store for metadata information. Protégé provides a flexible model for working with data and is well-suited for integrating and extending existing ontologies, such as the Ontology Metadata Vocabulary (OMV) [12]. Unfortunately, Protégé lacks some functionality that is generally accepted as standard on quad storage platforms, including the ability to query easily across records in different ontologies and scalability into the tens-of-million triples range.

Therefore, as you can see in Figure 1, there is no single uniform storage for the ontologies or their metadata. We implement this uniform layer through an API.

However, as the amount of information in BioPortal grew, we faced scalability issues with this approach.

We needed to represent 4.5 million term mappings between terms in BioPortal ontologies. Each mapping connects two terms from different ontologies, source and target. Each mapping also has several metadata properties associated with it. Protégé was not scalable enough for this dataset and thus we decided to represent the mappings in a quad store. Quad stores provided us with the same kind of flexibility that Protégé offers with the added ability to scale easily to support millions of triples and to provide opportunities for members of the Semantic Web community to query our data using a SPARQL endpoint.

However, using a quad store only for mappings turned out to have its own scalability problems: The triples that represent mappings are completely isolated from the storage of the ontologies themselves in Protégé and LexEVS. The mapping triples had only the URIs for the source and target terms. Thus, if the user needed labels, or any other information about these terms, a single query in the system that generated a result set with ten mappings needs at least 20 queries in order to correlate information about related terms. The BioPortal Web-based user interface shows 100 mappings on each page, and therefore at least 200 calls to the RESTful services are required to retrieve term information for a single page. While we do cache results, the experience can be less than ideal for users.

The same type of issues arise when examining systems for providing ontology notes, provisional terms, or any information about ontology content that isn't restricted to a single ontology, such as a query that looks for terms that share an ID across all ontologies. We implemented a Lucene-based index to overcome some of these issues, which will remain an integral part of the BioPortal architecture due to its speed in performing free-text searches. However, we clearly need a more comprehensive, scalable, and uniform solution.

Moreover, it is worth mentioning that both BioPortal's UI and REST API have been growing in traffic over the last three years. The number of page views has quadrupled between Q2 2009 and Q1 2011 (see Figure 2, top chart). Furthermore, BioPortal users proved to be very loyal and use the service frequently. Almost 45% of them visit the site at least 15 times per month and 18% return to the site 50 times or more (see Figure 2, bottom chart). The use of the REST services has experienced outstanding growth in 2011. The average number of hits per month grew from 3M hits in 2010 to 122M hits in 2011. [6] Our commitment to serve this growing demand is one of the primary motivations to undertake the project of migrating towards a more scalable infrastructure.

[6] We consider the average hits per month in order to be able to compare 12 months from 2010 versus 7 months in 2011.

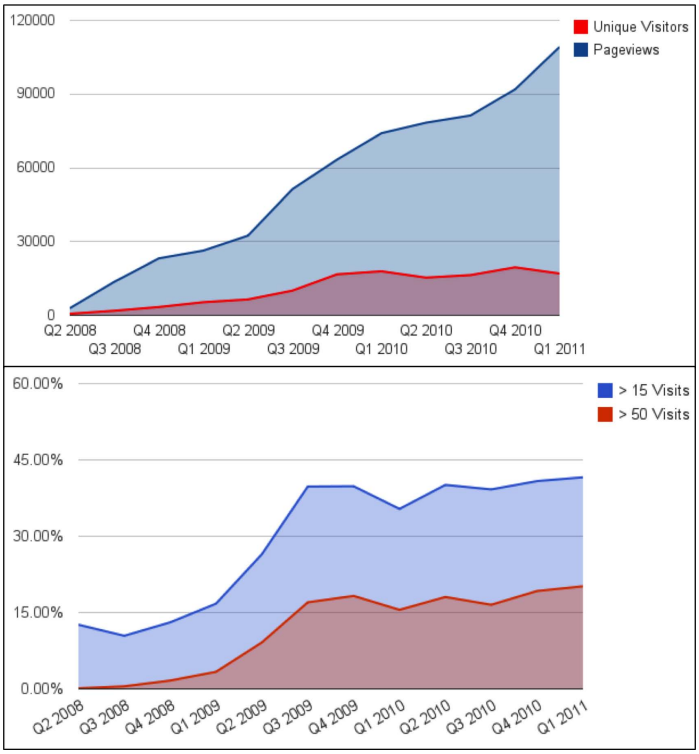

Fig. 2. BioPortal UI Traffic (per quarter, excluding Stanford) (top) and Returning visitors (bottom)

3.2 BioPortal Ontologies

At the time of this writing, there are 284 ontologies in BioPortal. As we mentioned in Section 3.1, the BioPortal ontology repository supports various ontology formats. Table 3 shows the number of ontologies per format. You can see that 90% of the ontologies are either in OBO or OWL formats. Of that 90%, 59% are in OWL and 41% in OBO.

OBO has been the format used for many of biomedical ontologies. Recently, however, there has been a shift and today OWL is the most popular format. This shift is due to the standardization of OWL and the rise of standard tools such as reasoners, query engines and editors. Moreover, there are tools that allow us to transform OBO ontologies into OWL—such as the OWL API [9]—and therefore use RDF serializations to store OBO ontologies in a RDF database. This study covers both OBO and OWL; we leave the rest of the formats for future analyses.

In BioPortal, the metadata associated with each ontology follows the BioPortal Metadata Ontology.[7] This ontology is used to declare projects, authorship,

[7] `http://www.bioontology.org/wiki/index.php/BioPortal_Metadata`

Table 1. Ontology Format Occurrences in BioPortal

Format	N. Ontologies
OWL	149
OBO	105
RRF	27
PROTEGE	2
UMLS-RELA	1

categories, coding schemes, users, etc. The details of classes and properties in this ontology are not relevant for this analysis; we will only look into the overall number of triples that we require for an ontology version. The average size of the metadata graph for the 284 ontologies is 38 triples without significant variability (94% of the ontologies are in the range [34,42]). In BioPortal, practically all the metadata are attached to an ontology version. Users can modify nearly every piece of metadata every time they submit a new ontology version. Thus, we will associate the cost of metadata triples to ontology versions rather than to the general concept of ontology.

There are large ontologies, such as the Gene Ontology (GO) with 73K terms in 738K triples, that get updated every day.[8] All of the versions of the Gene Ontology would expand to more than 350M triples. Today, we are not able to provide all the repository functions for every ontology version. Only ontology download is available for old versions. Capabilities such as term search and hierarchy visualization are accessible only for the latest version of each ontology.

For versioning, BioPortal deprecates ontologies using the Apple's "Time Machine" algorithm: keep all versions for the last 7 days; keep one most recent version per week for the last 30 days; keep one most recent version per 30 days from the beginning.[9] This logic applies only to ontologies automatically imported from remote locations. BioPortal keeps all manually submitted versions, because the number of versions of this kind is small.

Currently, BioPortal treats every ontology as a whole, including the materialization of all the `owl:imports`. Thus, if a small ontology imports a large one then the former becomes a large ontology. If we take into account that this problem gets reproduced for every version of that small ontology then we can end up with a very inefficient model that contains millions of duplicated items. Moreover, this model makes it very difficult to retrieve term provenance since many terms are duplicated in graphs where they do not originally belong.

Our hypothesis was that we could optimise the number of quads in the system by using a more granular model where `owl:imports` are not materialized and every ontology graph only contains its own RDF triples without the triples from the `owl:imports` ontologies. This type of granular model would economize the number of triples in our data store and, at the same time, enable provenance

[8] `http://bioportal.bioontology.org/ontologies/1070`
[9] `http://en.wikipedia.org/wiki/Time_Machine_(Mac_OS)`

retrieval. Figure 3 shows the difference between an **import-materialized** model versus an **ontology-per-graph** model.[10]

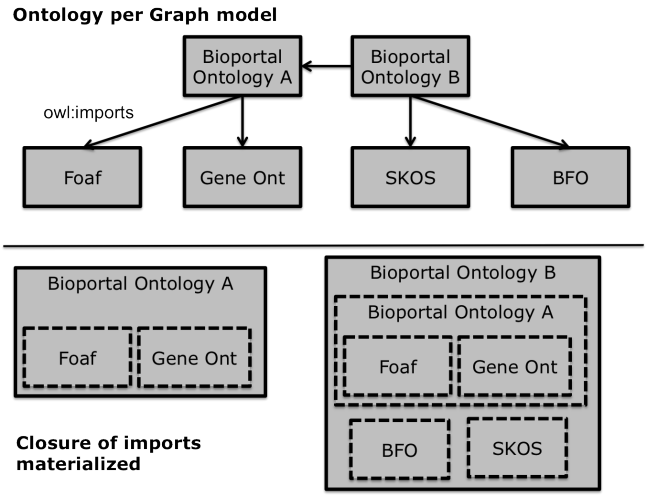

Fig. 3. Models: Per Graph Model vs Materialization of `owl:imports`

The upper part of Figure 3 depicts the import structure maintained by using named-graphs. The bottom part shows how ontologies are incorporated into other ontologies and how this model provokes a redundancy explosion. In order to see quantitatively the level of quad storage space optimisation that we can gain with an ontology-per-graph model we need to analyse the occurrences of `owl:imports` in the BioPortal repository.

In the next section, we analyse the size of BioPortal's data store in triples and the distribution of `owl:imports` in OWL ontologies.

4 Quad Economic Analysis

We estimate the cost, in number of quads, of asserting all the OWL ontologies into a quad store. We also look at the scalability numbers in case that maintaining all versions of each ontology becomes a requirement.

OWL and OBO formats use different mechanisms for import. In order to incorporate ontologies in the OBO format into the import analysis we need to study further the compatibility of OBO imports with OWL imports. Thus, the numbers presented in Figures 4, 5 and 6 include only statistics about imports for ontologies in OWL format.

[10] *Import-materialized* and *ontology-per-graph* are key terms and we shall refer to them in following sections.

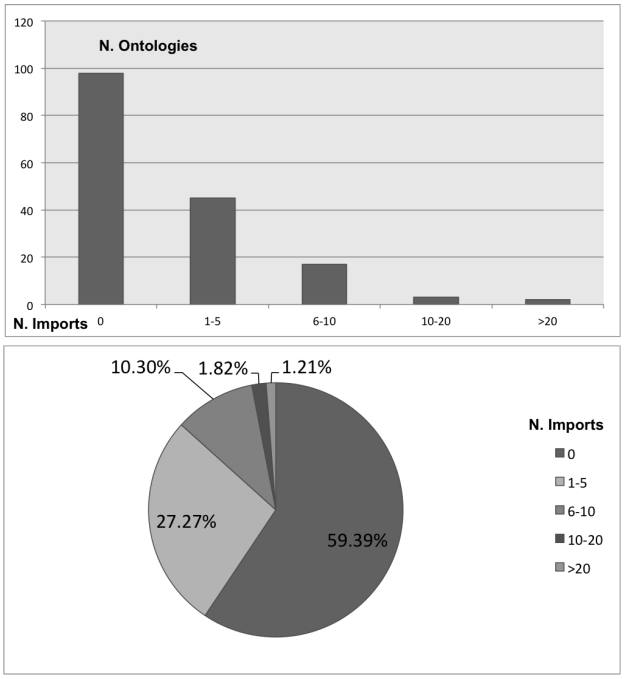

Fig. 4. Number of Imports per Ontology. Absolute numbers at the top, percentages at the bottom.

One of the questions to be answered is the optimisation ratio—in number of triples—when using an *ontology-per-graph* model rather than a *closure-materialized* model (see Section 2). This question is closely related to the number of `owl:imports` in our catalog. The `owl:imports` distribution gives us a mechanism to understand the level of reusability in the data store.

There are 299 ontologies in the import closure of the 149 OWL ontologies in BioPortal (i.e., if we follow all the `owl:imports` links from the 149 ontologies, we will create a set of 299 ontologies). These 299 OWL ontologies contain 303 `owl:imports`, the materialized import closure is a set of 495 `owl:imports`. Not all these 299 ontologies reside in BioPortal as first class citizens. Many of them are just pushed in the system as consequence of an `owl:import`. We do not have metadata for these ontologies. These ontologies could include FOAF or SKOS which are imported by several other ontologies but are not necessarily biomedical ontologies and thus they have not been submitted directly to BioPortal. From the 299 OWL ontologies in the study 165 are ontologies registered in BioPortal.

Figure 4 presents absolute numbers (top chart) and the percentages (bottom chart) of ontologies that have none, one, two or more than two imports. These numbers—from Figure 4—are not really conclusive by themselves, even though we can identify some tendencies. In order to extract conclusions, we also need

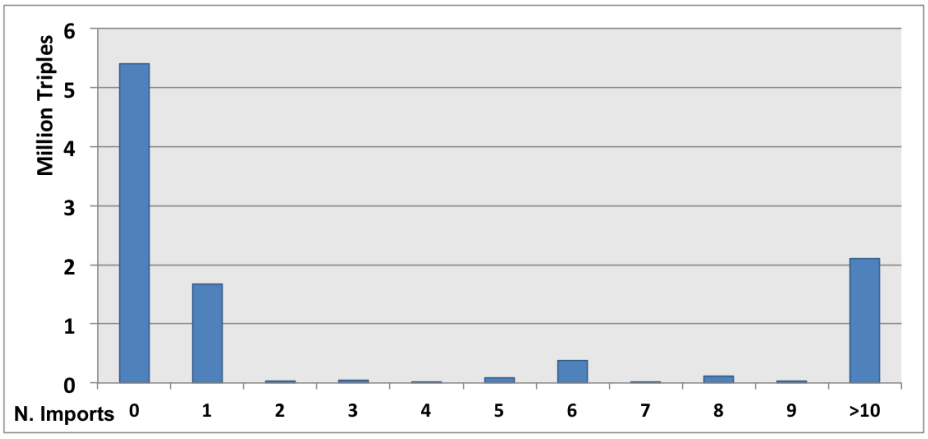

Fig. 5. Closure-Materialized Number of Triples Sum per Number of Imports. Number of imports on X Axis and number of triples on Y Axis

to know the ratio of reused triples. Figure 5 shows the distribution of number of triples, closure materialized, per number of imports. Ontologies that have no imports gather 5.4M triples in the system; ontologies with just one import 1.7M; none of the categories with imports from 2 to 9 reach 0.5M triples. For ontologies with more than 10 imports there is tendency change with a peak of 2.1M triples generated.

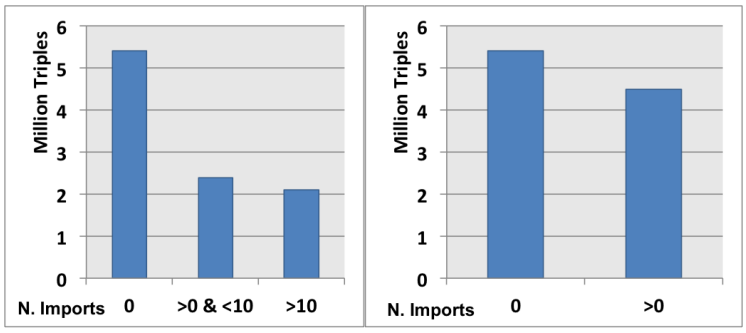

Fig. 6. Closure-Materialized Number of Triples Sum per Number of Imports Grouped. Number of imports on X Axis and number of triples on Y Axis.

Figure 6 groups the data presented in Figure 5 into two different charts. The left-hand side groups the number of triples into three categories: (a) no imports, (b) more than 1 and less than 10 and (c) more than 10. The right-hand side shows

the same number grouped in two categories, the ones with imports and the ones without. These two graphs are used in Section 5 for qualitative discussion.

In addition to the previous statistics we also measured various indicators per ontology. Table 2 shows the number of triples without materialized imports (T. Without C.), number with materialized imports (T. With C.), number of imports in closure (N. Import C), depth of tree closure (Depth TI), triples import ratio (I Ratio) and number of versions (NV). For practical reasons we only show a few records in Table 2, but it is worth mentioning that similar numbers have been collected for the 299 OWL ontologies. Note that the last column "NV - Number of Versions" only applies to BioPortal ontologies and not to ontologies fetched from the Web as part of an import. To calculate the ontology import ratio we simply calculate the percentage of triples brought via `owl:imports` against the ones held directly in the ontology, such that:
$import\_ratio = 1 - (T.\ Without\ C./T.\ With\ C.)$

Table 2. Statistical Indicators Sample - Ontologies with more than 10 imports.

Ontology	T. Without C.	T. With C	N. Import C	Depth TI	I Ratio	NV
NIF.owl	6	1,795,951	58	4	99.9%	9
sopharm.owl	1,317	359,087	24	4	99.6%	4
vo.owl	40,804	45,557	16	2	10.2%	92
aero.owl	698	3,981	14	2	82%	8
OntoDM.owl	4,685	8,240	11	4	43%	1
ddi.owl	2,388	5,347	10	2	55%	1

Table 3. Number of Triples and Versions for OBO and OWL Ontologies

	I. Not Mat.	I. Mat.	N. Versions	All V. Not Mat.	All V. Mat.
OWL	9.75M	14.65M	817	41.2M	65.6M
OBO	N/A	5.9M	3,022	N/A	593M
TOTAL	N/A	20.55M	3,839	N/A	658.5M

The last indicative figures are shown in Table 3. This table includes:

- Number of triples without materialized imports (I. Not Mat.).
- Number of triples with materialized imports (I. Mat.).
- Number of versions (N. Versions).
- Number of triples for all the versions of all the ontologies without materialized imports (All V. Not Mat.).
- Number of triples for all the versions of all the ontologies import materialized (All V. Not Mat.).

After having presented several important aspects related to the data size in triples and the `owl:imports` distribution, we shall discuss how they impact the design and scalability of our quad storage in the next section.

5 Discussion

Our initial hypothesis was to assume that we could optimise the number of quads in the system by using a more granular model where we do not materialize `owl:imports` and every ontology graph only contains its own RDF triples without bringing the triples from the `owl:imports` ontologies. In that sense, for the 165 biomedical ontologies in OWL, Figure 4 shows a steep tail distribution where 59% of the ontologies do not make use of `owl:imports` at all. Therefore, they do not reuse any terms from other ontologies. The rest of the ontologies (40.6%) use imports but the majority of them (27.27%) contain a small number of imports, 5 imported ontologies or less. Ontologies with high levels of imports are rare in our catalog, just 3.03% have more than 10 imports and from those only 1.21% have more than 20. These figures let us conclude that the levels of reusability in OWL ontologies for the biomedical domain is low. We do not try to explain why this situation happens, but, arguably, the experts in the biomedical domain have adopted ontologies earlier than many others [13]; if do not often practice ontology reuse, works in other domains have an even longer way to go.

Next, we look more closely at what type of ontologies reuse other ontologies, in order to determine the effects in terms of storage requirements. Because most ontologies are self contained and they do not reuse other ontologies, even in the model where we do not materialize imports, not many ontology graphs will refer to other ontology graphs. However, if the 303 `owl:imports` are placed in the right place, then the *ontology-per-graph* model would optimize the dataset size. In an *ontology-per-graph* model, the ideal situation is to have the majority of the imports in the larger and most frequently updated ontologies. The rationale behind this conclusion is that, not all modules in a new version of an ontology would change. In those cases we only need to assert a new version for the updated modules and modify the metadata. Thus, we look at the relationship between ontology sizes, number of imports and update frequency.

Figures 5 and 6 give us the relationship between ontology size, in number of triples, and the level of modularity, in number of imports. Contrary to what one might expect, the larger the ontology, the fewer imports it has. In other words, larger ontologies that could benefit more from modularity actually are not modular in practice. The exception is the peak for "> 10" category in Figure 5, which has 2.1M triples. This peak is caused by two very modular ontologies that are also large, `NIF` and `sopharm`, with 58 and 24 imports respectively; and 1.7M and 300.5K triples respectively (see Table 2). However, ontologies like these are an exception. Within the 14 largest ontologies in our collection, all of them with more than 100K triples, `NIF` and `sopharm` are the only ontologies with more than 2 imports and with more than 10% of reused import ratio. Thus, we can conclude that the size of the ontology is not directly related to the number of imports.

Next, we look at the absolute numbers for ontology sizes (Table 3). We can calculate the percentage of saved triples for the latest ontology versions if we materialize the imports, $1 - (I.\ Not\ Mat./I.\ Mat.)$. The percentage obtained is 33.4% for the last snapshot and, for all versions, it goes up to 37.2%. These per-

centages show that, even though the level of re-usability in the OWL ontologies is very low, we can still save more than one third of the storage by moving to an *ontology-per-graph* model. Moreover, that model also provides the requirements to track provenance and implement versioning.

One of the columns in Table 3 also shows the size of the OBO ontologies, but only for the import-materialized case. The last snapshot of the OBO ontologies contains 5.9M triples. However, historically BioPortal has a larger number of versions (3,022) for OBO ontologies than for OWL because they get updated more frequently and also because OBO was the predominant format in the past. This large number of versions makes the total number of triples to be 593M. We hope that, once OBO ontologies use a more explicit import mechanism or their developers switch to OWL, we will be able to save on the storage at a ratio that is similar to the one that we observed for OWL Ontologies.

One of the motivations to maintain an *import-materialized* model is to facilitate SPARQL query construction. There will be cases in which we need to run queries on a set of ontologies —and not over the global graph. In an *import-materialized* model, one ontology is reflected by one graph only, and thus queries targeting specific ontologies are easy to write. In a *ontology-per-graph* model, one ontology is reflected by a closure of tree imports, which can potentially complicate the query construction. It is possible to argue that, depending on the complexity of the `owl:imports` network, it might not be feasible to construct queries with hundreds or thousands of `GRAPH` or `FROM NAMED` clauses. You could even argue that for a very large `owl:imports` network the size of the SPARQL query string could become problematic. Even though all these arguments are based on solid foundations, they would not be valid for BioPortal's case. The statistics collected in Section 4 show that:

1. 59% of the ontologies would be reflected by just one graph, because they do not have imports.
2. The largest import closure contains 58 items, which is far from being the potential cause of a problem. 58 graph clauses to construct an RDF data set should not be a problem for any of the quad stores listed in Section 2.
3. The maximum depth of the import tree is 4 levels, and thus calculating the import closure will never steal many computational cycles. Moreover, the number of imports in the system makes it feasible to pre-cache all of them at the application level. With this approach, the application would not have to go back and forth to the SPARQL endpoint in order to calculate the closures.

To conclude, our analysis shows that, on the one hand, ontology reuse is still far from being the norm, but, on the other, effective reuse is a goal worth pursuing: as ontologies become larger in size, the level of reuse can have significant implications for the scalability of the ontology storage systems.

6 Acknowledgments

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

References

1. Noy, N.F., Shah, N., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Montegut, M., Rubin, D.L., Youn, C., Musen, M.A.: Bioportal: A Web Repository for Biomedical Ontologies and Data Resources. In: International Semantic Web Conference (Posters & Demos) (2008)
2. Bizer, C., Cyganiak, R.: The Trig Syntax. Tech. rep., FU Berlin (7 2007), http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/Spec/TriG-20070730/
3. Broekstra, J., Kampman, A., Harmelen, F.V.: Sesame: A Generic Architecture for Storing and Querying RDF and RDFS. pp. 54–68. Springer (2002)
4. Carroll, J.J., Bizer, C., Hayes, P.J., Stickler, P.: Named Graphs, Provenance and trust. In: WWW. pp. 613–622 (2005)
5. Chein, M., Mugnier, M., Simonet, G.: Nested Graphs: A Graph-Based Knowledge Representation Model with FOL Semantics. In: Proceedings of the 6th International Conference on Knowledge Representation (KR'98. pp. 524–534. Morgan Kaufmann (1998)
6. Cyganiak, R., Harth, A., Hogan, A.: N-Quads: Extending N-Triples with Context. Tech. rep. (2008), http://sw.deri.org/2008/07/n-quads/
7. Erling, O., Mikhailov, I.: RDF Support in the Virtuoso DBMS. In: CSSW. pp. 59–68 (2007)
8. Harris, S., Lamb, N., Shadbolt, N.: 4store: The Design and Implementation of a Clustered RDF Store. In: Scalable Semantic Web Knowledge Base Systems - SSWS2009. pp. (p. 94–109) (2009)
9. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL Ontologies. Semantic Web 2(1), 11–21 (2011)
10. Noy, N.F., Dorf, M., Griffith, N., Nyulas, C., Musen, M.A.: Harnessing the Power of the Community in a Library of Biomedical Ontologies. In: Workshop on Semantic Web Applications in Scientific Discourse at the 8th International Semantic Web Conference (ISWC 2009). Chantilly, VA (2009)
11. Owens, A., Seaborne, A., Gibbins, N., schraefel, mc: Clustered TDB: A clustered triple store for jena (November 2008), http://eprints.ecs.soton.ac.uk/16974/
12. Palma, R., Hartmann, J., Haase, P.: OMV: Ontology Metadata Vocabulary for the Semantic Web. Tech. rep., http://ontoware.org/projects/omv/ (2008)
13. Rubin, D.L., Shah, N.H., Noy, N.F.: Biomedical Ontologies: a Functional Perspective. Briefings in Bioinformatics 9(1), 75–90 (2008)
14. Sheridan, J., Tennison, J.: Linking UK Government Data, pp. 1–4. ACM Press (2010), http://events.linkeddata.org/ldow2010/papers/ldow2010_paper14.pdf