Querying the Web of Data with XSPARQL 1.1

Daniele Dell'Aglio[1], Axel Polleres[2], Nuno Lopes[3], and Stefan Bischof[4]

[1] DEIB, Politecnico of Milano, Milano, Italy
[2] Vienna University of Economics and Business, Vienna, Austria
[3] IBM Research, Smarter Cities Technology Centre, Dublin, Ireland
[4] Siemens AG Österreich, Vienna, Austria

Abstract. On the Web and in corporate environments there exists a lot of data in various formats. XQuery and XSLT serve as query and transformation languages for XML. But as RDF also becomes a mainstream format for Web of data, transformations languages between these formats are required. XSPARQL is a hybrid language that provides an integration framework for XML, RDF, but also JSON and relational data by partially combining several languages such as XQuery, SPARQL and SQL. In this paper, we present the latest open source release of the XSPARQL engine, which is based on standard software components (Jena and Saxon) and outline possible applications of XSPARQL 1.1 to address Web data integration use cases.

1 Introduction

The task of integrating data from different sources often bumps into problems related to heterogeneity [1]: the different data sources can adopt different data formats (syntactical heterogeneity), different schemata (structural heterogeneity) and assign different meanings (semantic heterogeneity). Those problems, usually identified under the label of data variety [2], are gaining more and more attention. For example: acquired companies can hold a huge amount of data that needs to be integrated into the new parent company's systems; software refactoring that needs to consume legacy data; and data mashup services merge data from different data sets. Especially on the Web—due to its distributed structure—data heterogeneity problems often arise: data has different format, (e.g. XML and JSON), different structures (e.g. weather data from different services) and different conceptual models (e.g. prices modelled in different ways in e-commerce Web sites).

RDF is a possible solution to cope with the structural heterogeneity: it allows representing the data independently of the language it is originally formatted. A simple way to process data exploiting RDF is the following: first data is converted into RDF (i.e. lifting); 2) data is merged and stored in a RDF store; and 3) data is processed (e.g. queried or transformed). The lifting operation is strictly related to the data format: XML can be lifted through XSLT, JSON can be transformed in JSON-LD and data stored in relational databases can be converted in RDF with R2RML. The storage and the merge of RDF data is done by exploiting RDF stores like Jena-TDB or Sesame, and the processing can be done through a query language like SPARQL.

Developers can gain advantage by using frameworks that enable them to perform the aforementioned process in a simplified and integrated way. XSPARQL fills this gap: it is a scripting language that combines SPARQL [3] and XQuery [4] in one language to integrate and query data. Originally proposed as a W3C member submission in 2009 [5], the XSPARQL language is at the basis of the XSPARQL engine: the first prototype has been released as Open Source project (under BSD licence) at Sourceforge[5]. Several releases followed the initial release, and development has been accompanied by research on query optimization in the combined language [6], and various extensions [7, 8].

In this paper, we present the latest versions of the XSPARQL language and engine. In Section 2 we present the new features and provide information about the engine architecture. Next, Section 3 reports on usage of XSPARQL and illustrates the next steps.

2 XSPARQL 1.1

The XSPARQL query language is an extension of XQuery that introduces operators to query RDF data sources and format data in RDF (such as RDF/XML and Turtle). Among the available features, XSPARQL supports (1) transformations between XML/JSON and RDF (i.e., lifting and lowering), (2) control flow to SPARQL by the additional power of XQuery, (3) relational data processing through RDB2RDF[6], and (4) scripting for Web data integration in general.

The latest release of the XSPARQL language is the 1.1. The main improvements are the full support to SPARQL 1.1 and the support to JSON document processing. The former enables the construction of more complex queries, e.g. it is possible to use aggregates and federation; the latter is key to improve the usability of XSPARQL in the current Web scene, where JSON is replacing XML as serialisation format for data exchange.

An implementation of XSPARQL 1.1 is available in the latest release of the XSPARQL engine (version 20140909). As the previous releases, the software is available as open source; moreover, in addition to the support of the XSPARQL 1.1 language, this version of engine includes several bug fixes and improvements, e.g. functions to put constraints of the variable data types and new methods to declare the RDF datasets.

Anatomy of a XSPARQL query. To illustrate the new features of XSPARQL, let's consider the following running example: we want retrieve the upcoming events of artists having Nuclear Blast as label. The information we need to solve this task is available on the Web: for example, Wikipedia (and consequently DBpedia) describes the music artists and the music label they belong, while Last.fm exposes the schedule of the music events.

[5] Cf. `http://sourceforge.net/projects/xsparql/`.
[6] Cf. `http://www.w3.org/2001/sw/rdb2rdf/implementation-report/`.

The XSPARQL query in Listing 1 solves the tasks described above. The `SPARQLforClause` clause in Lines 5–11 is a special XQuery FLOWR expression that allows the developer to compose queries over RDF data: its `WHERE` clause is compliant with the SPARQL 1.1 `WHERE` clause. In the example, the `WHERE` clause contains an invocation to DBPedia to get the artists having Nuclear Blast as label.

Next, the Last.fm REST service is invoked (Line 13): for each retrieved artist at the first step, the service returns the lists of the upcoming events, serialised in JSON (details about the JSON processing are provided below).

Finally, the last part of the query (Lines 15–20) describes the output: in this case, it is the RDF graph with the upcoming concerts. To do it, a `CONSTRUCT` clause is used (Lines 15–20): it is an alternative to the XQuery `RETURN` clause, and allows specifying the triple patterns to be used to build the RDF graph. In other words, while `RETURN` produces XML, `CONSTRUCT` produces RDF.

Listing 1. XSPARQL query example

```
 1 prefix lastfm: <http://xsparql.deri.org/lastfm#>
 2 prefix dbprop: <http://dbpedia.org/property/>
 3 prefix dbpedia: <http://dbpedia.org/resource/>
 4 
 5 for *
 6 where {
 7   service <http://dbpedia.org/sparql> {
 8     $artist dbprop:label dbpedia:Nuclear_Blast ;
 9       dbpprop:name $artistName
10   }
11 }
12 return
13   let $doc := concat("http://ws.audioscrobbler.com/2.0/?artist
       =",$artistName,"&method=artist.getEvents&...")
14   for $event in xsparql:json-doc($doc)//events/event/*
15   construct {
16     [] a lastfm:Event ;
17       lastfm:artist {$artistName} ;
18       lastfm:venue {$event/venue//city} ;
19       last:date {$event/startDate}
20   }
```

Consuming and querying JSON Data. Since the availability of data in JSON increases more and more, there is a growing need of querying data in this format. XSPARQL supports JSON documents as input: this feature is useful, for example, in scenarios where JSON data has to be combined with data in other formats (e.g. RDF and XML).

JSON does not specify a query language (this representation format is meant to be incorporated directly into the JavaScript scripting language): JSON data can be manipulated directly in JavaScript; the data access is made defining paths, in a similar way to XPath for XML: the access of members of objects (denoted by '`{`' and '`}`') can be done using the '`.`' separator and the object *key*, while the access to array elements is done using the standard bracket notation: '`[`' and '`]`'. For example, let's consider JSON document provided by the Last.FM REST service in Listing 2: it contains the list of concerts of the Nightwish band. If this data is assigned to a JavaScript variable named '`b`', we can access the member '`events`' by using '`b.events`' and the second event can be done with '`b.events.event[1]`'.[7]

Listing 2. Simplified Last.fm REST service answer

```
{
  "events": {
    "event": [
    {
      "id": "3963574",
      "venue": {
        "id": "8778813",
        "name": "Electric Factory",
        "city": "Philadelphia"
      },
      "startDate": "Fri, 10 Apr 2015 21:27:01"
    },
    {
      "id": "3963451",
      "venue": {
        "id": "8778774",
        "name": "The Palladium",
        "city": "Worcester"
      },
      "startDate": "Sat, 11 Apr 2015 19:00:00"
    },
    ...
```

To access JSON documents, XSPARQL exposes the `xsparql:json-doc` function, while the query can be then declared through XPath expressions, as shown in Line 14 of Listing 1: at each iteration of the for cycle, a concert is extracted and is associated to the `$event` variable.

Design and architecture. The XSPARQL engine is written in Java and it is developed in a modular way, as depicted in Figure 1: the query rewriter processes the XSPARQL query and rewrites it in a XQuery query, while the query evaluator executes it and produces the answer.

The first component of the engine is the `xsparql-rewriter`: it takes as input a XSPARQL query, then parses, rewrites and optimises it and outputs a XQuery query [6]. The parts of the query that have to be evaluated over RDF are

[7] Please note that in JavaScript the first element of an array is at position 0, while the first element of XPath sequences is 1.

rewritten as SPARQL queries and embedded in XQuery functions. To develop this component, we used JFlex to build the lexer, and ANTLR to build the parser. We decided to develop also the rewriter and the optimiser through ANTLR: this framework offers powerful instruments to process and manipulate tree structures, so it is a suitable choice for our needs. It follows that most of the effort we put in the rewriter development has been used in the writing of the grammar files, from which the code is automatically generated.

The output of the query rewriter is the input of the following component, the `xsparql-evaluator`: as the name suggests, its main goal is the execution of the query against the data. Being the input a XQuery query with embedded SPARQL queries, the evaluation requires to execute the queries and to put together the results according to the query plan. In this sense, the query evaluator acts as an orchestrator: it executes the query plan by delegating the query evaluations to two query engines: Saxon for XQuery and Jena-ARQ for SPARQL.

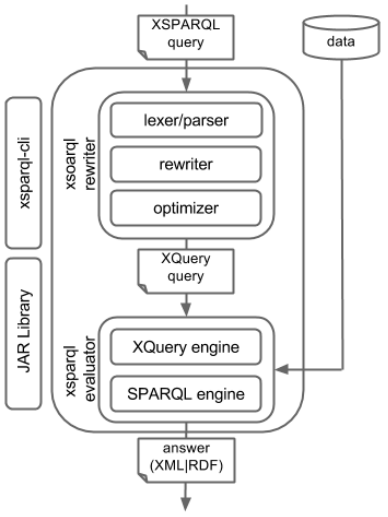

Fig. 1. XSPARQL architecture

The XSPARQL engine can be used in two different ways: as stand-alone application and as library. The stand-alone application, developed in the `xsparql-cli` project, can be executed through terminal, and offers a command line interface to set the query and the data that have to be used. Alternatively, XSPARQL can be used as library: the `xsparql-rewriter` and the `xsparql-evaluator` are also delivered as JAR files, available through Maven[8]: in this way, XSPARQL can be used in a programmatic fashion by other projects and applications.

3 Conclusions and future directions

The design of XSPARQL was lead by a set of use cases [9]. In the last years, XSPARQL has been used in several projects, to cope with the data integration issues. For example, XSPARQL has been used for RDB2RDF direct mapping and as an R2RML processor[9]. Another example is the usage of XSPARQL in the European research project digital.me[10]. The goal of the project is the integration of users personal information sphere. XSPARQL was used in the data transformation process, where social data is extracted from social networks,

[8] `http://oss.sonatype.org/content/repositories/releases/net/sf/xsparql/`

[9] Cf. `http://www.w3.org/TR/r2rml/`.

[10] Cf. `http://www.dime-project.eu/`.

such as Facebook, LinkedIn, Twitter, Google+ and any OpenSocial-compliant service, and is then mapped to an interoperable standard. XSPARQL enabled the construction of ontology-based technique for online profile resolution [10], which targets the discovery of multiple online profiles that refer to the same person identity[11].

XSPARQL is mature enough to be used in production systems and its development is ongoing. In the next months, we plan to add several features and improve the existing ones. For instance, we plan to extend support for JSON with more convenient syntax shortcuts, functions and support to JSON-LD [11]. Additionally, we plan to enable support for managing and querying streaming data: the processing of high-dynamic data from the Web is a challenging and ongoing trend that is gaining more and more attention.

Acknowledgments

This work has been partially funded by the WWTF Project (ICT12-015).

References

1. Ouksel, A.M., Sheth, A.P.: Semantic interoperability in global information systems: A brief introduction to the research area and the special section. SIGMOD Record **28**(1) (1999) 5–12
2. Laney, D.: 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group (February 2001)
3. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language (March 2013)
4. Boag, S., Chamberlin, D., Fernandez, M.F., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML Query Language (Second Edition) (December 2010)
5. Polleres, A., Krennwallner, T., Lopes, N., Kopecký, J., Decker, S.: XSPARQL Language Specification (January 2009) W3C member submission.
6. Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A.: Mapping between RDF and XML with XSPARQL. Journal on Data Semantics **1**(3) (2012) 147–185
7. Lopes, N., Bischof, S., Polleres, A.: On the semantics of heterogeneous querying of relational, XML, and RDF data with XSPARQL. In: Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA2011) – Computational Logic with Applications Track, Lisbon, Portugal (October 2011)
8. Bischof, S., Lopes, N., Polleres, A.: Improve efficiency of mapping data between XML and RDF with XSPARQL. In: Web Reasoning and Rule Systems – Fifth International Conference, RR2011. Volume 6902 of Lecture Notes in Computer Science (LNCS)., Galway, Ireland, Springer (August 2011) 232–237 Short paper.
9. Passant, A., Kopecký, J., Corlosquet, S., Berrueta, D., Palmisano, D., Polleres, A.: XSPARQL: Use cases (January 2009) W3C member submission.
10. Cortis, K., Scerri, S., Rivera, I., Handschuh, S.: An ontology-based technique for online profile resolution. In: Social Informatics. Volume 8238 of Lecture Notes in Computer Science. Springer International Publishing (2013) 284–298
11. Sporny, M., Kellogg, G., Lanthaler, M.: JSON-LD 1.0 (January 2014)

[11] Examples of the XSPARQL queries can be found in `http://www.dime-project.eu/resources/news/dime_257787_D03.03.pdf` within Section 4 (pgs 19-20), and the Appendix (pg31+)