SPARQLing: A Graphical Interface for SPARQL

Shima Dastgheib, Daniel Ian McSkimming, Natarajan Kannan and Krys Kochut

University of Georgia, Athens, GA, 30602; USA
shida@uga.edu, dim@uga.edu, kannan@bmb.uga.edu, kochut@cs.uga.edu

Abstract. SPARQLing presents a graphical SPARQL query builder, which allows users of domain ontologies and RDF datasets formulate queries through visual inspection of classes and properties in the ontology or RDF schema. Our approach takes into account the users' preferences in viewing the elements of the interface.

1 Introduction

Domain-specific ontologies and RDF datasets serve as an integral component of various semantics-driven applications and an important source of knowledge in those domains. Many users of domain ontologies and RDF datasets are not familiar with RDF and SPARQL, but are fluent or even expert in their own domain of knowledge. Hence, bridging the gap between Semantic Web developers and users could open an important avenue for knowledge discovery. This paper presents SPARQLing, a graphical SPARQL query builder, which allows domain users with no prior knowledge of formulating SPARQL queries over domain ontologies and RDF datasets, through visual inspection of classes, relationships, and in some cases instances available in the dataset. Our approach is based on a customizable visualization of the ontology/RDF schema. The schema provides information on how the contents of the domain ontology or RDF dataset is organized and hence, how it can be utilized to answer questions in that domain.

We have used SPARQLing to create a graphical query interface for ProKinO[1], the Protein Kinase Ontology, which provides a conceptual representation of protein kinases. ProKinO is a valuable resource for mining and annotating the cancer kinome. At present, complicated SPARQL queries must be formulated to retrieve a variety of kinase-related data necessary for testing and exploring various hypotheses.

2 Related Work

We have studied the existing graphical query builders first, based on what they visualize (i.e., schema and/or query), and second, based on the metaphors they have applied for visualization purposes. Many of the query builders visualize the query pattern using visual query languages (VQS) and are either form-based (e.g. SPARQLViz) or diagram-based (e.g. NITELIGHT [1], iSPARQL [2], Cuebee [3], SEWASIE [4], RDF-GL [5], SPARQLGraph [6]). Few systems such as Konduit VQB [7] visualize the schema along with the query, but are mainly a collection of drop-down lists.

[1] http://vulcan.cs.uga.edu/prokino/about/browser

3 SPARQLing: Approach and Methodology

As shown in Fig. 1, first, the ontology/RDF dataset is loaded into the system. Then, the Discovery Mode Module executes a set of pre-defined SPARQL queries to obtain the necessary information about the ontology/RDF schema. The results of the discovery queries are used in the Configuration Module and, finally, the query interface is created and displayed based on the configuration file.

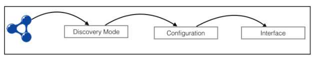

Fig. 1. SPARQLing Overview

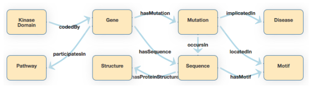

Fig. 2. A fragment of the schema-based query interface for ProKinO

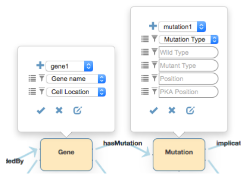

Fig. 3. Subclasses and datatype properties

The SPARQLing interface starts by displaying the overall layout of the ontology/RDF schema (or, more likely, its fragment of interest, configured during setup) as a graph, where each class (shown as a rectangle) is linked to its neighboring classes by arrows representing object properties (e.g. Fig. 2). To build a query, the user selects the desired classes and the links (relationships) between them. If the query requires additional constraints by class datatype property values, they are introduced within class popovers, as described below (e.g. Fig. 3). The main metaphors include:

- **Class Hierarchy**: To keep the schema graph more readable, we placed the subclasses of each class (if any) in the form of a cascading drop-down list inside the class popover. Each subclass may have other subclasses as well, shown as a drop-down list, which are populated as the parent class is selected.
- **Datatype Property**: Datatype properties can be viewed differently, depending on the values that can be assigned to them. For example, the value for the property "hasKinaseDomain" of the Structure class only accepts either "true" or "false" values, while the "hasPrimaryName" property of the Gene class accepts a string from the list of all kinases. Therefore, we designed three different types of metaphors for data property values, as described below.
  1) *Text field*: This is the default metaphor for the datatype properties. If the user wants to set a value of a datatype property, he/she can enter the value directly into the text field.
  2) *Text field with auto completion*: As the user types in a property value, the list of suggested values is shown. The list should be pre-populated and stored as a JSON file to speed-up the query time. Alternatively, it can be dynamically populated from the ontology when the query builder is started up.
  3) *Drop-down list*: Here, the values of a property (e.g., the primary names of diseases) are available as a drop-down list, to make the selection process both

effortless for the user and less error-prone for the query builder. Typically, this is used for properties with a relatively low numbers of discrete values. Again, the list should be retrieved ahead of time and stored as a JSON file.

4) *Check box*: This can be used for properties with only Boolean values (e.g., hasKianaseDomain). If the box is checked, the value of the property is set as "true" and its corresponding triple is added to the query. The box should be left unchecked if user does not intend to restrict that property. More Filter functionality is required to support the third possibility, in which, user intends to exclude the "true" values. By default, the check box is not selected.

- **Filter:** filter is a restriction on solutions over the whole group of graph patterns in which the filter clause appears. We added limited filter functionality to the query builder by allowing the relational operators <, <= >, >=, and != for properties with numerical values, such as Position in the Mutation class.

In the current version of SPARQLing, the user can build SELECT queries with only one group of triple patterns (OPTIONAL and UNION have not been implemented, yet). Also, support for aggregate queries and complex FILTER expressions is under development.

- **Query Graph Pattern:** The main graph pattern of a SPARQL query is composed of a set of triple patterns. With the help of the user-friendly interface described above, the user interacts with the schema metaphors, acquires a better insight into the schema and builds the query with the assistance of additional metaphors, described below:
  - **Multiple instances:** user can add multiple instances of one class to the query pattern by clicking the "+" button inside the popover. Each added instance is accessible at anytime for modification or deletion from the query pattern.
  - **Value assignment:** if needed in the query being formulated, datatype property values are set using the available metaphor, as described above.
  - **Result selection**: query variables are added to the query SELECT clause by choosing the "Select" option from the leftmost button on the side of each datatype property. Result selection is also needed in case the user is interested in viewing the class instances as a whole in the query result, rather than specific property values.
  - **Result count:** choosing the "Count" option from the leftmost button on the side of each datatype property returns the count of the query results.
  - **Class instance selection:** Once the user completed assigning values to datatype properties of each instance of the class**,** or adding them to the result set, he/she selects the "✔" button located inside the class popover. As a result, the corresponding triple patterns are added to the query panel.
  - **Object Property Selection:** the arrows connecting the classes can be clicked to add the corresponding triple relating the classes connected by the object property to the query. The color of the arrow changes, once selected.

As the user interacts with the schema and builds a query using the provided interface elements, SPARQL triples are gradually added to the query panel, placed on the right hand side of the query builder and visible to the user.

4 Demonstration

We have created a graphical interface for querying ProKinO, using the SPARQLing approach. The created interface is configured to include a selected set of classes and properties. In addition, based on the configuration file, the metaphors for different datatype properties have been set and, if required, their values have been obtained from the ontology. The schema graph is drawn using the jsPlumb library with jQuery.

The query interface and links to demonstration videos are available at *http://kronos.cs.uga.edu/prokino/query/build.* We will use a set of example queries to demonstrate our query builder functionalities explained in section 3.

5 Conclusions and Future Direction

SPARQLing enables users to create complex, integrative queries over domain ontologies and RDF datasets without prior knowledge of formulating SPARQL queries or full understanding of the schema. The customization of the query interface is currently done by interacting with a configuration file. We are working on an interactive user interface to facilitate this process.

Our system assists users in capturing the classes, properties and relationships from the queries and mapping it to the classes and relations in the ontology. We plan to work on an NLP approach to further facilitate this transition. We plan to make the source code available in future, promote the query interface generated with SPARQLing to a large research community, and get their feedback to make additional enhancements. We also plan to enable additional types of queries the interface supports, such as nested queries, more complex filter expressions, and aggregate queries. Another future direction is to enable suggestions of paths between classes, which are not directly connected, in order to further facilitate building of integrative queries.

Acknowledgement: Funding for NK from the National Science Foundation (MCB-1149106) is acknowledged.

6 References

1. Russell, A.a.S., Paul, *NITELIGHT: A Graphical Editor for SPARQL Queries*, in *At 7th International Semantic Web Conference (ISWC 2008)*. 2008: Germany. p. 26-30.
2. *iSPARQL*. Available from: http://oat.openlinksw.com/isparql/index.html.
3. *Cuebee*. Available from: http://cuebee.sourceforge.net/.
4. Fillottrani, P.R. *The SEWASIE architecture: a multi-agent system for data integration*. in *XI Congreso Argentino de Ciencias de la Computación*. 2005.
5. Hogenboom, F., et al., *RDF-GL: a SPARQL-Based graphical query language for RDF*, in *Emergent Web Intelligence: Advanced Information Retrieval*. 2010, Springer. p. 87-116.
6. *SPARQLGraph*. Available from: http://sparqlgraph.i-med.ac.at/.
7. Ambrus, O., K. Möller, and S. Handschuh. *Konduit vqb: a visual query builder for sparql on the social semantic desktop*. in *Workshop on Visual Interfaces to the Social and Semantic Web*. 2010.