Sparklis: a SPARQL Endpoint Explorer for Expressive Question Answering

Sébastien Ferré

IRISA, Université de Rennes 1
Campus de Beaulieu, 35042 Rennes cedex, France
Email: ferre@irisa.fr

Abstract. Sparklis is a Semantic Web tool that helps users explore SPARQL endpoints by guiding them in the interactive building of questions and answers, from simple ones to complex ones. It combines the fine-grained guidance of faceted search, most of the expressivity of SPARQL, and the readability of (controlled) natural languages. No endpoint-specific configuration is necessary, and no knowledge of SPARQL and the data schema is required from users. This demonstration paper is a companion to the research paper [2].

1 Motivation

A wealth of semantic data is accessible through SPARQL endpoints. DBpedia alone contains several billions of triples covering all sorts of topics (e.g., people, places, buildings, species, films, books). Although different endpoints may use different vocabularies and ontologies, they all share a common interface to access and retrieve semantic data: the SPARQL query language. In addition to being a widely-adopted W3C standard, the advantages of SPARQL are its *expressivity*, especially since version 1.1, and its *scalability* for large RDF stores thanks to highly optimized SPARQL engines (e.g., Virtuoso, Jena TDB). Its main drawback is that writing SPARQL queries is a tedious and error-prone task, and is largely unaccessible to most potential users of semantic data.

Our motivation in developing Sparklis[1], shared by many other developers of Semantic Web tools and applications, is to unleash access to semantic data by making it easier to define and send SPARQL queries to endpoints. The novelty of Sparklis is to combine in an integrated fashion different search paradigms: Faceted Search (FS), Query Builders (QB), and Natural Language Interfaces (NLI). That integration is the key to reconcile properties for which there is generally a trade-off in existing systems: user guidance, expressivity, readability of queries, scalability, and portability to different endpoints [2].

2 Principles

Sparklis re-uses and generalizes the interaction model of Faceted Search (FS) [8], where users are *guided step-by-step* in the selection of items. At each

[1] Online at http://www.irisa.fr/LIS/ferre/sparklis/osparklis.html

step, the system gives a set of suggestions to refine the current selection, and users only have to pick a suggestion according to their preferences. The suggestions are specific to the selection, and therefore support exploratory search [7] by providing overview and feedback during the search process.

To overcome *expressivity* limitations of FS and existing extensions for the Semantic Web (e.g., gFacet [4], VisiNav [3], SemFacet [1]), we have generalized it to Query-based Faceted Search (QFS), where the selection of items is replaced by a structured query. The latter is built step-by-step through the successive choices of the user. This makes Sparklis a kind of Query Builder (QB), like SemanticCrystal [5]. QBs have the advantage to allow for a high expressivity while assisting users about syntax, e.g. avoiding syntax errors, listing eligible constructs. However, the FS-based guidance of Sparklis is more fine-grained than in QBs. Sparklis avoids vocabulary errors by retrieving the URIs and literals right from the SPARQL endpoint. It needs not be configured for a particular dataset, and dynamically discovers the data schema. In fact, Sparklis only allows the building of queries that *do* return results, preventing users to fall on empty results. That is because system suggestions are computed for the individual results, not for their common class. In fact, Sparklis is as much about building answers as about building questions.

To overcome the lack of *readability* of SPARQL queries for most users, Sparklis queries and suggestions are verbalized in natural language so that SPARQL queries never need to be shown to users. This makes Sparklis a kind of Natural Language Interface (NLI), like PowerAqua [6]. The important difference is that questions are built through successive user choices in Sparklis instead of being freely input in NLIs. Sparklis interaction makes question formulation more constrained, slower, and less spontaneous, but it provides guidance and safeness with intermediate answers and suggestions at each step. Moreover, it avoids the hard problem of NL understanding: i.e., ambiguities, out-of-scope questions. A few NLI systems, like Ginseng [5], are based on a controlled NL and autocompletion to suggest the next words in a question. However, their suggestions are not fine-grained like with FS, and less flexible because they only apply to the end of the question. In Sparklis, questions form complete sentences at any step of the search; and suggestions are not words but meaningful phrases (e.g., `that has a director`), and can be inserted at any position in the current question.

3 User Interface and Interaction

Figure 1 is a Sparklis screenshot taken during an exploration of book writers in DBpedia. From top to bottom, the user interface contains (1) navigation buttons and the endpoint URL, (2) the current question and the current *focus* as a subphrase (highlighted in green), (3) three lists of suggestions for insertion at the focus, and (4) the table of answers. The shown question and answer have been built in 10 steps (8 insertions and 2 focus moves): `a Writer/that has a birthDate/after 1800`/focus on `a Writer/that is the author of something/a Book/a number of/the`

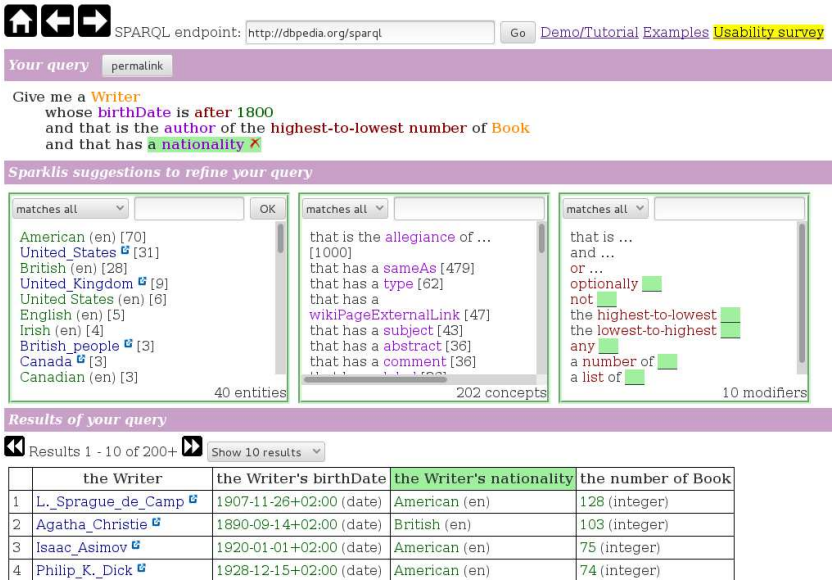


Fig. 1. SPARKLIS screenshot: a list of writers with their birth date (after 1800), nationality, and (decreasing) number of written books. Current focus is on writer's nationality.

`highest-to-lowest`/focus on `a Writer`/`that has a nationality`. Note that different insertion orderings are possible for a same question. Navigation buttons allow to move backward/forward in the construction history. A permalink to the current navigation state (endpoint+question) can be generated at any time. To switch to another SPARQL endpoint, it is enough to input its URL in the entry field. The query focus is moved simply by clicking on different parts of the question, or on different table column headers. Every suggestion in the three lists, as well as every table cell, can be inserted or applied to the current focus by clicking it. The first suggestion list contains entities (individuals and literals). The second list contains concepts (classes and properties). The third list contains logical connectives, sorting modifiers, and aggregation operators. Each suggestion list is equipped with an immediate-feedback filtering mechanism to quickly locate suggestions in long lists. With the first list, filters can be inserted into the query with different filter operators listed in a drop-down menu (e.g., `matches`, `higher or equal than`, `before`). Questions and suggestions use indentation to disambiguate different possible groupings and improve readability, and syntax coloring to distinguish between the different kinds of words.

4 Performances and Limitations

Portability. SPARKLIS conforms to the SPARQL standard, and requires no preprocessing or configuration to explore an endpoint. It entirely relies on the end-

point to discover data and its schema. The main limitation is that URIs are displayed through their local names, which is not always readable.

Expressivity. Sparklis covers many features of SPARQL: basic graph patterns (including cycles), basic filters, `UNION`, `OPTIONAL`, `NOT EXISTS`, `SELECT`, `ORDER BY`, multiple aggregations with `GROUP BY`. Almost all queries of the QALD[2] challenge can be answered. Uncovered features are expressions, named graphs, nested queries, queries returning RDF graphs, and updates.

Scalability. Sparklis is responsive on the largest well-known endpoint: DBpedia. Among the 100 QALD-3 questions, half can be answered in less than 30 seconds (wall-clock time including user interaction and system computations).

5 Demonstration

The demonstration has shown to participants how QALD questions over DBpedia can be answered in a step-by-step process. Those questions cover various retrieval tasks: basic facts (*Give me the homepage of Forbes*), entity lists (*Which rivers flow into a German lake?*), counts (*How many languages are spoken in Colombia?*), optimums (*Which of Tim Burton's films had the highest budget?*). More complex analytical question answering has also been demonstrated (*Give me the total runtime, from highest to lowest, of films per director and per country*). Participants were also given the opportunity to explore any SPARQL endpoint of their choice.

References

1. Arenas, M., Grau, B., Kharlamov, E., Š. Marciuška, Zheleznyakov, D., Jimenez-Ruiz, E.: SemFacet: Semantic faceted search over YAGO. In: World Wide Web Conf. Companion. pp. 123–126. WWW Steering Committee (2014)
2. Ferré, S.: Expressive and scalable query-based faceted search over SPARQL endpoints. In: Mika, P., Tudorache, T. (eds.) Int. Semantic Web Conf. Springer (2014)
3. Harth, A.: VisiNav: A system for visual search and navigation on web data. J. Web Semantics 8(4), 348–354 (2010)
4. Heim, P., Ertl, T., Ziegler, J.: Facet graphs: Complex semantic querying made easy. In: et al., L.A. (ed.) Extended Semantic Web Conference. pp. 288–302. LNCS 6088, Springer (2010)
5. Kaufmann, E., Bernstein, A.: Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. J. Web Semantics 8(4), 377–393 (2010)
6. Lopez, V., Fernández, M., Motta, E., Stieler, N.: PowerAqua: Supporting users in querying and exploring the semantic web. Semantic Web 3(3), 249–265 (2012)
7. Marchionini, G.: Exploratory search: from finding to understanding. Communications of the ACM 49(4), 41–46 (2006)
8. Sacco, G.M., Tzitzikas, Y. (eds.): Dynamic taxonomies and faceted search. The information retrieval series, Springer (2009)

[2] `http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/`