SPARQture: A More Welcoming Entry to SPARQL Endpoints

Fadi Maali

Insight Centre for Data Analytics, National University of Ireland Galway
fadi.maali@insight-centre.org

Abstract. The semi-structured nature of Semantic Web data and its use of multiple vocabularies make it challenging to describe the structure of some RDF dataset. When trying to understand the content behind some SPARQL endpoint, users usually need to write a number of SPARQL queries to "mine" the types contained in the dataset and their relations. We argue that this is a tedious, error-prone and time consuming task to be carried out manually. SPARQture, the tool described in this paper, automates this process and renders the structure of a particular RDF data available behind some SPARQL endpoint using a number of visually rich components. Moreover, SPARQture supports user interaction to further explore the data. We describe SPARQture and discuss its fit to the IESD Challenge. The reader is encouraged to view the demo at: http://sparqture.appspot.com

1 Introduction

The Semantic Web community succeeded in making large amounts of structured data available on the Web in the last decade. This data, represented using the RDF data model, spans various domains from social networks, to bioinformatics to geography. Many of these datasets are made available via SPARQL endpoints. SPARQL [4] provides a standardised expressive query language that supports slicing and dicing RDF data. More than 500 SPARQL endpoints are currently listed on datahub.io.

SPARQL endpoints are commonly accessible over the Web. They can be queried using any tool that supports the HTTP protocol or via an HTML form, typically with a large input text. The user is then left with the daunitng task of discovering the content of the endpoint in order to be able to issue meaningful queries. This task is particularly challenging for RDF as it is semi-structured data that tends to be heterogeneous and uses multiple vocabularies.

A common way to scrutinise the content of some SPARQL endpoint is via issuing a series of SPARQL queries[1]. These usually go from listing classes in the

[1] For example see http://dallemang.typepad.com/my_weblog/2008/08/rdf-as-self-describing-data.html and http://answers.semanticweb.com/questions/25696/extract-ontology-schema-for-a-given-sparql-endpoint-data-set

endpoint, to properties of each class, to sample instances of some classes, etc. Furthermore, to get an overview of the structure and relationships in the data, a more complex sequence of SPARQL queries are needed. We argue that this is a tedious, error-prone and time consuming task to be carried out manually. SPARQture, the tool described in this paper, automates this process.

SPARQture extracts and explicitly presents the schema used in a particular RDF data available behind some SPARQL endpoint. The tool issues a number of queries and then renders the information it collects using a number of visually rich and well-known information visualisation components. Furthermore, SPARQture supports user interaction to further explore the data. In the remainder of this paper, we provide more details about SPARQture (section 2) and discuss its fit to the IESD Challenge by addressing the three key themes of the workshop: human factors, computational models and application domains (section 3).

2 Overview of the System

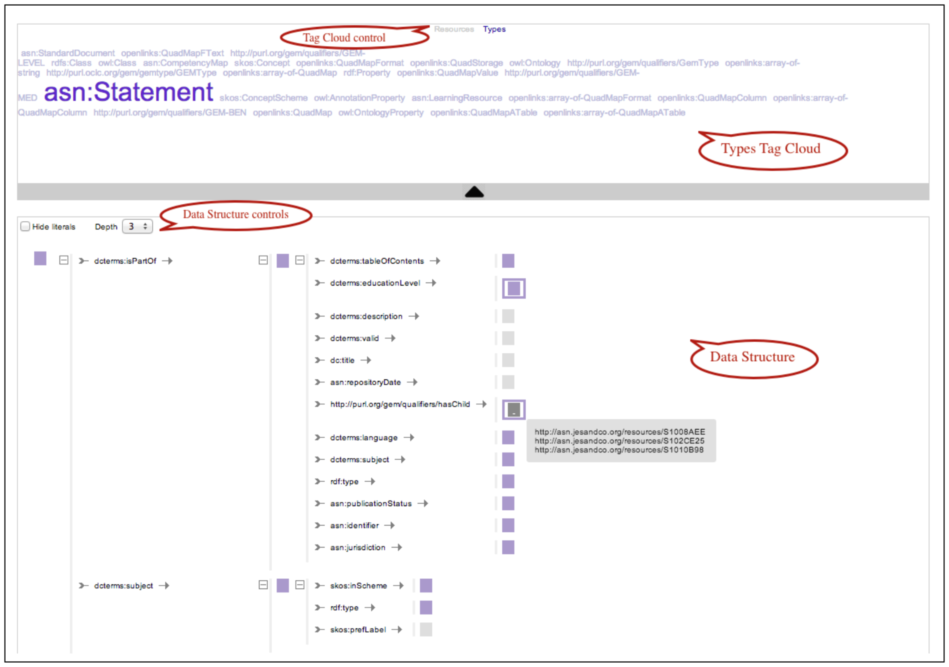
Tag Cloud control
Resources
Types
asn:StandardDocument openlinks:QuadMapFText http://purl.org/gem/qualifiers/GEM-LEVEL rdfs:Class owl:Class asn:CompetencyMap skos:Concept openlinks:QuadMapFormat openlinks:QuadStorage owl:Ontology http://purl.org/gem/qualifiers/GemType openlinks:array-of-string http://purl.oclc.org/gem/gemtype/GEMType openlinks:array-of-QuadMap rdf:Property openlinks:QuadMapValue http://purl.org/gem/qualifiers/GEM-MED asn:Statement skos:ConceptScheme owl:AnnotationProperty asn:LearningResource openlinks:array-of-QuadMapFormat openlinks:QuadMapColumn openlinks:array-of-QuadMapColumn http://purl.org/gem/qualifiers/GEM-BEN openlinks:QuadMap owl:OntologyProperty openlinks:QuadMapATable openlinks:array-of-QuadMapATable
Types Tag Cloud
Data Structure controls
Hide literals
Depth 3
dcterms:isPartOf
dcterms:tableOfContents
dcterms:educationLevel
dcterms:description
dcterms:valid
dc:title
asn:repositoryDate
http://purl.org/gem/qualifiers/hasChild
dcterms:language
dcterms:subject
rdf:type
asn:publicationStatus
asn:identifier
asn:jurisdiction
dcterms:subject
skos:inScheme
rdf:type
skos:prefLabel
http://asn.jesandco.org/resources/S1008AEE
http://asn.jesandco.org/resources/S102CE25
http://asn.jesandco.org/resources/S1010B98
Data Structure

Fig. 1. SPARQture main interface

Figure 1 shows the main interface of our tool, SPARQture. The main interface comprises two main panels: the tag cloud panel (top part) and the structure panel (bottom part). The tag cloud panel provides an overview of the data by displaying the types of entities in the endpoint. On the other hand, the structure panel allows getting detailed view of the structure of entities of a particular type. The structure is presented as a tree and can be further navigated by clicking on the nodes.

The user starts by entering the URL of a sparql endpoint or a VoID[2] file location. SPARQture presents a tag cloud of types in the endpoint where the size of a tag reflects the number of instances of that type. By clicking on some type, a set of sample resources are shown in the tag cloud panel and the structure of some of these resources is plotted in the structure panel. It is worth mentioning here that the root of the tree does not have to be a single resource but can represent multiple resources. In the process of navigation, a node can represent multiple values. By clicking on such node, SPARQture combines the structure of the relations of these resources collectively, hence, supporting what can be referred to as set-based browsing [5].

Finally, we summarise what we believe are the novel aspects of SPARQture:

- Extracts and explicitly presents the schema used in the underlying RDF data.
- Provides multiple visually-rich views over the data that allows exploring it at various levels of abstraction.
- Can run on top of any endpoint that supports **SPARQL 1.1**.

3 Addressing Key Themes of the Workshop

3.1 Human Factors

SPARQture combines a number of familiar components to support presenting and exploring information. Specifically, tag clouds and tree visualisations are used. Tag clouds utilise font size and colour as visual indicators of the importance of an item as recommended in [1]. On the other hand, the tree component provides a comprehensible way of representing data structure. Colour of a node in the tree encodes its type (resources vs. literals). Furthermore, for nodes that represent multiple items, a thick border is added. To emphasise the structure of the data, we do not show the values of the tree nodes (resources and literals), but make them available via a tooltip upon hovering over nodes. We believe that these visual clues allow getting an idea about the data structure in a short time. In both the tag cloud and the tree view, URIs are presented in the shortened form of `prefix:local-name` where prefixes come from `prefix.cc` service.

The choice of a tree instead of a graph to present data structure is based on our belief that trees are easier to comprehend. By allowing duplicate nodes in trees, graphs that contain cycles can still be represented as trees. A similar decision was made in the Tabulator browser [2].

[2] `http://www.w3.org/TR/void/`

3.2 Computational Model

The core of SPARQture is an engine that translates user's interactions into a series of SPARQL queries. The results of the SPARQL queries are serialised as JSON objects that are rendered using JavaScript on the client side. We use Jquery TagCloud library[3] to render tag clouds and reuse Javascript code from Open Refine [4] project to render trees.

In the following we highlight a number of aspects of the computational model:

SPARQL queries generation: We try to group SPARQL queries together to reduce the number of queries issued at the SPARQL endpoint and enhances the interface responsiveness at the same time. For example, the resource representing Dublin in DBpedia[5] has 261 related resource (i.e., a triple with a URI object). Nevertheless, SPARQture issues only 8 queries to compute the tree structure with depth 3 rooted at Dublin. This is achieved via sending queries based on unique properties and grouping multiple properties in one query.

Content discovery: The entry point for the tool can be a SPARQL endpoint URL or a VoID file. SPARQture tries to locate the SPARQL endpoint location from the VoID file by searching for `void:sparqlEndpoint` property. We are currently working on extending this discovery feature to any resource URI by consulting the Endpoint Lookup Service[6].

A few heuristics: We embedded a number of heuristics in the tool to select resources shown as sample resources or used as a root in the tree.

3.3 Application Domain

SPARQture is independent from the application domain. It can support any domain as long as an endpoint that supports SPARQL 1.1 is provided.

4 Discussion & Conclusion

Extracting the schema of some semi-structured data has also been studied in the context of XML data [6]. This same task is more challenging in RDF data because it is a graph data, contrary to the tree structure of XML data. Tree structures have a designated root that provides an entry point to access all the items in the tree. A tree root has no guaranteed equivalent in graphs. Our initial work tried to use `void:rootResource` as a root for the RDF graph data. However, this proved to be infeasible as this property is very rarely used and, when it exists, computing the structure starting from it is prohibitively expensive. There exists some work in the Semantic Web community to summarise RDF graphs based

[3] `https://github.com/addywaddy/jquery.tagcloud.js/`
[4] `http://openrefine.org/`
[5] `http://dbpedia.org/resource/Dublin`
[6] `http://void.rkbexplorer.com/endpoint-search/`

on clustering resources by types and then extracting relationships between the computed clusters [3]. SPARQture can be used on top of RDF graph summary as a presentation layer that utilises the precomputed structure. We hope to implement this functionality in the future.

To conclude, we described SPARQture, a tool that aims at enhancing accessibility of SPARQL endpoints by providing a more welcoming interactive entry point. SPARQture uses tag clouds and trees to provide an overview of the data and supports interactive exploration of the data behind some SPARQL endpoint.

Acknowledgements. Fadi Maali is funded by the Irish Research Council, Embark Postgraduate Scholarship Scheme.

References

1. Scott Bateman, Carl Gutwin, and Miguel Nacenta. Seeing Things in the Clouds: The Effect of Visual Features on Tag Cloud Selections. In *Proceedings of the Nineteenth ACM Conference on Hypertext and Hypermedia*, HT '08, 2008.
2. Tim Berners-lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In *In Procedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06)*, 2006.
3. Stephane Campinas, Thomas E. Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. Introducing RDF Graph Summary With Application to Assisted SPARQL Formulation. In *Proceedings of the 23rd International Workshop on Database and Expert Systems Applications*, 2012.
4. Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013. http://www.w3.org/TR/sparql11-query/.
5. David F Huynh and David Karger. Parallax and Companion: Set-based Browsing for the Data Web. Proceedings of WWW'09, 2009.
6. Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, and Sudarshan S. Chawathe. Representative Objects: Concise Representations of Semistructured, Hierarchial Data. In *Proceedings of the Thirteenth International Conference on Data Engineering*, ICDE '97, 1997.