Semantically-driven recursive navigation and retrieval of data sources in the Web of Data

Valeria Fionda[1], Claudio Gutierrez[2], Giuseppe Pirró[1]

[1] KRDB, Free University of Bolzano-Bozen, Bolzano, Italy
[2] DCC, Universidad de Chile, Santiago, Chile

Abstract. This paper introduces a semantically-driven recursive RDF link-based navigation system for the Web of Data. It combines the power of regular expressions with triggers based on SPARQL queries, to perform controlled exploration and retrieval of semantic data sources.

1 Introduction

With the advent of the Semantic Web, a whole new family of *semantic* data sources appeared, whose interlinking gives birth to the Web of Data. This new Web brings many new data management challenges. Among them, browsing, exploring, discovering, searching, retrieval and querying of semantic data sources.

Regarding exploration and retrieval of data sources, for the Web of Data there is yet no specific tool that permits to recursively specify data sources one would like to explore and/or retrieve using their intrinsic RDF semantics. SPARQL allows static specification of sources via the `FROM` operator, and a certain flexibility in selecting from a given set of (fixed) sources via the `GRAPH` operator. The upcoming SPARQL 1.1 specification extends this idea by allowing to specify *endpoints* sources via the `SERVICE` operator. This solution as well as some other enrichments of SPARQL, although allowing link traversal (e.g., [1]), only provide limited solutions when it comes to explore data sources. Another family of tools focuses on crawling and retrieving mainly syntactically data from the Web of Data (e.g., Raptor [4], LDSPider [3]). We discuss them in the related work section.

There are simple and natural tasks that Semantic Web and Linked Data developers and users need to perform at everyday basis, for example: retrieve me from the FOAF network of Daniel those friends that have more than 2 emails; find all information about people interested in Jazz up to 3 `foaf:knows` links away from me; find out all information about my co-authors but only from data sources with certain properties (e.g., trust, file size).

This paper introduces `swget`, a semantically-driven recursive RDF link-based navigation system for the Web of Data to help developers and users in tasks as those above mentioned. It combines the power of regular expressions with triggers based on SPARQL queries, to perform controlled exploration and retrieval of semantic data sources. `swget` is endowed with a GUI for non-technical users, which enables to easily execute `swget` commands and perform controlled navigation of the results. Further details and examples along with a downloadable version of `swget` are available at `http://swget.wordpress.com`.

2 Example and architecture

One of the novel features of `swget` is that it supports regular-expression-based expansion of RDF predicates. This means that one can, for example, discover people 3 `foaf:knows` links away from a given entity or the interests of the friends of my friends. However, if one would like to: *"find information about cities with less than 15000 persons in which musicians, currently living in Italy, were born"*, recursive expansion alone will not suffice, because eventually a lot of information not actually needed would be retrieved.

Indeed, executing a query like this requires not only navigation capabilities but also some kind of control in the way recursive navigation and data retrieval is performed. For instance, by using an RDF crawler (e.g., LDSpider), starting from `{dbpedia:Italy}` one would be able to obtain all the entities (i.e., Persons, Bands, Painters) for which the property `{dbpedia-owl:hometown}` holds. However, we are interested only in *Persons* that are *Musicians* and not *Bands* and therefore further expanding of such kind of links is not necessary.
The following `swget` command declaratively solves this information need based on the SPARQL queries `Q1` and `Q2` .

```
swget http://dbpedia.org/resource/Italy  -p {dbpedia-owl:hometown}
          [Q1]{dbpedia-owl:birthPlace}[Q2]{owl:sameAs}*
```

```
[Q1]= {ASK ?person rdf:type dbpedia:Person . ?person rdf:type dbpedia:MusicalArtist.}
```

```
[Q2]= {ASK ?town dbpedia-owl:populationTotal ?pop. FILTER (?pop <15000).}
```

As it can be noted, `swget` enables to intertwine in a regular expression triggers based on SPARQL queries. This means that at each step of the evaluation of the regular expression, the SPARQL query will drive the subsequent expansion of predicates and then the navigation process. In more detail, after dereferencing the URI for `{dbpedia:Italy}`, `swget` will only continue to dereference URIs of resources describing *Persons* (i.e., for which the predicates `dbpedia:Person` and `dbpedia:MusicalArtist` in Q1 hold). By continuing the navigation process, after retrieving all description of the cities (by following the `dbpedia-owl:birthPlace` predicate) in which the Italian musicians were born, only links for cities with less than 15000 habitants will be further expanded (by the `FILTER` clause in Q2). The last part of the command (i.e., `{owl:sameAs}`*) instructs `swget` to expand a chain of *identity links* of whatever length.

Architecture. The architecture of `swget`, reported in the left part of Fig. 1, includes seven main components. The *command interpreter* receives the input, i.e., a `URI` and a set of options. The input is then passed to the *controller* module, which checks if a network request is admissible and possibly passes it (on the form of a URI) to the *network manager*. A request is admissible if it complies with optional constraints (e.g., total time-out and/or the time-out for each data source visited, data size and/or the presence/absence of specific RDF properties). The *network manager* performs HTTP GET requests of the form `HTTP GET u`, where

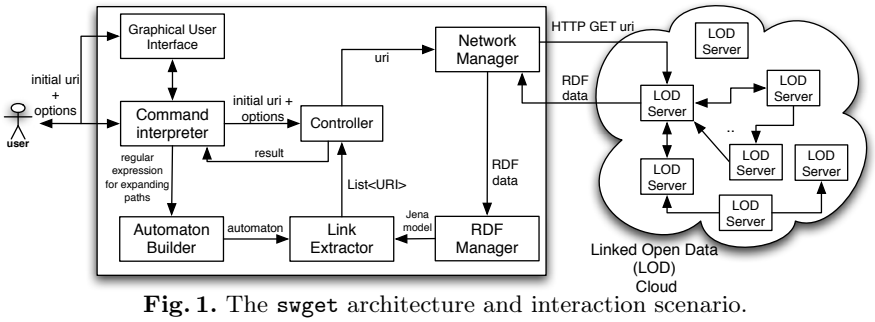

Fig. 1. The `swget` architecture and interaction scenario.

`u` is a URI, and obtains a stream of RDF data. This stream is then processed for obtaining a Jena RDF model, which will be sub-sequentially passed to the *link extractor* module with the aim to discovery links, on the form of URIs, toward other resources. Note that the *link extractor* can possibly take as input an automaton constructed by the *automaton builder* module starting from a regular expression given by the user. In this case, before passing the model to the *link extractor*, the SPARQL query at the current step of the regular expression is executed. If any result is found, then the process of expansion of links will continue according to the results of the SPARQL query. If no results are found, then this model will not be added to the set of model in the result and the process continues with the next step of the regular expression. The *link extractor* selects a subset of links among all the possible links by considering the current state of the automaton and returns this set to the *controller module*, which starts over the cycle. This module will terminate the execution and return the result either when some restriction imposed by the user is satisfied or when there are no more URIs to be dereferenced.

Graphical User Interface. `swget` is also available with an intuitive GUI, for non-technical users (see Fig. 2). Parameters for controlling the navigation can be easily settled. The result of the execution of a given command is represented as a graph interlinking the descriptions of the various resources whose URIs have been dereferenced. It is also possible to control the navigation of the results by filtering predicates, searching for specific predicates or resources and expanding resources by providing a radius value from a target URI.

3 Related and Future Work

Table 1 compares `swget` with similar systems for retrieving data from the Web. `swget` differs from `wget` because even if `wget` can retrieve RDF files, it does not process semantically the contents (e.g., it cannot perform controlled navigation according to the semantics of RDF). Raptor works with RDF but it does not offer navigation features. LDSpider is a crawler for the Web of Data offering limited semantic control over the navigation. `swget` enables semantically-driven controlled navigation thanks to regular expressions and SPARQL queries.

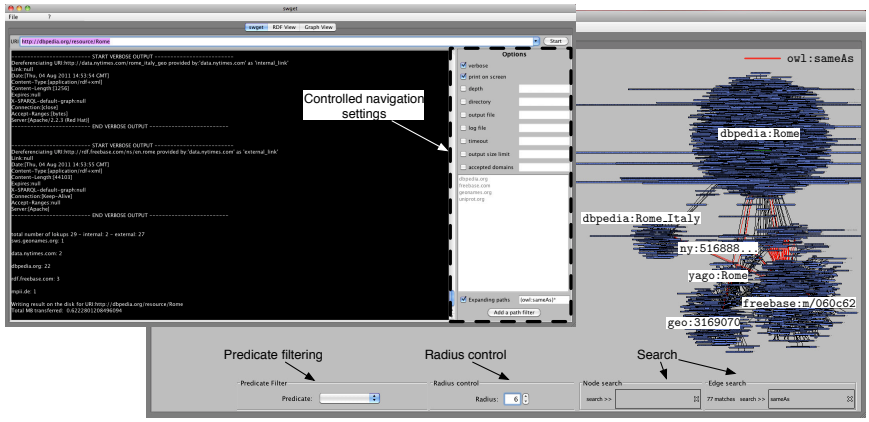

Fig. 2. The `swget` GUI.

Table 1. Comparison between `swget` and systems for retrieving data on the Web

	Features					
System	*Recursive lookup*	*Controlled navigation*	*RDF*	*REGEXPR for expanding paths*	*GUI*	*Semantic Check*
`swget`	X	domains,size,time,depth	X	X	X	X
`wget` [5]	X	size	-	-	-	-
Raptor [4]	-	-	X	-	-	-
LDSpider [3]	X	domains,depth	X	-	-	-

`swget` shares some features with link-traversal-based query approaches to answer SPARQL queries over the Web of Data [1, 2] such as SQUIN [6], which offers an interface for querying the Web of Linked Data. But in these approaches, link-traversal is introduced as an extension of SPARQL semantics, thus limiting explicit control over the navigation. Conversely, `swget` offers a *controlled navigation* mechanism for the Web of Data, which enables to consider only relevant data sources. Regarding SPARQL 1.1, it exploits property paths regular expressions over one dataset. The main difference is that `swget` uses them to control navigation by intertwining regular expressions with calls to other sites.

There is room for improving several aspects of the current `swget`. Particularly, optimization and including additional expressiveness on the constraints are currently being investigated. We also plan to export `swget`'s features under other forms (e.g., Web services).

References

1. Hartig, O., Bizer C., Freytag, J.C.: Executing SPARQL Queries over the Web of Linked Data. In *Proc. of ISWC*, pp. 293-309, (2009).
2. Hartig, O.: Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *Proc. of ESWC*, pp. 154-169, (2011).
3. LDSpider: `http://code.google.com/p/ldspider`
4. RDF Raptor website: `http://librdf.org/raptor`
5. `wget` website: `http://www.gnu.org/s/wget`
6. SQUIN website: `http://www.squin.org`