Slicing Linked Data by Extracting Significant, Self-describing Subsets: The DBpedia Case

Michele Minno[1], Davide Palmisano[2], and Michele Mostarda[2]

[1] Pro-netics
michele.minno@pronetics.it
[2] Fondazione Bruno Kessler
{palmisano,mostarda}@fbk.eu

Abstract. The Linked Data cloud is a huge set of data sources, the same objects being instances of many different ontologies and being described by different overlapping concepts and properties. This paper shows an innovative approach to represent in an unambiguous and homogeneous way instances of such large, highly unpredictable, linked ontologies. Instead of making use of external, static and ad-hoc devised knowledge bases, the approach followed in this paper leverages an existing ontology in the Linked Data cloud to univocally identify all the instances of all linked ontologies. Following this kind of approach, different views over the same set of instances can be devised, depending of which spot of the cloud we choose to see things through.

Keywords: LinkedData, Ontology, SKOS.

1 Introduction

The Linked Data cloud is a set of interlinked, loosely-coupled ontologies in a continuos incremental expansion. The positive aspects of this phenomenon are huge and easily understandable: knowledge formal representation grows along with their users', reflecting the typical proper nature of the Web, being a place where 'things happen', more than where things happened elsewhere are registered and flatly listed. The drawback of all this is that an actor (typically: a human through a software interface and service) who would like to access this big multi-layered knowledge base will easily loose the grasp over it. The approach here presented allows to get a partition of the whole ontology set, according to a reasonably fixed subset of it. This partition, informally depicted as a 'set of slices', adds a semantically meaningful description of the resources contained in the Linked Data cloud (the instances, we can say), because obtained through a subset of the Linked Data cloud itself. The use case explored in the following of the paper relates to subjects assigned to things, like 'actor' or 'business man' to a person. With the algorithm here presented we show how to obtain a partition of all Linked Data instances in terms of subjects, in which thus every instance has got a single meaningful subject assigned to it. For prototyping purposes the DBpedia RDF ontology is used, more specifically a relational view of it. The rest of the paper is structured as follows. Section 2 describes the problem while section 3 justifies the adopted solution and details its main aspects, jointly with the algorithms used. Section 4 suggests some variants and applications. Conclusions and acknowledgements close the paper.

2 The Linked Data Slicing: Problem Statement

The actual Linked Data slicing problem can be reduced to the simpler one of partitioning a set of points placed in an n-dimensional space in a set of m equivalence classes (see figure 1). In mathematics, the equivalence class of an element a in the set X is the subset of all elements in X which are equivalent to a. In our case the set X comprises all Linked Data resources. We'll constrain ourselves to consider just DBpedia[1] for sake of simplicity and because it is one of the biggest and referred ontology within the Linked Data cloud.

The initial condition of the problem is the chaotic situation where each DBpedia resource belongs to several classes (not equivalence classes, but just sets), by means of being instance of several concepts and participating in many properties, which belong in a scattered fashion to different ontologies in the Linked Data cloud, according to the Linked Data best practices. The resources are the points represented in the cloud of figure 1, ideally standing for the whole Linked Data cloud. The 3-dimensional space stands for the different ontologies describing the same instances from different point of views, like geo-location, musical genres, and so on. Our aim is to see these points through a filter that is simply one of the ontologies of the cloud, represented in the figure by the external rectangle. This rectangle is the partition we want to achieve, where each point is of a certain color (concept or property value), even though in the chosen ontology in the cloud it participates of many different colored sets. As a first step of our approach, a subset of this Linked Data cloud has to be chosen, in order to divide the resources set in equivalence classes. The mentioned subset must be something that applies to the greatest number of resources, ideally to all of them. So it has to be quite generic and not narrowed to a specific domain. In the DBpedia case, we chose SKOS[2]. SKOS stands for Simple Knowledge Organisation System. It is a very light model to express relations between concepts, basically 'broader' or 'narrower' relations. So if a set of concepts has a very straight hierarchy it would be easily modeled by a SKOS concepts scheme. Each DBpedia resource has at least a SKOS subject associated to it through the *skos:subject* property that links a resource to a subject. Some resources have even several tens of subjects, from the most generic (i.e.: Living_people) to the most specific (Presidents_of_the_United_States, for the resource: Barack_Obama). As it is within the Linked Data cloud, this SKOS categorization is simply another layer describing

[1] http://dbpedia.org/

[2] A good guide can be found at `http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102`

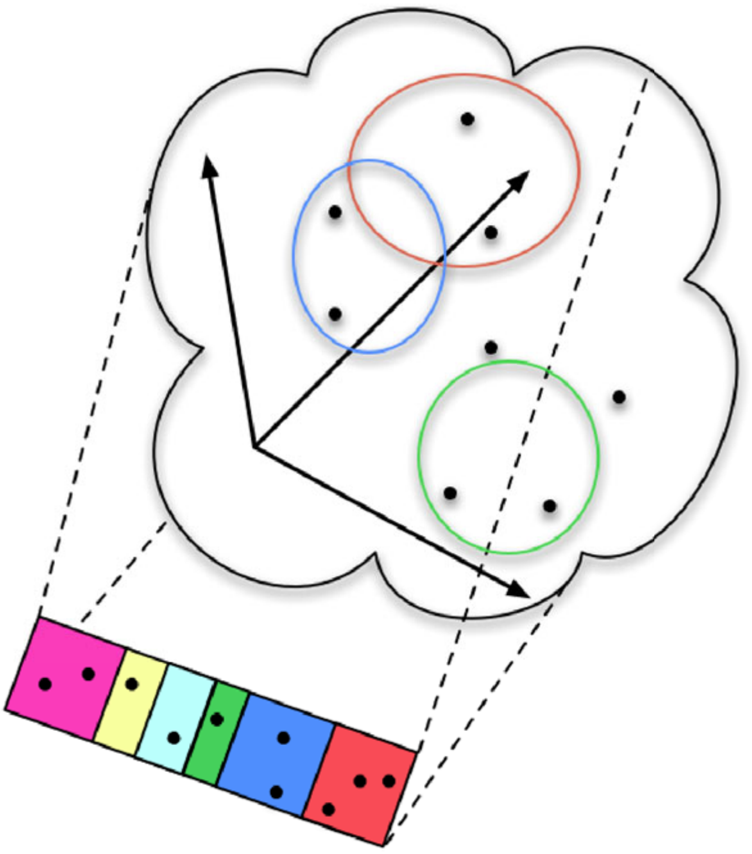

Fig. 1. The 'Slicing' problem

the DBpedia resources, and it doesn't gives back much information. The situation is well sketched in the figure 2. To have the whole DBpedia resource set divided in equivalence class we have to achieve instead the situation sketched in figure 3, in which each resource belongs to exactly one SKOS subject. As we are going to see in section 4, the SKOS subject has to be chosen as the most representative one, according to an algorithm explained in the following of the paper.

Just a trivial consideration: choosing the narrowest category as the most representative one is a solution that, even if characterized by a straightforward computational complexity, often leads to undesirable results. Actually such kind of solutions don't take into consideration how the resource is linked with the rest of the data space, loosing the opportunity to discover new relationships and links between different instances and concept. In one word, they loose the opportunity offered by such linked knowledge.

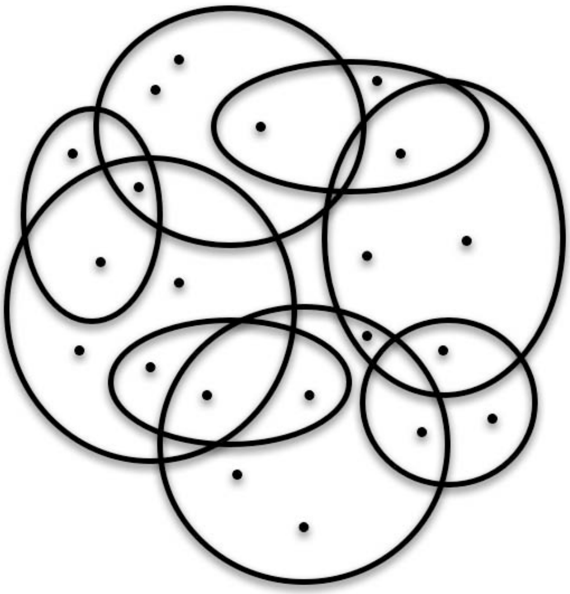

Fig. 2. The SKOS layer

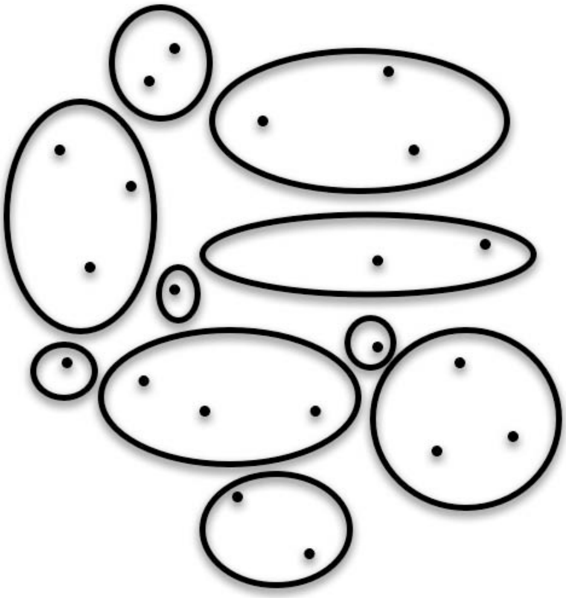

Fig. 3. The SKOS equivalence classes

3 The Relational Snapshot Solution

What we call here the relational snapshot is essentially an index built on an RDF ontology, where each tuple of the relation refers to a resource R in the ontology and can be represented as:

$$(ur, lr, us, ls)$$

where:

ur : URI (Uniform Resource Identifier) of R.
lr : the first one of the values of the property *rdfs:label* in which R participates.[3]
us : URI of the SKOS subject S describing R.
ls : the value of the property *skos:prefLabel* in which S participates.

in which *rdfs:label* is one of the core properties of RDF Schema[4], whereas *skos:prefLabel* is one of the properties of SKOS[5].

The adoption of a solution based on an index natively stored in a relational model is motivated mainly by the following considerations:

Responsiveness of the adopted solution - The values composing each tuple described above could be also on-the-fly computed by a series of SPARQL[6] queries, without needing to be stored in a relational structure. Still, this holds in principle, but since the result of this slicing process has in a Web based environment its natural use and applications, the responsiveness raises a serious issue about the scalability of any adoptable solution. It comes obvious that a 30-years old consolidated technology is still the most suitable to address such problems, also from a mere computational point of view. Accessing a relational structure has a $LOGSPACE$ complexity (this is the complexity of evaluating a First Order Logic query, also expressible in SQL, over a database[1]), while accessing a graph-like structure as RDF by query languages as SPARQL is a *PSPACE*-complete problem[2], thus leading to intractable query evaluations.

Transparency - The relational view allows the potential end users of any application of the slicing process to access the indexed ontology without regarding how it is effectively stored and structured.

Index adaptability - Every change in the ontology subjected to the slicing should reflect onto the index itself. The adopted solution allows any modification of the relational view simply accessing the stored tuples with SQL. RDF resources cancellation or new insertion could be resolved trivially by one single SQL delete or update instruction on the index.

[3] The first one is usually the most used.
[4] The prefix *rdfs* stands for: *http://www.w3.org/2000/01/rdf-schema#*
[5] The prefix *skos* stands for: *http://www.w3.org/2004/02/skos/core#*
[6] The query language for RDF: `http://www.w3.org/TR/rdf-sparql-query`

3.1 A SKOS Based Indexing

Even if the identification of the most representative SKOS subject of a certain resource seems too much related to the perception that each person could have regarding that specific resource, several assumptions could be done around the concept of pertinence of a SKOS subject measured by the degree of similarity between two resource. More precisely,

– given two resources the degree of similarity is the number of SKOS subjects that they have in common and,
– the most linked resource MLR of a given SKOS subject S is the resource that has S among its SKOS subjects and that is referred by the highest number of resources in the ontology (here referred means: having a property *P* participation like: *P(r, MLR)*, with r any other resource)

With this two roughly defined properties is possible to informally state that the most representative SKOS subject of a given resource R is the one for which the highest similarity degree between its most linked resource and R has been found. A formalization of the algorithm that computes the most representative SKOS subject of a given resource is provided in the rest of this section, using the SPARQL syntax when needed.

3.2 The Most Representative SKOS Subject Identification Algorithm

Hereby follows the formal description of the algorithms that are the foundation of our indexing techniques. Even if a formal analysis of the computational complexity is not provided, this formalization is necessary to make this dissertation more readable and concrete. However, the computation complexity of the algorithm strongly depends on the underlying algorithm used to resolve the SPARQL queries on the targeted ontology.

MRSS: The most representative SKOS subject identification algorithm

```
Input: a resource URI r
Output: a SKOS subject URI s

subjects <- SkosSubjects(r);
integer similarity <- 0;
SKOS subject URI s <- subjects[0];
foreach subject in subjects do
      mostLinked <- MostLinkedResource(subject);
      actualSimilarity <- ResourceSimilarity(r, mostLinked);
      if similarity <= actualSimilarity then
           similarity <- actualSimilarity;
           s <- subject;
      end
end
return s
```

SkosSubjects: The SPARQL query to retrieve the SKOS subjects of a resource

```
Input: a resource URI r
Output: a set of SKOS subject URIs

subjects <- ExecuteSPARQLQuery("SELECT DISTINCT ?uri WHERE ?r
skos:subject ?uri")
return subjects
```

MostLinkedResource: The function that retrieves the most linked resource of a given SKOS subject

```
Input: a SKOS subject URI subject
Output: a resource URI resource

subjectResources <- ExecuteSPARQLQuery("SELECT DISTINCT ?uri WHERE ?uri
skos:subject ?s");
mostLinked <- subjectResources.first();
linkedSize <- GetLinkedResources(mostLinked).size();
foreach resource in subjectResources do
     actualLinkedSize <- GetLinkedResources(resource).size();
     if linkedSize <= actualLinkedSize then
           linkedSize <- actualLinkedSize;
           mostLinked <- resource;
     end
end
return mostLinked
```

GetLinkedResources: The SPARQL query that retrieves all the instances linked to a given resource

```
Input: a resource URI resource
Output: a set of resource URIs resources

resources <- ExecuteSPARQLQuery(SELECT DISTINCT ?uri WHERE ?uri ?prop ?r)
return resources
```

ResourceSimilarity: The function that measures the degree of similarity between two instances

```
Input: a resource URI uri1, a resource URI uri2
Output: an integer N (0 <= N <= SkosSubjects(uri1))

similarity <- 0
foreach subject in SkosSubjects(uri1) do
     if subject in SkosSubjects(uri2) then
          similarity ++
     end
end
return similarity
```

4 Variation on Theme and Potential Applications

In this paper it has been showed how to slice the whole DBpedia using SKOS. By the way, this is just an example of partition of an ontology instances set that can be achieved. The following notation can be used to abstractly represent a partition:

$$(I, K)$$

where I is the input ontology to be sliced and K is the ontology used to slice I (metaphorically, the knife). Thus several other partitions are possible, depending on which specific view over I is desired. For example, slices on an I containing music artists can be obtained through a K modeling music genres, and so on. This slicing process can be straightforwardly implemented as a service and used in applications where real semantic[7] data coming from the Web are strongly preferred, but with the wish of avoiding all the real complexity involved in such retrieval (in terms of know-how, technologies and response efficiency). To give an example of a real application that would leverage such kind of services, an input field powered by a semantic autocompletion functionality can be devised. In such kind of field, every suggested completion of the string typed by the user would come with a category, i.e.: (Godfather - Male godparent), (Godfather - Movie), and so on. Moreover, all the suggestions can be grouped by the category to improve the response readability. As we have seen, using the defined MRSS algorithm is possible to couple each resource label with its MRSS label. This allows to achieve the autocompletion process, resolving each substring matching, as the following SQL query shows, where a $< substring >$ is matched with the label of an instance coupled with its MRSS:

$$SELECT\, label, category\, FROM\, index$$

$$WHERE\, label\, REGEXP\, ("< substring > \%")\, GROUPBY\, category$$

In this way the user is enabled to uniquely and explicitly identify instances of an ontology that represent what she/he really has in mind during typing.

5 Conclusions and Future Work

In this paper a Web of Data partition enabling algorithm has been justified, detailed and prototyped using the DBpedia and the SKOS ontologies. The deeper idea behind is that the Linked Data cloud is a special data source, somehow unique under many points of view, which are first of all its collaborative linked nature and its semantic formalization. But unfortunately it has not been so far fully exploited, its potential being hardly understood and appreciated. So this effort may be considered an attempt to see the Linked Data machinery at work,

[7] Here semantic stands for meaningful, in the most generic acceptation.

producing the high-quality data which are its own strength, but hiding or internally managing all the tricks, subtleties and efficiency lacks that prevent it from growing as a mainstream technology and solution. To enforce this idea of realizing something in the Semantic Web world that simply 'works', as other killer applications in other computer science fields did, parallelization implementations using the map-reduce algorithm[3] are under investigation.

Acknowledgements

This work has been partially founded within the NoTube Project (EU FP7 Integrated Project 231761), during research activities focused on the Semantic Web technologies application in the field of user modeling.

References

1. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data Complexity of Query Answering in Description Logics (2006)
2. Perez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL (2006)
3. Dean, J., Ghemawat, S.: MapReduce: Simplied Data Processing on Large Clusters (2004)