SimplePARQL: a New Approach Using Keywords Over SPARQL to Query the Web of Data

Sonia Djebali
De Vinci Technology Lab – ESILV
Pôle Universitaire Léonard De Vinci
Paris La Défense, France
sonia.djebali@devinci.fr

Thomas Raimbault
De Vinci Technology Lab – ESILV
Pôle Universitaire Léonard De Vinci
Paris La Défense, France
thomas.raimbault@devinci.fr

ABSTRACT

The SimplePARQL is a new and intuitive approach to query the Web of Data through existing SPARQL endpoints by using *keywords* in addition to SPARQL elements. Thus, the user is able to write more expressive pseudo-SPARQL queries where knowing the ontology (classes and properties) and resources' identifiers from an RDF base are not required. Concretely, a SimplePARQL query is transformed into N valid SPARQL queries that extend the initial query in order to reach the IRIs or literals in the RDF bases corresponding to keywords. We implemented our approach on the platform universal-endpoint.com, where SimplePARQL queries can be written and executed on different RDF bases at the same time; SPARQL queries are accepted too.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms & Methods**]; H.3.3 [**Information Search and Retrieval**]: Query formulation, Search process; H.5.2 [**Information interface**]

1. INTRODUCTION

The *Web of Data*, historically called Semantic Web [3], is a collaborative movement led by W3C [1]. It promotes common methods and approaches, computable by a machine, to store, to share, to find and to combine data. The mechanism behind the Web of Data is to use IRIs[2] as global identifiers for resources (such as people, places and organizations), and to link together the resources to provide some meaningful and useful information (*i.e.* semantics of the data) when these resources are looked up. This is given by the W3C standard called RDF that uses web technologies, like other standards for the Web of Data (*e.g.* RDFS, OWL, and SPARQL).

Nowadays, the Web of Data is made of several billions of resources, which are organized in triples, are distributed in a hundred RDF bases, and are accessible through SPARQL endpoints (publicly accessible, one per base). The SPARQL query language enables, in theory, to search and to explore the resources of the Web of Data. However, writing a relevant SPARQL query is complex and not an easy task, and often inaccessible to novice users. Indeed, the two main causes are: (i) the structure of data (*i.e.* the ontology) is usually unknown due to its high flexibility, unlike the scheme of tables in a relational database, and (ii) the identifiers of the resources (*i.e.* the IRIs) are in general unknown. Therefore, an intuitive way to access to the Web of Data becomes necessary and more important.

In the literature, face to the challenges of writing a SPARQL query, several approaches are made to explore in a user-friendly way one or more RDF bases. Among these approaches, we can cite the Visual data Web[3] [5], Sparklis[4] [4], and Freebase Easy[5] [2]. They provide an access to the resources step-by-step by selecting a type, a relationship or an individual in a graphical interface. Unfortunately, these approaches limit the exploration of the resources of the RDF bases that are previously and internally indexed (and often locally stored), with the risk of loosing data synchronization: that is the price to pay for having a friendly experience to explore data into the bases.

Our contribution is to extend the expressiveness of SPARQL to be able to query an RDF base without knowing either its structure (*i.e.* ontological vocabulary) or the resources' identifiers (*i.e.* IRIs). We aim to reduce the gap between the interpretation capabilities of a machine and how a user grasps the data. Indeed, some of those data are obvious and implied for the human, and therefore it seems not necessary to clearly explain them. To do that, we offer the possibility to write simplified and intuitive SPARQL-like queries, by using *keywords* in addition to SPARQL elements. In this way, (key) words that are human readable and understandable can replace (unknown) IRIs. We called such a query a *SimplePARQL* query. As in a SPARQL query, the triples' organization is preserved with SimplePARQL, thereby giving semantics of the elements into the query (as opposed to keywords used to query the classic Web like with Google). Concretely, a SimplePARQL query is rewritten in N "classic" SPARQL queries that extend the initial query in order

[1]World Wide Web Consortium, http://www.w3.org/
[2] As URL provides a standard way of accessing documents on the Web, an IRI – *Internationalized Resource Identifier* – provides a standard way of accessing resources on the Web of Data.

[3] http://www.visualdataweb.org,
[4] http://www.irisa.fr/LIS/ferre/sparklis/osparklis.html
[5] http://freebase-easy.cs.uni-freiburg.de/browse/

SEMANTiCS '15, September 15 - 17, 2015, Vienna, Austria
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-3462-4/15/09...$15.00
DOI: http://dx.doi.org/10.1145/2814864.2814890

to reach the IRIs or literals in the RDF bases corresponding to the keywords, while taking account of element organization into the triples of the initial query. Finally, the generated SPARQL queries can be sent to different SPARQL endpoints of the Web of Data by using existing technologies and engines like Jena[6] and Virtuoso,[7]. An implementation of our approach is available on the platform universal-endpoint.com, where SimplePARQL queries can be written and executed on RDF bases. Note that we also offer on our platform the possibility to query different bases at the same time. Note also that SPARQL queries are accepted too.

This paper is organized as follows. The section 2 presents the Web of Data environment, in term of the RDF framework and the SPARQL query language. The section 3 is dedicated to present the SimplePARQL syntax. The section 4 presents how a SimplePARQL query is extended to SPARQL queries to be computed through the Web of Data. Finally, the section 5 discusses the perspectives of our study and concluded our work.

2. Web of Data

The RDF framework provides a very simple way of formatting data that is based on a series of statements about resources. A statement relates one resource to another one via a given relation, also called property. Formally, these statements take the form (subject, predicate, object) and are known as *triples*. A triple (x, y, z) is similar to a basic sentence that means "the subject x has for property y the value z". A simple example of triple could be "Albert Einstein has for field physics". However, in the RDF the resources 'Albert Einstein' and 'physics' would be identified by an IRI as would the predicate 'field'.

An RDF base[8] is a set of triples (*e.g.* see at the bottom of the Figure 1). To retrieve data in an RDF base, SPARQL [6] – *Sparql Protocol and RDF Query Language* – is the W3C query language, which is a SQL-like language. For instance, the SPARQL query asking the death place of Albert Einstein in the DBPedia[9] base is as follows:

```
PREFIX dbp-rsc: <http://dbpedia.org/resource/>
PREFIX dbp-owl: <http://dbpedia.org/ontology/>
SELECT ?place
WHERE
{ dbp-rsc:Albert_Einstein dbp-owl:deathPlace ?place }
```

In this query, `?place` is a variable (*i.e.* the searched element), and dbp-rsc:Albert_Einstein and dbp-owl:deathPlace are respectively the IRIs `http://dbpedia.org/resource/Albert_Einstein` and `http://dbpedia.org/ontology/deathPlace` both unavoidably needed to identify the relevant resources of the query.

Remark that an IRI (which is a global identifier) is not always human-intelligible. For instance, Albert Einstein is identified by `https://www.wikidata.org/wiki/Q937` in Wikidata[10] base.

3. SimplePARQL SYNTAX

Like SPARQL, a SimplePARQL query is encoded as a conjunction of triples. The SimplePARQL approach is as an intermediate layer between a user and the SPARQL query language. SimplePARQL is a generalization of SPARQL, where in addition to SPARQL elements the user is allowed to write some *keywords*. Keywords can replace resources' identifiers (IRIs) into an RDF base, that are generally unknown by users but needed to write a SPARQL query. We called such a keyword a *imprecise field*. Of course, imprecise fields can also replace literals.

An imprecise field is a word or a set of words. It can be in quotation marks, *e.g.* `"John Smith"`, which means that all resources containing the exact expression composed by the words of the imprecise field into the queried base have to be returned. An imprecise field in parenthesis, *e.g.* `(Smith John)`, means that all resources containing all words from the imprecise field– in any order, successively or not – have to be returned.

So, in SimplePARQL query, a triple is composed by:

- a variable, preceded by a question mark (`?`),
- an IRI, either enclosed in angle brackets (`< >`), as a short IRI (using `PREFIX`), or the element `a`[11].
- a blank node (`[]` or _:b0), as anonymous resource.
- imprecise field that is either:
  1. single word;
  2. several words in parenthesis;
  3. several words in quotation marks.
     In this case, if the field is in object position, it is equivalent than literal in SPARQL. But it can also be used in other positions (subject or predicate positions). In all cases, it is considered as an imprecise field.

For example, considering the following SimplePARQL query asking[12] for the birth place of John Smith, which has to be in the department[13] 92.

```
SELECT ?place
WHERE {
  "John Smith" (birth place) ?place
  ?place department 92
}
```

In this query, `?place` is a SPARQL variable, while `"John Smith"`, `(birth place)`, `department` and `92` are some imprecise fields.

In a SPARQL query, a redundant variable name (used in several triples) should be handled as a same and a unique resource to be sought. Similarly, in SimplePARQL approach, when a same imprecise field is used, this imprecise field is considered as a same resource.

4. HOW A SimplePARQL IS COMPUTED?

In RDF bases, resources are described by their relations between them and by their links with literals. Indeed, the semantics is contained in relationships. Nevertheless, it is clear that a user makes a shortcut in his/her mind about

[6] http://jena.apache.org
[7] http://virtuoso.openlinksw.com
[8] *a.k.a.* a triplestore, or an RDF graph.
[9] http://dbpedia.org/sparql
[10] http://www.wikidata.org

[11] We recall that the special SPARQL element `'a'` corresponds to rdf:type that also is `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.
[12] Currently the RDF bases contain few texts written in other languages than English. However, if French literals (for instance) are present in a base, the user can write French words as imprecise fields into a SimplePARQL query.
[13] In France, the territory is administratively divided into regions, which are divided into departments (*e.g.* "Hauts-de-Seine" is numbered 92), which finally contain cities.

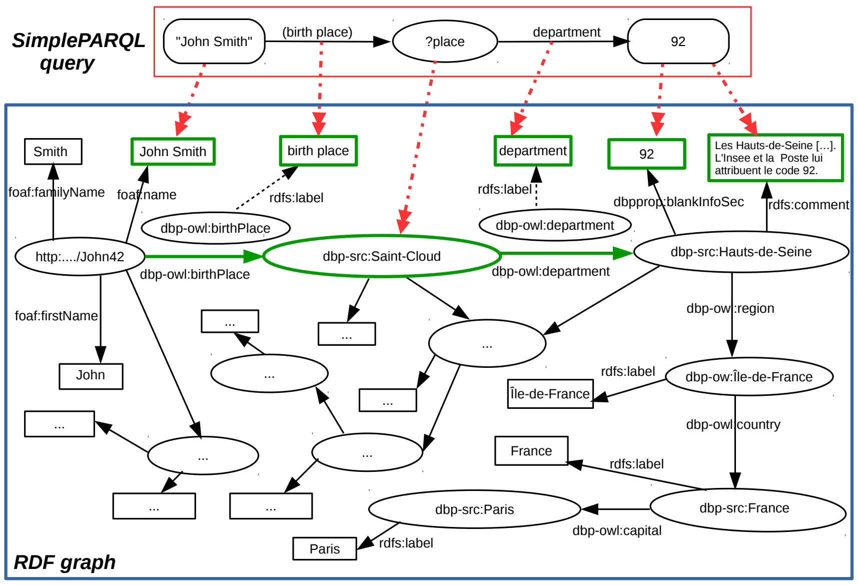

Figure 1: Matching between a SimplePARQL query (at the top) and a part of an RDF triplestore (at the bottom), both in their graph representations.

```
-------------------------------------------------------------------------------------------------------------------------------------------
| place                 | Such as "John Smith" is   | Such as (birth place) is | Such as department is |  Such as 92 is              |
===========================================================================================================================================
|dbp-rsc:Saint-Cloud    | sample:John42             | dbp-owl:birthPlace       | dbp-owl:department    | dbp-src:Hauts-de-Seine      |
|                       | (because has              | (because has             | (because has          | (because has                |
|                       |  foaf:name "John Smith")  |  label "birth place")    |  label "department")  |  dbpprop:blankInfoSec 92)   |
-------------------------------------------------------------------------------------------------------------------------------------------
|dbp-rsc:Saint-Cloud    | sample:John42             | dbp-owl:birthPlace       | dbp-owl:department    | dbp-src:Hauts-de-Seine      |
|                       | (because has              | (because has             | (because has          | (because has                |
|                       |  foaf:name "John Smith")  |  label "birth place")    |  label "department")  |  rdfs:comment  [...] 92.)   |
-------------------------------------------------------------------------------------------------------------------------------------------
```

Figure 2: Results of the SimplePARQL query on the RDF base (RDF graph), both showed in Figure 1.

data contained in an RDF base by reducing the length of the paths between the resources: some information do not seem necessary to be clearly explained. The goal of SimplePARQL approach is to reduce the gap between the interpretation capabilities of a machine and how a user grasps the data, some of which are obvious and implied for the human.

The Figure 1 illustrates our approach: the SimplePARQL query in section 3 is presented as a graph, and the results into the base is given in the Figure 2.

Concretely, to query the Web of Data endpoints with the SimplePARQL query, we extend the SimplePARQL query into N SPARQL queries to match imprecise fields with corresponding resources in RDF graph.

4.1 Different Rewriting Cases

From an intuitive pseudo-SPARQL query, called SimplePARQL query, our approach rewrites it to a set of "real" SPARQL queries that will offer the possibility – when queries will be executed – to make a matching between imprecise fields and RDF resources in a base. Therefore, according to the position of an imprecise field into a triple, some SPARQL queries are generated *w.r.t.* defined rewriting rules. So, the number of rewritings depends on positions of the imprecise fields.

Case 1: the inaccurate-field is in position of *subject*. In this case SimplePARQL query is rewritten into three SPARQL queries.

(a) The first SPARQL query allows to seek all IRIs containing the value of the imprecise field.
(b) The seconds allows to seek all resources that have a rdfs:label property with a value containing the imprecise field.
(c) The third allows to seek all resources having any property (not only a label) with a value containing the imprecise field.

Case 2: the imprecise field is in the position of *predicate*. In this case SimplePARQL query is rewritten into two SPARQL queries.

(a) The first SPARQL query allows to seek all IRI of property (from the ontology of a base) that contains the value of the imprecise field.
(b) The seconds allows to seek all resources, typed as a property *w.r.t.* the ontology of a base, that have a rdfs:label relation with a value containing

the imprecise field (all ontological resources, like properties, created by an expert should have a label).

Case 3: the imprecise field is in the position of *object*. In this case SimplePARQL query is rewritten into three SPARQL queries.

(a) The first SPARQL query allows to seek all literals **or** IRIs containing the value of the imprecise field.
(b) The seconds allows to seek all resources that have a rdfs:label property with a value containing the imprecise field.
(c) The third allows to seek all resources having any property (not only a label) with a value containing the imprecise field.

Even if an IRI is "just" an identifier, the rewriting rules corresponding to the cases 1(a), 2(a), and 3(a) are relevant, as well as words in a URL are significant on the contents of the associated website (*e.g.* `http://dbpedia.org/resource/Albert_Einstein` in DBPedia is more relevant than an id).

There are several various of SPARQL servers such as Apache-Jena (Fuseki), D2R and Virtuoso. Unfortunately, most servers are very slow to seek a sub-string (*i.e.* interpretation of a `CONTAINS` or other `REGEX` functions into a `FILTER`). The SimplePARQL approach takes into account the Web of Data bases server performances. So, when it is possible in a triplestore we use available text search primitives. For instance, on a Virtuoso server, we use the optimized Virtuoso/PL function `'bif:contains'` instead of any SPARQL sub-string search function. Then, the fast execution time of a such primitive makes our SimplePARQL query approach totally workable.

4.2 SimplePARQL Query Rewriting Example

Let us see an example of how a SimplePARQL query with one imprecise field in the subject position is rewritten to the three corresponding SPARQL queries.

The original SimplePARQL query:

```
SELECT ?concept
WHERE { Smith  a  ?concept }
```

The three SPARQL queries corresponding:

- First rewriting:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?concept
WHERE { ?imprField_1   rdf:type  ?concept.
FILTER CONTAINS(UPPER(STR(?imprField_1, "SMITH") }
```

- Second rewriting:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?concept
WHERE { ?imprField_1  rdf:type  ?concept .
        ?imprField_1  rdfs:label ?tmp_var1.
FILTER CONTAINS(UPPER(STR(?tmp_var1), "SMITH") }
```

- Third rewriting:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?concept
WHERE { ?imprField_1  rdf:type  ?concept.
        ?imprField_1  ?tmp_var1 ?tmp_var2.
FILTER CONTAINS(UPPER(STR(?tmp_var2), "SMITH") }
```

If a query is composed of more than one imprecise field and/or more than one triple, all possible combinations of the rewriting cases are done. For example, if a triple is composed of two imprecise fields, the first in the position of subject and the second is in the position of predicate, then the number of rewritings is 6: three for rewriting subject times two for rewriting predicate.

4.3 "Real" Variable *vs* Temporary Variable

When we rewrite a SimplePARQL query to N SPARQL queries, we add several temporary variables into these last queries, which represented the interpretation of imprecise fields. For example, in the first rewriting in section 4.2, a temporary variable `?imprField_1` is added to the real variable `?concept`. However, when we display results to users, we make a specific process to present only the "real" variables' matchings, but we also display as a proof the imprecise fields' matchings (see Figure 2). Note that an imprecise field can be represented by more than one temporary variable.

5. CONCLUSION

Given the difficulty of writing SPARQL queries to access data in RDF bases due to the nearly inevitable handling of resource IRIs, we propose for users to write more expressive and intuitive pseudo-SPARQL queries by using *keywords* instead of IRIs. On our platform universal-endpoint.com, SimplePARQL queries can be written and executed – at the same time – on different bases through their SPARQL endpoints of the Web of Data. SPARQL queries are also accepted on the platform.

Currently, most SPARQL elements [1] can be used on the platform (*i.e.* `PREFIX`, `SELECT`, `WHERE`, `FILTER`, `OPTIONAL` and `LIMIT`). Other elements are not yet available, like `UNION` or `ORDER`, due to the poor speed up executing time [7] by SPARQL engines today.

In future works, it could be interesting to rank the SimplePARQL results according to different cases of rewritings, and also to merge similar results (*e.g.* same label but from different native languages, or due to owl:sameAs relations).

6. ACKNOWLEDGMENTS

This work has been funded by ESILV (École Supérieure d'Ingénieurs Léonard de Vinci) in the frame of the French cooperative project TIMCO, Pôle de Compétitivité Systematic (FUI 13).

7. REFERENCES

[1] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world sparql queries. In *Proc. of the USEWOD Workshop in the 20th WWW Conference*, 2011.

[2] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. Easy access to the freebase dataset. In *Proc. of the 23rd WWW Conference*, pages 95–98, 2014.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 279(5):34–43, 2001.

[4] S. Ferré. Sparklis: a sparql endpoint explorer for expressive question answering. In *the ISWC Posters & Demonstrations Track*, 2014.

[5] S. Lohmann, P. Heim, T. Stegemann, and J. Ziegler. The relfinder user interface: Interactive exploration of relationships between objects of interest. In *Proc. of the 15th IUI Conference*, pages 421–422. ACM, 2010.

[6] E. Prudhommeaux and A. Seaborne. SPARQL Query Language for RDF, `www.w3.org/TR/rdf-sparql-query/`, 2008.

[7] M. Schmidt, M. Meier, and G. Lausen. Foundations of sparql query optimization. In *Proc of the 13th ICDT*, pages 4–33. ACM, 2010.