Querying Federations of SPARQL Endpoints and JSON-based Web APIs with HeFQUIN

Olaf Hartig[1] and Jonas Westman[2]

[1] Linköping University, Sweden
[2] TRATON, Sweden

Abstract. This paper demonstrates the capability of the HeFQUIN federation engine to integrate JSON-based Web APIs as federation members. Hence, the engine can run SPARQL queries that join data from such APIs with RDF data from SPARQL endpoints, where the JSON data from the APIs is mapped into RDF at runtime. Instead of imposing a fixed mapping for this process, HeFQUIN supports arbitrary user-defined mappings, which can be described declaratively using RML.

1 Introduction

Federated query processing is a powerful paradigm that enables applications to run structured queries over a virtual combination of autonomously maintained datasets, without requiring central coordination of the respective maintainers. While early work on this paradigm in the Semantic Web community has focused mostly on federations consisting solely of SPARQL endpoints [8] or of Linked Data documents [3], in more recent years, we have seen approaches that consider federations of multiple types of RDF-based data sources [1,5] and even of other forms of query-language-based data sources such as relational databases [2].

Another very common type of data sources are (HTTP-based) Web APIs. The data available via such APIs—not only on the open Web but also within the intranet of an enterprise—can provide an important complement to data in directly-accessible knowledge graphs and databases. Therefore, in this paper we introduce a novel approach to enable SPARQL federation engines to support such APIs as another type of federation member. We have implemented this approach for JSON-based Web APIs in our HeFQUIN engine and will showcase this functionality via live Web queries during the demo session of the conference.

2 Approach

To express that some part of a SPARQL query is meant to be evaluated over data from a Web API, our approach builds on the notion of SPARQL `SERVICE` clauses and extends it in two ways: First, instead of a SPARQL endpoint, the service IRI of a `SERVICE` clause may refer to a Web API, which is specified in a so-called federation description that is provided to the query processor. In such a case, the evaluation of the graph pattern of the `SERVICE` clause is defined over

```
PREFIX dbo:    <http://dbpedia.org/ontology/>
PREFIX geo:    <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX ex:     <http://example.org/>

SELECT * WHERE {
 SERVICE <http://dbpedia.org/sparql> {
    ?city rdf:type dbo:City ;
          geo:lat ?la ;  geo:long ?lo ;
          dbo:country <http://dbpedia.org/resource/Germany>  }
 SERVICE <http://example.org/OpenMeteoWrapper> PARAMS(?lo AS "longitude",
                                                      ?la AS "latitude") {
    ?x ex:temperature ?temp ;  ex:windSpeed ?wind  }
}
```

Listing 1. A SPARQL query that retrieves the current temperature and wind speed for every city in Germany for which the DBpedia knowledge graph contains geographic coordinates. The query joins data from the DBpedia SPARQL endpoint with live weather data from the Web API of the Open-Meteo weather service.

an RML-based RDF view of data obtained via a request to the Web API. The second part of our extension is the introduction of a so-called `PARAMS` clause that may be added between the service IRI and the graph pattern of a `SERVICE` clause. As an example of this feature, consider the second `SERVICE` clause of the query in Listing 1. In the following, we explain the two parts of our extension of `SERVICE` clauses in more detail, beginning with the purpose of `PARAMS` clauses.

For every Web API that may be used as a member of a federation to be queried, the query processor must be provided with a definition of how request URIs for that API can be formed. In our approach, such definitions are captured in an RDF-based federation description and they consist of a URI template, together with a specification of the parts of this template that are to be instantiated when forming a URI from the template. Listing 2 illustrates such a federation description, with a URI template in line 15 and the definition of its two variable parts in lines 16–19.[3] Overall, the description in Listing 2 introduces the Web API of the Open-Meteo weather service as a federation member to be considered by the approach. This federation member can then be referred to in `SERVICE` clauses of SPARQL queries by using the service IRI given in line 8 of Listing 2, as done in the second `SERVICE` clause of the example query in Listing 1.

The purpose of the `PARAMS` clause is to identify the SPARQL query variables whose values are to be used for instantiating the URI template and, more specifically, to define for which variable part of the template the value(s) of a query variable are meant to be used. Every such query variable must be bound by a graph pattern that comes before the `PARAMS`-based `SERVICE` clause. For the example query in Listing 1, this is the case. Then, during the execution of this query, a solution mapping such as, say, $\mu_{\text{ex}} = \{$`?la` $\mapsto$ `"52.52"^^xsd:float`,

[3] The format of the URI templates adheres to the RFC 6570 standard. Therefore, even if the example template discussed in this paper contains variable parts only for query arguments of the URIs to be created, it is also possible to have URI templates with variable parts in the hostname part and the path part of URIs.

```
1  PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
2  PREFIX fd:     <http://w3id.org/hefquin/feddesc#>
3  PREFIX hydra:  <http://www.w3.org/ns/hydra/core#>
4  PREFIX rml:    <http://w3id.org/rml/>
5  PREFIX ex:     <http://example.org/>
6
7  ex:openMeteo a fd:WrappedFederationMember ;
8               fd:serviceURI <http://example.org/OpenMeteoWrapper> ;
9               fd:interface  _:openMeteoRestEndpoint ;
10              fd:wrapper    _:openMeteoWrapper .
11
12 _:openMeteoRestEndpoint a fd:TemplateBasedInterface ;
13     fd:supportedProtocol fd:GenericWebAPIProtocol ;
14     fd:uriTemplate [
15         hydra:template "https://api.open-meteo.com/v1/forecast?current= ↩
           ↪ temperature_2m,wind_speed_10m{&latitude,longitude}" ;
16         hydra:mapping [ hydra:variable "latitude" ;
17                         hydra:required true ; fd:paramType xsd:float ] ;
18         hydra:mapping [ hydra:variable "longitude" ;
19                         hydra:required true ; fd:paramType xsd:float ] ] .
20
21 _:openMeteoWrapper fd:rmlTriplesMaps ( _:openMeteoTM1 ) .
22
23 _:openMeteoTM1 rml:logicalSource [ rml:source _:openMeteoRestEndpoint ;
24                                    rml:referenceFormulation rml:JSONPath ;
25                                    rml:iterator "$.current" ] ;
26     rml:subjectMap [ rml:termType rml:BlankNode ] ;
27     rml:predicateObjectMap [
28                  rml:predicateMap [ rml:constant ex:temperature ] ;
29                  rml:objectMap [ rml:reference "temperature_2m" ;
30                                  rml:datatype xsd:double ] ] ;
31     rml:predicateObjectMap [
32                  rml:predicateMap [ rml:constant ex:windSpeed ] ;
33                  rml:objectMap [ rml:reference "wind_speed_10m" ;
34                                  rml:datatype xsd:double ] ] .
```

Listing 2. A snippet of an RDF-based federation description (presented in the RDF Turtle syntax) that introduces the Web API of the Open-Meteo weather service as a federation member to be considered by our approach. The description defines the corresponding IRI to be used in `SERVICE` clauses (line 8); the URI template for creating the request URIs (lines 14–18); and the RML mapping (lines 23–34).

`?lo` $\mapsto$ `"13.41"^^xsd:float`$\}$ arrives at the query plan operator responsible for the second `SERVICE` clause of the query. The operator uses the values in this solution mapping to instantiate the URI template of Listing 2, which results in the following request URI for the Open-Meteo Web API.

https://api.open-meteo.com/v1/forecast?current=temperature_2m,wind_speed_10m&latitude=52.52&longitude=13.41

For this URI, the API returns a JSON document with the current temperature and wind speed at the specified geographic coordinates (see Listing 3).

The evaluation semantics of our new type of `SERVICE` clause for Web APIs is then defined informally as follows: For every input solution mapping μ to such a `SERVICE` clause (such as the solution mapping μ_{ex} in the example above), if a request URI can be created from μ and JSON data is retrieved for this URI, then the graph pattern of the `SERVICE` clause is evaluated over an RML-based

```
{  "latitude" : 52.52,
      ...
   "elevation" : 38.0,
   "current_units" : { "time" : "iso8601",
                           ...
                       "wind_speed_10m" : "km/h" },
   "current" : { "time" : "2026-02-28T21:00",
                 "interval" : 900,
                 "temperature_2m" : 13.1,
                 "wind_speed_10m" : 12.6 }  }
```

Listing 3. Example of JSON data returned by the Open-Meteo API.

RDF view of this JSON data and the solution mappings from this evaluation are joined with μ. For our Open-Meteo example, the RML mapping to be used is specified in lines 23–34 of Listing 2; applying this mapping to the JSON response in Listing 3 results in the following two RDF triples (written in Turtle syntax).

```
_:b ex:temperature "13.1"^^xsd:double ; ex:windSpeed "12.6"^^xsd:double .
```

Evaluating the graph pattern of the second `SERVICE` clause of the example query (Listing 1) over this RDF view results in a single solution mapping: $\mu'_{\text{ex}} = \{$`?x` $\mapsto$ `_:b`, `?temp` $\mapsto$ `"13.1"^^xsd:double`, `?wind` $\mapsto$ `"12.6"^^xsd:double`$\}$. Thus, the μ_{ex}-specific part of the complete result of the second `SERVICE` clause is the solution mapping $\mu''_{\text{ex}} = \mu_{\text{ex}} \cup \mu'_{\text{ex}}$. The other parts of this result are produced by processing the other input solution mappings to the `SERVICE` clause in the same way is done for μ_{ex}. Input solution mappings for which creating the corresponding request URI fails (as they may not have the right types of values for the URI template), or for which the request itself fails, are ignored.

Finally, while we describe the approach using a JSON-based example and our implementation is currently only for JSON, the approach itself is not limited to JSON; instead, it can be applied to any form of data supported by RML.

3 Implementation

To implement the approach in HeFQUIN we have extended the SPARQL parser to recognize `PARAMS` clauses, added new logical and physical operators to capture the new functionality in logical and physical query plans, and developed a dedicated RML-processing component. This RML component employs our recently-introduced algebra for declarative mapping languages [7]. That is, during startup, the component translates any given RML mapping into an expression of this algebra by applying the translation algorithm of our earlier work [7, Algorithm 1].[4] At runtime, such an algebra expression is translated into a physical mapping plan that, then, is executed to convert the retrieved JSON data into RDF. An additional feature of the RML component is that it can extract whole JSON arrays and whole JSON sub-objects as composite values in the form of RDF literals; these composite values can then be queried further in SPARQL [4].

[4] In fact, while the translation in our earlier work was designed for version 1.1.2 of RML, for HeFQUIN we have adapted this algorithm to the new RML vocabulary [6].

HeFQUIN is available as an open-source project and can be used as a standalone engine where queries can be issued via a command-line interface or via a Web service; alternatively, the engine can also be integrated directly as a component of a Java application. The different usage options, the features (including the integration of Web APIs), and the vocabulary for federation descriptions are described in the documentation on the HeFQUIN Website.[5]

4 Demonstration

For the demo at the conference we have prepared a number of example queries.[6] These queries include not only the running example of this paper (i.e., Listing 1), but also examples that consider other data sources and more data sources per query. Participants can run these queries, and also modify the queries as well as the corresponding federation descriptions (including the RML mappings) to observe the effects of these modifications. Moreover, for participants interested in the internals, HeFQUIN provides options to inspect logical and physical query plans and to print comprehensive query processing statistics (to the level of each individual operator within the executed plan).

Acknowledgments. The presented work was supported by the Knut and Alice Wallenberg Foundation (KAW 2023.0111) and by Vetenskapsrådet (project 2025-06246).

References

1. Cheng, S., Hartig, O.: FedQPL: A Language for Logical Query Plans over Heterogeneous Federations of RDF Data Sources. In: Proceedings of the 22nd Int. Conf. on Information Integration and Web-based Applications & Services (iiWAS) (2020)
2. Gu, Z., Corcoglioniti, F., Lanti, D., Mosca, A., Xiao, G., Xiong, J., Calvanese, D.: A Systematic Overview of Data Federation Systems. Semantic Web **15**(1) (2024)
3. Hartig, O.: An Overview on Execution Strategies for Linked Data Queries. Datenbank-Spektrum **13**(2) (2013)
4. Hartig, O.: The HeFQUIN Approach to Extract and to Process Composite Values with RML and SPARQL. In: Proceedings of the 7th International Workshop on Knowledge Graph Construction (KGCW) (2026)
5. Heling, L., Acosta, M.: Federated SPARQL Query Processing over Heterogeneous Linked Data Fragments. In: Proceedings of the ACM Web Conference 2022 (2022)
6. Iglesias-Molina, A., Assche, D.V., Arenas-Guerrero, J., Meester, B.D., Debruyne, C., Jozashoori, S., Maria, P., Michel, F., Chaves-Fraga, D., Dimou, A.: The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. In: Proceedings of the 22nd International Semantic Web Conference (ISWC) (2023)
7. Min Oo, S., Hartig, O.: An Algebraic Foundation for Knowledge Graph Construction. In: Proceedings of the 22nd Europ. Semantic Web Conference (ESWC) (2025)
8. Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngomo, A.N.: A Fine-Grained Evaluation of SPARQL Endpoint Federation Systems. Semantic Web **7**(5) (2016)

[5] https://liusemweb.github.io/HeFQUIN/
[6] https://liusemweb.github.io/HeFQUIN/doc/examples.html