Towards Federated Queries for Web of Things Devices

Sejal Jaiswal
Université. Jean Monnet, CNRS,
F-42023 Saint-Étienne, France
cj.sejal@gmail.com

Maxime Lefrançois
Univ. Lyon, Mines Saint-Étienne, CNRS, Laboratoire
Hubert Curien UMR 5516,
F-42023 Saint-Étienne, France
maxime.lefrancois@emse.fr

ABSTRACT

In the emerging Web of Things (WoT), a vast majority of devices do not consume nor produce RDF, notably because of their inherent constraints that prevent them from manipulating textual RDF syntax. However, it would be relevant to query against these sources irrespective of the (lightweight) formats they use, combined with other Linked Data. In fact the data these WoT devices expose contain crucial real-time information, and being able to tap directly into this information could enable smarter industrial applications. In this paper, we are interested in querying indifferently SPARQL endpoints, RDF documents, and non-RDF document exposed by WoT devices in a flexible and extensible way. The core of this solution is an extension of SPARQL-LD. The SPARQL SERVICE clause is extended and can be used to query also non-RDF sources for which we know some RDF lifting mechanism.

1 INTRODUCTION

Even though RDF was adopted as a W3C recommendation for data interchange on the Web, not all data producer in the IoT/WoT industry follow RDF as a standard model. Constrained devices or things on the WoT ecosystem tend to prefer lightweight formats, mostly binary such as EXI[1] or CBOR[2] due to their inherent bandwidth, memory, storage, and/ or battery constraints. However, it would be relevant to query against these sources irrespective of the formats they use, combined with other Linked Data. In fact the data these WoT devices expose contain crucial real-time information, and being able to tap directly into this information could enable smarter industrial applications.

The *SPARQL* query language enables to retrieve and manipulate a RDF dataset, which consists of a default graph, and set of named graphs. SPARQL 1.1 Federated Query [14] introduces the SERVICE clause, thanks to which a query author can direct a portion of a query to a particular SPARQL endpoint that is potentially working with a different RDF dataset. SPARQL-LD [5] extends the applicability of the SERVICE operator to RDF Sources so as to exploit the Web of Linked Data: the SPARQL engine attempts to retrieve the RDF Graph located at the endpoint URL, and executes the portion of the query against this RDF Graph.

In this paper, we propose to extend this solution further for querying also non-RDF Web resources for which some RDF lifting mechanism is known. This allows one to semantically query WoT devices while allowing them to expose the data in the format and structure they prefer. Doing so also allows for standardized access to all formats of data through the use of SPARQL. This shall be helpful for the current industry to adapt to the principles of Semantic Web without having to change much their existing solutions. Such an extended SERVICE clause allows to write queries with portions that are explicitly targeting some specific WoT devices that host their own HTTP server. This allows for low-level querying scenarios such as *"What temperature value does sensor x observe?"*. On the other hand, we argue that a higher level of abstraction could be beneficial for other querying scenarios such as *"What is the temperature on the second floor?"*.

The rest of the paper is organized as follows: Section 2 introduces a motivating example for the proposed solution. Section 3 discusses related work done in the field. Section 4 describes the proposed extension of SPARQL-LD. Section 5 describes how the proposed solution can be integrated with principles of query federation to query non-RDF data sources as well. Section 6 reuses the motivating example to explain the working of the solution. Section 7 discusses in brief the evaluation, implementation. Finally, Section 8 concludes the paper and suggests future works planned for further enhancement of the solution.

2 MOTIVATING EXAMPLE

Consider a WoT-enabled smart office[3] with two floors. For simplicity, we consider the case of only the 2nd Floor with just one room that has connected sensors and actuators. Floor 2 (`<room/2>`) houses a heater that exposes temperature property (`<room/2#temperature>`) and an occupancy sensor with occupancy property (`<room/2#occupancy>`). We assume that some description of the building and the devices is available in a Data Catalogue [Section 5.1] and the data generated by the devices and it's sensors is hosted on some URL. Listing below is a condensed version of the Data Catalogue and shows some details for Floor 2. The namespaces are those available with the online service http://prefix.cc/.

```
<room/2> a seas:Room , sosa:FeatureOfInterest ;
  ssn:hasProperty <room/2#temperature> ;
  seas:onFloor 2 .
```

[1] https://www.w3.org/TR/exi/
[2] https://tools.ietf.org/html/rfc7049
[3] https://w3c.github.io/wot-architecture/

SEMANTiCS 2017 workshops proceedings: SIS-IoT
September 11-14, 2017, Amsterdam, Netherlands

```
<room/2#temperature> a sosa:ObservableProperty,
      seas:TemperatureProperty .
<Heater/2> a sosa:Sensor, rdfp:RDFSource ;
  sosa:observes <room/2#temperature> ;
  rdfp:describedBy [
    a rdfp:GraphDescription ;
    rdfp:validationRule """
    ?temp_obs a sosa:Observation ;
      sosa:hasFeatureOfInterest ?room ;
      sosa:observedProperty ?temperature ;
      sosa:hasSimpleResult ?temp_value ."""^^ex:bgp ;
    rdfp:presentedBy [ a rdfp:GraphPresentation ;
      rdfp:mediaType "application/json" ;
      rdfp:liftingRule <lifting-rule-1.rqg> ] .
```

We want to allow an end user to query the devices by launching a SPARQL query such as the one in Listing below without having to worry about the various data formats used by the devices/sensors or even the distributed nature of the data sources. This query should answer the question: *"What are the rooms that have the property 'temperature' and 'occupancy'? What values do the sensors that have these property depict?"*

```
SELECT ?room WHERE {
  ?room a seas:Room , sosa:FeatureOfInterest ;
    ssn:hasProperty ?temperature, ?occupancy ;
    seas:onFloor ?floor .
  ?temperature a sosa:ObservableProperty, seas:TemperatureProperty .
  ?occupancy a sosa:ObservableProperty, seas:OccupancyProperty .
  ?temp_obs a sosa:Observation ;
    sosa:hasFeatureOfInterest ?room ;
    sosa:observedProperty ?temperature ;
    sosa:hasSimpleResult ?temp_value .
  ?occ_obs a sosa:Observation ;
    sosa:hasFeatureOfInterest ?room ;
    sosa:observedProperty ?temperature ;
    sosa:hasSimpleResult ?occ_value .
}
```

There are multiple problems to be tackled here, such as the physical setup during deployment, setting up APIs to access the sensor's data along with the content lifting rule [12], publishing of the data catalogue, etc. However our major focus for this paper is integrating the proposed solution with query federation principles, such that it allows us to query the light-weight format used by the sensors and the devices to expose their data.

3 RELATED WORK

To execute queries over the Web of Linked Data, two main infrastructure exist based on the data source location: *central repository* and *distributed repository*. With central repository, query service is provided over a repository where data is collected from various sources on the Web. For distributed repository, the data need not be available at a single location for query service. Distributed method of data access can be further divided into two different querying approaches: *Link Traversal* [8] and *federation* [6]. In link traversal, data is discovered by following HTTP URIs. Link Traversal could also be an efficient method for query federation [1], however the scenario we work with will not be able to exploit the advantages of Link Traversal fully due to the lack of referenced links within a link to identify further data. Hence, we choose only Federated Query principles for federating the results from various sources. Federated query is the ability to take a query and provide solutions based on information from many different sources. In query federation, a user query is transformed into several sub-queries and the result is generated by combining the intermediate results from the integrated data sources. There are studies done explaining the infrastructure of federation query [6] as well as studies focused on the basis of federation query processing strategy [9].

SPARQL-LD [5] is an extension of *SPARQL 1.1 Federated Query* [14] that exploits the Web of Linked Data by extending the applicability of the SERVICE operator. This enables to query any HTTP Web source containing RDF data (what is called RDF Source in RDF 1.1).We shall use an extended version of SPARQL-LD to exploit the real-time and dynamic nature of Linked Data in our proposed solution.

One of the primary visions of Semantic Web is to enable machines to exchange and process web content easily. This vision is hampered by the coexistence and usage of many heterogeneous data formats and models. For data conversion from various formats to RDF (called RDF *lifting*) which can then be easily queried using SPARQL, we have used principles of SPARQL-Generate [11, 12] with respect to constrained devices that produce binary data. SPARQL-Generate is based on SPARQL and leverages its expressible and extensible nature to be able to support RDF lifting for new data formats. Although we chose to use SPARQL-Gnerate, one could design similar solutions that make use of any other existing RDF-lifting mechanisms such as JSON-LD contexts [16], RML mappings [4], or XSPARQL rules [13].

The benefits of extending SPARQL-LD with the principles of RDF lifting is that, we shall be able to integrate in the same SPARQL query: i) RDF data stored in RDF dataset, ii) data from SPARQL endpoints, iii) RDF data fetched from any RDF source (in any of the RDF syntax) and iv) non-RDF data obtained in any arbitrary data format, but for which a RDF lifting mechanism is known.

4 QUERYING DOCUMENTS IN ARBITRARY FORMATS

In order to query one or more SPARQL Protocol services, one can use the principles of the SPARQL 1.1 the SERVICE operator for federated SPARQL queries [3]. SPARQL 1.1 Federated Query allows for combining group graph patterns that are to be evaluated over several SPARQL Protocol services within a single query. The endpoints of the services to be queried are provided as parameters to the SERVICE operator.

However, for data that is published/available in a RDF format but not necessarily set up through an endpoint, we make use of SPARQL-LD to directly access and exploit the RDF graphs. SPARQL-LD has an extended SERVICE definition that tries to fetch and query the RDF triples that may exist in the given resource at execution time.

SPARQL-LD does not cater to resources that do not have a RDF representation. Hence we make an extension to SPARQL-LD, enabling the use of non-RDF web documents published by constrained devices that host their own HTTP server. Our implementation uses the Content-Lifting-Rule HTTP response header field as defined in [10]. The value of this parameter is an absolute URI that identifies some RDF lifting mechanism (SPARQL-Generate, JSON-LD

Context, RML Mapping, etc.). Our extension of SPARQL-LD implements the support of lifting rules expressed as SPARQL-Generate queries [11]. This allow us to execute portions of a query to the RDF generated from lightweight documents in arbitrary formats exposed by constrained WoT devices.

Such a lifting rule could be hosted on the device manufacturer's website for example.

5 QUERY FEDERATION

The proposed solution is described and integrated within the 3 main phases of any query federation engine: Sources Selection, Federated Query Formation and Federated Query Execution as shown in Figure 1.

5.1 Sources Selection

It would not be efficient to send every piece of query to all the data sources, we need to determine the relevant data sources. In fact, it is crucial in constrained environments to preserve the longevity of the devices (battery life) and the bandwidth. For the solution we propose, we assume contextual information about the devices and what they expose is available in a *Data Catalogue*. Such a Data Catalogue could be constructed by the dataset publisher, the installer of the devices, or automatically thanks to automatic registration from the devices. We issue queries against this catalogue to identify which of the data sources are relevant for a particular part of the query. More precisely, we suppose that each data source is linked to some Basic Graph Pattern (i.e., a RDF Graph with variables) that describes the type of RDF graph that would be the result of lifting the document retrieved at any time. This would allow us to check whether a source has a partial solution to the high level query. It is worth noting here, that many other mechanisms exists[7, 15] to identify proper data sources other than a Data Catalogue and the proposed solution can be adjusted with respect to the meta-data source available.

5.2 Federated Query Formation

In the Federated Query Formation phase, we decompose the input query and build a new query with the union of multiple SERVICE clauses that are to be issued to the source endpoints selected in the previous phase: that means those (i) whose context is relevant (e.g., that are on the floor one want to query the temperature of), and (ii) capable of providing some relevant information. Each sub-query is built combining the biggest subset possible that is common to the Basic Graph Pattern the source exposes, and the Basic Graph Pattern in the input Query. Grouping several triples together like reduces the number of look-up to the same source and minimizes the intermediate join process.

5.3 Federated Query Execution

In the Federated Query Execution phase, the sub-queries built in the Federated Query Formation phase are executed upon the relevant sources as identified in the Sources Selection phase. This phase involves the use of the extended SERVICE clause as described in Section 4. If the endpoint/document to be queried against is not in RDF format, we launch a GET request to get the lifting rule and the document whose content are to be lifted or transformed. We then transform the data defined in the lifting rule and execute the sub-query against the resulting RDF graph. The obtained intermediate results are then federated and the final result is passed back to the user.

6 USAGE SCENARIO: EXAMPLE

In the Sources Selection phase, we learn from the data catalogue that not all triple patterns of the input query can be answered from a single data source. Hence, we issue a SELECT query to determine what triples can be answered by each of the data sources. This particular data source is added in the SERVICE clause of the sub-query. In the Federated Query Formation phase, we use the information drawn from the Sources Selection phase to put together the triples aimed for the same data sources and this creates a sub-query. However, the SERVICE clause might not necessarily be a SPARQL endpoint, such a case is the major focus of the proposed solution. If the solution is in a format not handled by SPARQL 1.1 or SPARQL-LD, we look for lifting rule information in the data catalogue or directly in the HTTP response header field Content-Lifting-Rule. This lifting rule is used during the Federated Query Execution phase, and the intermediate results are then federated to form the final result.

As a real-world use case, the end user can use the result to launch another command to change the temperature through the heater in Room 2 based on the occupancy sensor data.

7 EVALUATION AND IMPLEMENTATION

We ran 100 tests against various data sources. Figure 2 shows the run-time for the tests in increasing order. 50 tests were run against direct SPARQL endpoints and RDF sources and another 50 tests against data sources in arbitrary formats. The information for all data sources was provided through a Data Catalogue.

As expected, data sources with either SPARQL endpoint or RDF formats have less run-time in general as compared to the arbitrary data format sources. The results are highly affected by the network status and the number of calls made to the Data Catalogue and most importantly, the number of endpoints present and the time taken for lifting the data to RDF as well as the size of the data. The average run-time for querying against SPARQL endpoints, RDF sources is noted as 30.31 seconds and the average run-time against arbitrary formats is noted as 38.02 seconds.

The experiments were run on a computer with processor Intel Core i5 @2.5 GHz CPU, 4GB RAM and running macOS Sierra (64 bit). The implementation is done in Java 1.8.

The extension of SPARQL-LD to leverage the RDF Presentation protocol is implemented as a fork of Apache Jena v3.3, and is available on GitHub[4]. The motivating example is provided as a test-case. We aim to get better results on a more advanced system and using optimization techniques as mentioned in the future works.

8 CONCLUSION AND FUTURE WORK

The problem of exploiting data from heterogeneous sources and formats is common in the linked data world. In this paper, we have proposed a solution that draws benefits of SPARQL-LD and is able to directly fetch and query RDF data from any HTTP Web

[4] https://github.com/thesmartenergy/sparql-ld-extended

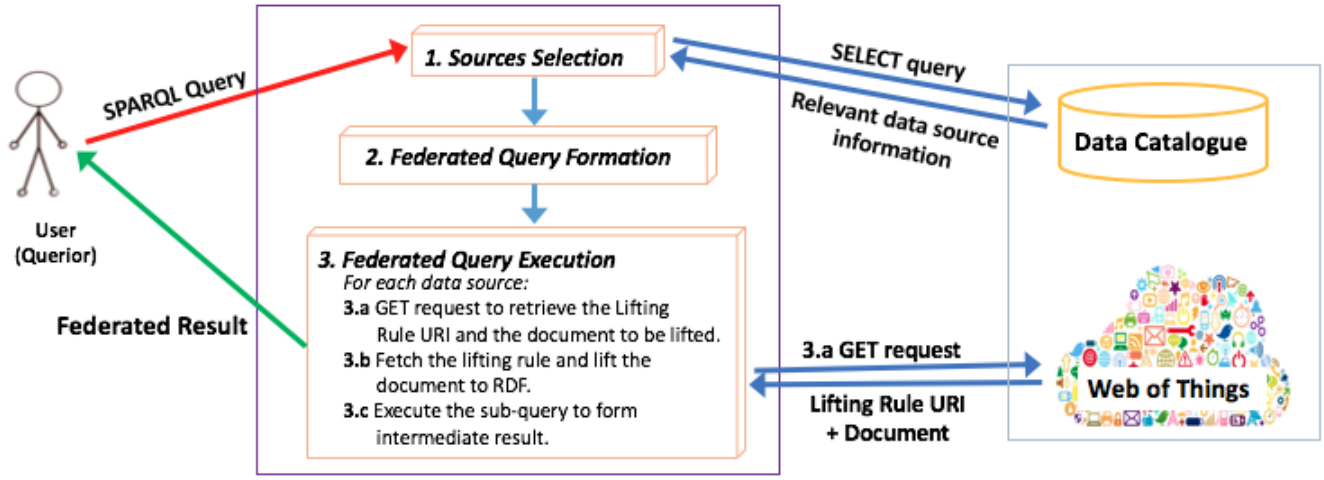

Figure 1: Proposed solution integrated in Query Federation Phases

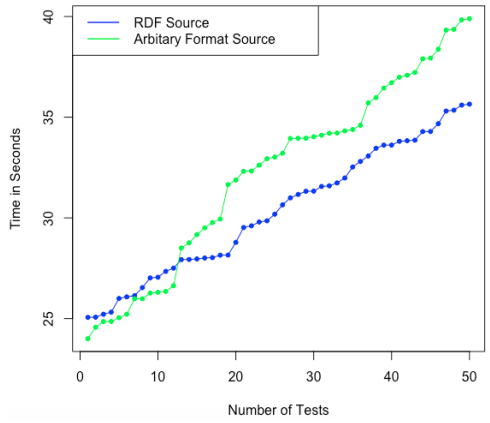

Figure 2: Run time against various Data Sources

document. The solution comprises an extension of SPARQL-LD that allows to also query data that is not directly presented in a RDF format, but for which a RDF lifting rule is known. We then fetch the URI of the RDF lifting rule along with the document, lift the data to RDF format and then querying the resultant RDF Graph. This solution allows us to exploit the dynamic nature of data sources such as light-weight sensors or devices. We have discussed extensively the working principles and phases of the proposed solution through the use of a motivating example, which is very close to a real world use-case scenario.

The proposed solution has the potential to spawn research directions towards a plethora of exciting new use cases and services as well as contribute towards the larger picture of flexible and scalable semantic interoperability for devices and services on the Web of Things or the Internet of Things at large, making these devices seemingly follow the principles of *Linked Data* [2].

The HTTP header field Content-Lifting-Rule we use could be worth being standardized at the W3C to also include non-native RDF serializations as Linked Data sources.

Future work planned for the solution includes implementing the federated query optimization techniques [7, 15] and to include more test-cases.

REFERENCES

[1] F. Alahmari. Evaluating SPARQL using query federation and link traversal. In *Digital Information Management (ICDIM), 2011 Sixth International Conference on*, pages 79–84. IEEE, 2011.
[2] T. Berners-Lee. Linked data. https://www.w3.org/DesignIssues/LinkedData.html, jul 2006. Last Accessed: 2017/05/02.
[3] 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.
[4] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. V. de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Proceedings of the Workshop on Linked Data on the Web, co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea*, 2014.
[5] P. Fafalios and Y. Tzitzikas. SPARQL-LD: a SPARQL Extension for Fetching and Querying Linked Data. In *International Semantic Web Conference (Posters & Demos)*, 2015.
[6] O. Görlitz and S. Staab. Federated data management and query optimization for linked open data. In *New Directions in Web Data Management 1*, pages 109–137. Springer, 2011.
[7] O. Görlitz and S. Staab. Splendid: SPARQL endpoint federation exploiting void descriptions. In *Proceedings of the Second International Conference on Consuming Linked Data-Volume 782*, pages 13–24. CEUR-WS. org, 2011.
[8] O. Hartig, C. Bizer, and J. C. Freytag. Executing SPARQL queries over the web of linked data. *The Semantic Web-ISWC 2009*, pages 293–309, 2009.
[9] G. Ladwig and T. Tran. Linked data query processing strategies. *The Semantic Web–ISWC 2010*, pages 453–469, 2010.
[10] M. Lefrançois. Interopérabilité sémantique libérale pour les services et les objets. In *Actes de la 17ème conférence Extraction et Gestion des Connaissances (EGC'17)*, Grenoble, France, Jan. 2017.
[11] M. Lefrançois, A. Zimmermann, and N. Bakerally. A SPARQL extension for generating RDF from heterogeneous formats. In *Proc. Extended Semantic Web Conference (ESWC'17)*, Portoroz, Slovenia, May 2017.
[12] M. Lefrançois, A. Zimmermann, and N. Bakerally. Flexible RDF generation from RDF and heterogeneous data sources with SPARQL-generate. In *Proc. of the 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW'16)*, 2016.
[13] A. Polleres, T. Krennwallner, N. Lopes, J. Kopecký, and S. Decker. XSPARQL Language Specification. W3C Member Submission, Jan. 20 2009.
[14] E. Prud'hommeaux, C. Buil-Aranda, et al. SPARQL 1.1 federated query. *W3C Recommendation*, 21, 2013.
[15] B. Quilitz. DARQ–Federated Queries with SPARQL, 2007.
[16] M. Sporny, G. Kellogg, and M. Lanthaler. A JSON-based Serialization for Linked Data. W3C Recommendation, Jan. 16 2014.