SPARQL-LD: A SPARQL Extension for Fetching and Querying Linked Data

Pavlos Fafalios and Yannis Tzitzikas

Computer Science Department, University of Crete, Greece, and
Institute of Computer Science, FORTH-ICS, Greece
{fafalios,tzitzik}@ics.forth.gr

Abstract. SPARQL is a standard query language for retrieving and manipulating RDF data. However, the majority of SPARQL implementations require the data to be available in advance, i.e., to exist in main memory or in a RDF repository (accessible through a SPARQL endpoint). Nonetheless, Linked Data exists in the Web in various forms; even an HTML Web page can contain RDF data through RDFa, or RDF data may be dynamically created by a Web Service. In this paper, we propose and demonstrate an extension of SPARQL 1.1, called `SPARQL-LD`, that allows to directly and flexibly exploit this wealth of data. `SPARQL-LD` allows to fetch, query and integrate in the same SPARQL query: i) data coming from online RDF or JSON-LD files, ii) data coming from dereferenceable URIs, iii) data embedded in Web pages as RDFa, iv) data that is dynamically created by Web Services, and v) data coming by querying other endpoints. A distinctive characteristic of this extension is that it enables to fetch and query even data in datasets returned by a portion of the query, i.e. discovered at query-execution time.

1 Introduction

Linked Data [6] provides a publishing paradigm in which data can be a first class citizen of the Web. Linked Data exists in various forms: in RDF/XML (e.g. through dereferenceable URIs), as Notation3 (N3) or Turtle files, as JSON-LD files [1], in HTML Web pages as RDFa [2], even a Web Service may create RDF data dynamically (at request time). SPARQL [4] is a standard query language for retrieving and manipulating RDF data. Although most SPARQL implementations require the data to be available in advance (in main memory or in a repository), the specification of SPARQL allows to directly query an RDF dataset accessible on the Web (in a standard format) and identifiable by an IRI through the operators `FROM`/`FROM NAMED` and `GRAPH`. However, this has an important limitation: it requires knowing *in advance* the IRI of the dataset and having declared it in the `FROM NAMED` clause. Thus, an IRI coming from partial results (that get bound after executing an initial query fragment) cannot be used in the `GRAPH` operator as the dataset to run a portion of the query. Furthermore, although RDFa and JSON-LD are W3C standards that are exploited by an ever-increasing number of publishers, we have not managed to find a SPARQL implementation that can directly query such RDF data. In addition, using the

```
1 SELECT DISTINCT ?authorURI (count(distinct ?paper)  AS ?numOfPapers)
2                            (count(distinct ?series) AS ?numOfDiffConfs) WHERE {
3   SERVICE <http://users.ics.forth.gr/~fafalios/> {
4     ?p <http://purl.org/dc/terms/creator> ?authorURI }
5   SERVICE ?authorURI { ?paper <http://purl.org/dc/elements/1.1/creator> ?authorURI }
6   SERVICE <http://dblp.l3s.de/d2r/sparql> {
7     ?p2 <http://purl.org/dc/elements/1.1/creator> ?authorURI .
8     ?p2 <http://swrc.ontoware.org/ontology#series> ?series  }
9 } GROUP BY ?authorURI ORDER BY ?numOfPapers
```

Fig. 1: Example SPARQL query that can be answered by a `SPARQL-LD` implementation.

SERVICE operator of SPARQL 1.1 Federated Query [5], we can invoke a portion of a query against a remote RDF repository. However, SERVICE requires the IRI to be the address of a SPARQL endpoint, thus one cannot exploit it for querying RDF data accessible on the Web but not available through an endpoint.

In this paper, we propose and demonstrate an extension of SPARQL, called `SPARQL-LD` (from Linked Data), that overcomes the aforementioned limitations enhancing thereby the flexibility of the language. `SPARQL-LD` extends the applicability of the SERVICE operator enabling to fetch and query any Web source containing RDF data (even RDF data that is created dynamically, e.g. by RESTful Web APIs). `SPARQL-LD` does not require to have declared the named graphs, thus one can even query a dataset returned by a portion of the query, i.e. its IRI is derived at query execution time. Fig. 1 illustrates an example of a query that can be answered by `SPARQL-LD`. The query returns all co-authors of P. Fafalios (1st author of this paper) together with the number of their publications and the number of distinct conferences in which they have a publication. Notice that this query combines and integrates: i) data embedded in a Web page as RDFa (lines 3-4), ii) data coming from dereferenceable IRIs *derived at query-execution time* (line 5), and iii) data coming by querying another endpoint (lines 6-8).

For answering the above query with the original SPARQL, one must download and load to a repository the triples that are contained in the Web page (line 3) as well as the triples regarding the publications of all co-authors (line 5), considering of course that these triples are not available through an endpoint. The problem may become unfeasible if these resources are derived at query-execution time, i.e. in case we are unaware of their IRIs at query-writing time. On the contrary, using `SPARQL-LD` and exploiting the Linked Data principles, such different types of resources can be directly queried without needing to retain a repository.

Note also that the proposed extension is actually a generalization of SPARQL in the sense that every query that can be answered by the original SPARQL can be also answered by `SPARQL-LD`. Specifically, if the IRI given to the SERVICE operator corresponds to a SPARQL endpoint, then it works exactly as the original SPARQL (the remote endpoint evaluates the query and returns the result). Otherwise, instead of returning an error (and no bindings), it tries to fetch and query the triples that may exist in the given resource.

Related Works. `SPARQL-LD` offers a method to execute queries over the Web of Linked Data. Such approaches can be classified in three categories: *query federation* (integrated and transparent access to distributed sources, e.g. the DARQ engine [9]), *data centralization* (query service over a collection of data copied and transformed from different sources, e.g. the MarineTLO-based Warehouse [10]), and *link traversal* (discover data related to IRIs given in the query, e.g. the work by Hartig et al. [8]). The functionality offered by `SPARQL-LD` *complements* and can be used in combination to the aforementioned approaches.

2 `SPARQL-LD`: SPARQL 1.1 Federated Query Extension

The SERVICE operator of SPARQL 1.1 (SERVICE a P) is defined (in [7]) as a graph pattern P evaluated in the SPARQL endpoint specified by the IRI a, while (SERVICE $?X$ P) is defined by assigning to the variable $?X$ all the IRIs (of endpoints) coming from partial results, i.e. that get bound after executing an initial query fragment. The idea behind `SPARQL-LD` is to enable the evaluation of a graph pattern P not absolutely in a SPARQL endpoint a, but generally in an RDF graph G_r specified by a Web Resource r. Thus, now an IRI given to the SERVICE operator can also be the dereferenceable IRI of a resource, the Web page of an entity (e.g. of a person), an ontology (OWL), Turtle or N3 file, etc. If the IRI is not the address of a SPARQL endpoint, the RDF data that may exist in the resource are fetched at real-time and queried for the graph pattern P.

Implementation. `SPARQL-LD` has been implemented using Apache Jena. Jena is an open source Java framework for building Semantic Web applications (`http://jena.apache.org/`). Specifically, we have extended Jena 2.13 ARQ component. ARQ is a query engine for Jena that supports SPARQL 1.1. The implementation is available as open source[1]. An endpoint that realizes `SPARQL-LD` (and that also includes several query examples) has been deployed for experimentation[2].

The implementation can be described through the following process: we first check if the IRI corresponds to a SPARQL endpoint by submitting the `ASK` query "`ASK {?x ?y ?z}`". In case we get a valid answer, we continue just like the default query federation approach, i.e. the corresponding graph pattern (query) is submitted to the endpoint. In case we do not get a valid answer, it means that the IRI is not the address of an endpoint. Then, we read the *content type* header field of the IRI by opening an HTTP connection and setting the value `application/rdf+xml` to the `ACCEPT` request header. Now, according to the returned content type, we fetch and query the corresponding triples. For the case of HTML Web pages (the content type is `text/html` or `application/xhtml+xml`), we try to fetch and query the RDF triples that may be embedded in the Web page as RDFa. If the Web page does not contain any RDF data, the query returns no bindings. For reading possible RDF triples in a Web page, we exploit the `Semargl` framework [3] which also offers integration with Jena. The implementation also allows to read and query JSON-LD files.

Optimizations. We adopt the following optimization techniques for saving both time for the user side and load for the server side.

-*Index of known endpoints.* We have seen that, compared to the original SERVICE operator, the only additional cost is the time to run an `ASK` query (which is in average less than 200 ms). To eliminate this cost, we keep a small index with the IRIs of known endpoints (like DBpedia's) and of endpoints that have been

[1] `https://github.com/fafalios/sparql-ld`

[2] `http://users.ics.forth.gr/~fafalios/sparql-ld-endpoint.html`

already checked. Thereby, if the SERVICE IRI exists in the index, the query is directly forwarded to the endpoint, otherwise an ASK query is first submitted.
-*Request-scope caching of the retrieved dataset(s).* A SPARQL query may contain multiple SERVICE invocations against the same Web resource. In such cases, fetching and loading repeatedly the same resource triples costs both in time and in computer resources. To avoid this, for a submitted query we use a *request-scope* cache of datasets that have been already fetched. Thereby, in each new SERVICE invocation, we first check if the corresponding IRI exists in the cache in order to avoid re-fetching its triples. The cache is cleared after query execution.

Demonstration. The ISWC'15 participants will have the opportunity to experiment with SPARQL-LD for several query scenarios including: *parameterize and call a named entity extraction Web service at query-execution time, query Web pages containing RDFa, query dereferenceable IRIs that derive at query-execution time, query ontologies, query JSOL-LD, N3 and N-Triples files.*

3 Conclusion

We have proposed SPARQL-LD, a SPARQL 1.1 extension that allows to directly fetch and query RDF data from heterogeneous sources via SERVICE calls. Using SPARQL-LD one can even query a dataset coming from query's partial results, i.e. identified at query-execution time. Such a functionality motivates Web publishers to enrich their documents with RDF since it makes their data directly accessible via SPARQL (without needing to set up an endpoint), while it also enables the direct exploitation of RDF data that is created dynamically (e.g. by RESTful Web applications). In future, we will study query planning approaches and more optimization techniques aiming to reduce the transfer of data between server/endpoint and remote sources.

Acknowledgements. This work was partially supported by DIACHRON (FP7-ICT, 2013-2016) and BlueBridge (H2020 Research Infrastructures, 2015-2018).

References

1. A JSON-based Serialization for Linked Data. http://www.w3.org/TR/json-ld/.
2. RDFa Core 1.1. http://www.w3.org/TR/2015/REC-rdfa-core-20150317/.
3. Semargl Framework. https://github.com/levkhomich/semargl.
4. SPARQL 1.1 Query Language (W3C). http://www.w3.org/TR/sparql11-query/.
5. SPARQL Federated Query. http://www.w3.org/TR/sparql11-federated-query.
6. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data-The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3), 2009.
7. C. Buil-Aranda, M. Arenas, O. Corcho, and A. Polleres. Federating queries in SPARQL 1.1: Syntax, Semantics and Evaluation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 18(1):1–17, 2013.
8. O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL Queries over the Web of Linked Data. *The Semantic Web-ISWC 2009*, 2009.
9. B. Quilitz and U. Leser. Querying distributed RDF data sources with SPARQL. In *5th European/Extended Semantic Web Conference (ESWC'08)*. Springer, 2008.
10. Y. Tzitzikas, C. Alloca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating Heterogeneous and Distributed Information about Marine Species through a Top Level Ontology. In *MTSR*, 2013.