XSPARQL-Viz: A Mashup-based Visual Query Editor for XSPARQL ⋆

Syed Zeeshan Haider Gillani, Muhammad Intizar Ali, and Alessandra Mileo

DERI, National University of Ireland, Galway, Ireland
{syed.gillani,ali.intizar,alessandra.mileo}@deri.org

Abstract. XSPARQL is a query language which facilitates query, integration and transformation between XML and RDF data formats. Although XSPARQL supports semantic data integration by providing uniform access over XML and RDF, it requires users to be familiar with both of its underlying query languages (e.g XQuery and SPARQL). In this system demo, we show how mashup-based techniques can be used for auto generation and execution of XSPARQL queries. XSPARQL-Viz provides an easy to use *drag and drop* visual query editor, which supports novice users in designing complex mappings between XML and RDF and based on these mappings auto generates and executes XSPARQL queries. Results can also be visualised as a graph, table or list.

1 Introduction

XSPARQL[1] is a query language that combines XQuery and SPARQL for mapping, querying and exchanging XML and RDF on the Web [1]. Despite both XQuery and SPARQL are widely adopted for XML and RDF data respectively, many modern data integration applications require interoperability and simultaneous access over both data formats. XSPARQL bridges this gap by providing uniform access over XML and RDF data, enabling query and transformation from one data format to the other using a single query. Figure 1 depicts a simplified version of the XSPARQL architecture.

Different approaches for the integration of XML and RDF data have been proposed in the literature [6, 3]. However all proposed solutions, including XSPARQL, require the user to be familiar with both XQuery and SPARQL. Complex query language syntax is the main hindrance in the wide adoption of XSPARQL, because the majority of users are domain experts of their respective field but lack expert knowledge of both query languages. Query-by-example, query-by-form, graphical user interface and assisted query editors have been proposed and used for auto query generation of SQL, XQuery and SPARQL. Similarly, mashup are web based ad hoc applications that consume the available data from third parties and combine them to build a new application. However mashup

⋆ This work has been funded by Science Foundation Ireland, Grant No. SFI/08/CE/l1380 (Lion2).

[1] http://www.w3.org/Submission/xsparql-implementation/

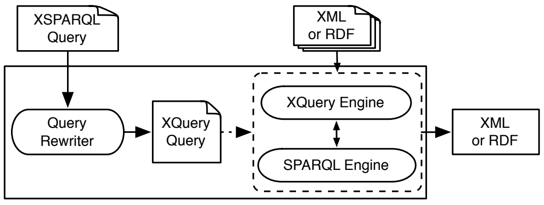


Fig. 1. XSPARQL Architecture[1]

mostly consume only web feeds or API's , hence, lacking the query language capabilities. Concept of data mashup has been utilised for auto query generation of SPARQL and for integration of XML and RDF [7, 4]. Following the same principles, we design XSPARQL-Viz (a mashup-based visual query editor) for auto generation of XSPARQL queries. We believe that potential impact, adoption and usability of XSPARQL can be highly increased by providing such support for novice users to write XSPARQL queries, and our proposed Mashup-based XSPARQL visual query editor represents a step forward in this direction.

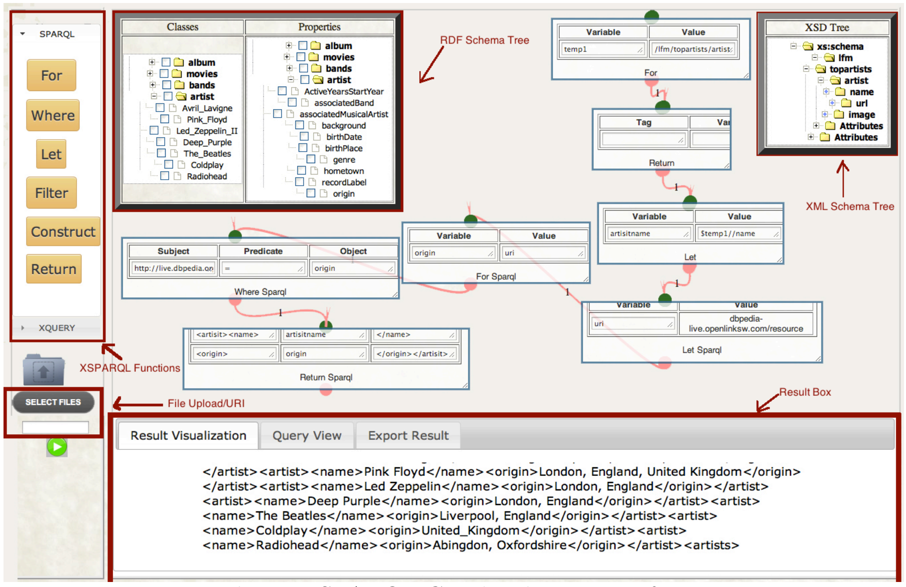
In this paper, we demonstrate a system demo which provides (i) *auto generation of the XSD schema for XML data sets and RDFS schema for RDF data sets,* (ii)*a visual query editor for facilitating mapping between XML and RDF data sets* (iii) *mashup-based approach for auto generation of XSPARQL queries, and* (iv) *tranformation of query results into any desired output format or as input for another query.*

2 System Overview

Our system is written in Java using the Spring Framework. A request based Model-View-Controller (MVC) Framework is built over Spring Inversion of Control (IoC) Framework, which provides layered architecture for a stricter separation between model, view and controller. The underlying layered services can be categorised into three classes (i) *XML Schema Generation Services,* (ii) *RDF Schema Generation Services,* (iii) *Query Generation Services, and*(iv) *Result Generation Services.*

The XSPARQL engine is used as a core engine for query and transformation between XML and RDF data, while XSPARQL query engine utilises Saxon[2] query engine for XQuery and ARQ[3] query engine for SPARQL queries execution. The visual query editor is a web application accessible using any web browser, which is designed using a combination of Jsplumb and Jquery.

[2] http://www.saxonica.com
[3] http://jena.apache.org/documentation/query/index.html



```
</artist><artist><name>Pink Floyd</name><origin>London, England, United Kingdom</origin>
</artist><artist><name>Led Zeppelin</name><origin>London, England</origin></artist>
<artist><name>Deep Purple</name><origin>London, England</origin></artist><artist>
<name>The Beatles</name><origin>Liverpool, England</origin></artist><artist>
<name>Coldplay</name><origin>United_Kingdom</origin></artist><artist>
<name>Radiohead</name><origin>Abingdon, Oxfordshire</origin></artist><artists>
```

Fig. 2. XSPARQL Graphical User Interface

3 Demonstration Scenario

Inspired by [9], we consider the usecase of integrating Last.fm[4] music data with DBpedia[5]. Consider the scenario of a user wanting to know about the hometown of her top artists. Last.fm contains music information about a particular user e.g type of music or list of top artists listened by the user, while DBpedia extracts structured data from Wikipedia and stores them in RDF format. Last.fm provides access to its data using public api's or web services and returns the results in XML format, while DBpedia is accessible through SPARQL endpoint. Hence, in order to answer the question above, we need to deal with data heterogeneity and here is where XSPARQL comes handy, enabling us to retrieve, integrate and transform XML and RDF data within a single query.

```
prefix dbprop: <http://dbpedia.org/property/>
let $doc :="http://ws.audioscrobbler.com/2.0/?method=user.gettopartists"
        for $artist in doc($doc)//artist
        return
let $artistName := fn:data($artist//name)
let $uri := fn:concat("http://dbpedia.org/resource/", $artistName)
        for $origin from $uri
        where { [] dbprop:origin $origin }
        return
<artists>
        <artist>
                <name>{$artistname}</name>
                <origin>{$origin}</origin>
        </artist>
<artists>
```

Listing 1. A Sample XSPARQL Query

[4] http://last.fm
[5] http://dbpedia.org

Listing 1 provides an XSPARQL query to get the desired answers. However, in order to write such XSPARQL query one needs to know both its underlying query languages. We will use this use case scenario to depict several components of XSPARQL-Viz and generate mashup for the execution of the XSPARQL query. Figure 2 demonstrates the mashup for the generation and execution of XSPARQL query shown in Listing 1.

Uploading Data Sets: XSPARQL-Viz requires access to the data where the answer to our query lies. Lower left corner of the XSPARQL-Viz Editor as shown in Figure 2, allows user to register the data sources by either uploading an XML or RDF data set from local directory, or by providing a URI of a remotely available data set. Users can upload multiple data sets in one session which are temporarily stored for each user's session.

Schema Generations: Once data sets are uploaded or their URI's are provided, Schema Builder will execute multiple queries to automatically generate XML Schema for XML data sets and RDF Schema for RDF data sets. The generated schema will be displayed in tree format at the top center of the XSPARQL-Viz Editor, as shown in Figure 2. The generated tree structure component provides also an option to change the view into different visual structures, including lists, grids and graphs.

Query Editor: XSPARQL-Viz Editor provides a richer graphical front-end on top of the XSPARQL engine. This front-end enables users not only to execute custom queries but also to have drag-and-drop support from schema tree for efficiently designing the complex mapping between XML and RDF data sets. The left panel of the XSPARQL-Viz Editor in Figure 2 allows users to choose any of the XSPARQL clauses/statements to be applied over the data sets. In order to set variable values in the query, users can either drag and drop any data value from a generated schema tree or provide direct input. The flow of the query is maintained by combining various components of XSPARQL clauses. XSPARQL-Viz editor separates tabs for SPARQL and XQuery clause, and also provides separate tabs for aggregate and other functions available in XSPARQL.

Output Data Format: XSPARQL serves as a bridge for transformation of XML and RDF data in both directions, which allows users to pick any of the RETURN or CONSTRUCT clause for the output data format. This feature enables to perform transformations either by lowering (from RDF to XML) or by lifting (from XML to RDF). Depending on users' choice, the output of an XSPARQL query will be in XML if the RETURN clause is used, or in RDF if the CONSTRUCT clause is used.

Visualisation of Results: XSPARQL-Viz Editor provides visualisation capabilities for the results of the mashup to be displayed as graphs, tables or lists.

User can also export the results as an external file or data sets. A very important additional benefit of using mashup-based approach is that it allows to save the mashup as a component within XSPARQL-Viz Editor, and to use it later for similar query execution over different data set or use it as input for another query.

4 Potential Impact and Future Directions

In this demo, we showcase the XSPARQL visual query editor for auto-generation of XSPARQL queries. XSPARQL-Viz assists for mapping and transformation between XML and RDF data without having extensive knowledge of XQuery and SPARQL. XSPARQL-Viz is built upon and compliant with the strong theoretical foundations of XSPARQL defined in [1, 5]. We believe that initiatives like XSPARQL-Viz can play a pivotal role for the wide adaption and usability of XSPARQL in various domains and real-world applications.

As next steps we plan to perform extensive evaluation of XSPARQL-Viz usability, and extend it to incorporate two recent extensions of XSPARQL, namely (i) XSPARQL over relational data [8], and (ii) XSPARQL Update Facility [2]. We are also considering to use XSPARQL-Viz as a platform for adaptive generation of data set statistics which can be later used for query analysis, planning and optimisation.

References

1. W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF Worlds - and Avoiding the XSLT Pilgrimage. In *Proc. of ESWC 2008*, pages 432–447, 2008.
2. M. I. Ali, N. Lopes, O. Friel, and A. Mileo. Update Semantics for Interoperability Among XML, RDF and RDB. In *Proc. of APWeb*, 2013.
3. M. I. Ali, R. Pichler, H. L. Truong, and S. Dustdar. DeXIN: An Extensible Framework for Distributed XQuery over Heterogeneous Data Sources. In *Proc. of ICEIS 2009*, LNBIP, pages 172–183. Springer, 2009.
4. M. I. Ali, R. Pichler, H. L. Truong, and S. Dustdar. On integrating data services using data mashups. In *Proc. of BNCOD*, volume 7051 of *Lecture Notes in Computer Science*, pages 132–135. Springer, 2011.
5. S. Bischof, S. Decker, T. Krennwallner, N. Lopes, and A. Polleres. Mapping between RDF and XML with XSPARQL. *Journal on Data Semantics*, 1:147–185, 2012.
6. S. Groppe, J. Groppe, V. Linnemann, D. Kukulenz, N. Hoeller, and C. Reinke. Embedding SPARQL into XQuery/XSLT. In *Proc. of SAC*, 2008.
7. M. Jarrar and M. D. Dikaiakos. A data mashup language for the data web. In *Proc. of LDOW*, volume 538 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
8. N. Lopes, S. Bischof, S. Decker, and A. Polleres. On the Semantics of Heterogeneous Querying of Relational, XML and RDF Data with XSPARQL. In *Proc. of EPIA*, 2011.
9. A. Polleres and S. Sakr. Querying and Exchanging XML and RDF on the Web. WWW 2012 Tutorial-http://www2012.wwwconference.org/program/tutorials/.