Query Templates for RDF Stream Processing

Robin Keskisärkkä

Linköping University, Linköping, Sweden
robin.keskisarkka@liu.se

Abstract. In recent years there has been a growing interest in using Semantic Web technologies to process streaming information, and several RDF Stream Processing (RSP) systems have been developed to bridge the gap between static and streaming Linked Data. However, the lack of a standardized query language makes testing and benchmarking of systems difficult, since queries need to be developed for each supported language in parallel. This process is both time consuming and error prone.
The RDF Stream Processing W3C Community Group[1] is currently working towards a standardized query language for RSP (RSP-QL). This paper proposes an extension to the SPIN Modeling Vocabulary to support the representation of RSP-QL queries as RDF. The vocabulary provides mechanisms to capture queries as parameterized templates that can be shared and queried using standard Semantic Web technologies.
The vocabulary is query language agnostic, and we demonstrate how different serializations (CQELS, SPARQLstream, and C-SPARQL) can be supported by modeling the queries of the CSRBench benchmark as a single set of RSP-QL queries. We also show some of the language features that are currently only supported in RSP-QL, and discuss some additional features that the SPIN extension might offer.
We argue that templates increase the flexibility and re-usability of queries, and by extension we believe that this can help in the more general adoption of RSP technologies.

Keywords: RSP-SPIN, RDF Stream Processing, RSP-QL, Complex Event Processing

1 Introduction

The amount of data made available as online streams is growing rapidly, and with it the need for handling streaming data efficiently. Applications requiring real-time processing are pushing the limits of traditional data processing infrastructures [12]. There are several parallel lines of development in this direction, ranging from the re-purposing of Database Management Systems (DBMS) into Data Stream Management Systems (DSMS)[1, 3], to technologies building on completely new paradigms for data stream processing. These systems depend on long-running continuous queries as opposed to one-time queries, and may

[1] https://www.w3.org/community/rsp/

be either data driven (where data is processed as it arrives), or rely on batch processing.

In recent years there has been a growing interest in using Semantic Web (SW) technologies for leveraging semantics in data streams. Traditional SW technologies are optimized for performance on more or less static data and do not scale well for continuous querying of volatile data. As a result several RDF Stream Processing (RSP) systems have been implemented, designed to handle the dynamic character of streaming data while still supporting static background information.

A challenge in the development of RSP technologies is that there are currently no standards for the syntax and semantics of RSP. Typically, RSP language extensions are based on SPARQL but take inspiration from SQL-based stream processing languages (e.g., Oracle CQL) and event processing languages, to enable querying of streaming data. This causes implementations, which superficially appear very similar, to be incompatible with respect to how queries are expressed and processed. However, the RSP Community Group[2] is working on defining a standardized RSP Query Language (RSP-QL), and the current drafts incorporate many of the features in existing RSP implementations. Notably, the draft adds the notion of streaming RDF graphs rather than RDF triples.

The effort required to develop and verify any type of non-trivial query against streaming data is often considerable. The fact that data is dynamically changing means that it is difficult to anticipate the correct results, especially if not working against recorded or generated streams. This problem becomes very challenging if a query has been designed to match very rare patterns, where a result may not be produced in hours, day, or even weeks. In the worst case scenario a query may never find a match, and we would be unable to validate our query on any actual results. Despite the effort that goes into writing and validating queries little or no attention has been directed at making RSP queries reusable and shareable. At present, anyone who wants to reuse a query will need to manually modify the query string, a process that in itself risks introducing errors or inconsistencies into the query.

In this paper we present an extension to the SPIN Modeling Vocabulary[3] to encapsulate RSP-QL queries as reusable parameterized query templates in RDF. The rest of the paper is organized as follows: Section 2 describes current state-of-the-art in RSP, the ongoing work in defining a standardized RSP Query Language (RSP-QL), and gives an overview of SPIN. Section 3 presents the RSP-SPIN extension of SPIN and discusses the extensions of the vocabulary required to support RSP-QL. In Section 4 we evaluate the extension by extending the SPIN API to support RSP-SPIN. We show how RSP-SPIN can be used to model the RSP-QL sample queries[4], and show that the template functionality of SPIN is compatible with the RSP-SPIN extension. We also demonstrate proof-of-concept serializers from RSP-SPIN into three RSP language extensions based

[2] https://www.w3.org/community/rsp/
[3] http://spinrdf.org/spin.html
[4] https://github.com/streamreasoning/RSP-QL

on the queries of CSRBench [6], and we discuss some of the trade-offs involved when handling unsupported query features. In Section 5 we discuss the results of the evaluation in the context of current RSP standardization efforts, and describe some implications with regard to the adoptation of RSP technologies outside the Semantic Web community. Finally, we summarize our findings in Section 7.

2 Background

There are a number of RSP implementations that each provide its own extension of SPARQL to support the processing of streaming data. EP-SPARQL [2], C-SPARQL [4], CQELS[9], INSTANS [11], and SPARQLstream [5] are examples of the most well-known extensions. Although these query language extensions may look very similar on the surface they are incompatible at the query level. This means that any user facing the task of choosing between one of the available RSP systems needs to make a commitment towards that particular system, since all queries would have to be formulated and verified against any other system. This divergence in RSP language extensions, both on the semantic and syntax level, has motivated the standardization efforts of the RSP Community Group[5], which is now in the process of defining a common RSP query language (RSP-QL) semantics and syntax for RSP.

Developing and validating the correctness of any non-trivial continuous query is time consuming. Where a static context allows errors to be detected by simply comparing the results of a query with the expected output, this is often not possible in a streaming context, where the underlying data is constantly changing. This becomes a problem when queries should detect low-frequency patterns, since an issue with a query can go undetected for long periods of time.

The time required to develop and validate, as well as maintain, queries in a streaming context is therefore substantial. Using prepared queries that can be modified on demand, similar to how stored procedures are used in relational databases, would be useful, but there is currently no way of defining such reusable structures for RSP queries. Prepared queries also have several other advantages, for example, they can be used to enforce data access control and to prevent query injections. Additionally, templatized queries do not require the user to have much experience in writing queries, which is often especially important if domain experts are to use the system.

Although a general template manager, such as Mustache[6] or FreeMarker[7], would be a viable option for defining flexible templates they fail to take advantage of the background data, such as ontologies, that is typically available in the context of RSP. This data can be used, for example, to support constraint checking, or to provide input suggestions for parameters.

SPIN was developed to support standard compliant SPARQL 1.1 queries, and enables information expressed as RDF do be used, for example, when defining

[5] https://www.w3.org/community/rsp/
[6] https://mustache.github.io/
[7] http://freemarker.org/

constraints for a parameter in a query template. In this paper we propose to extend SPIN to capture RSP-QL queries, to provide flexible query templates that can be reused and shared using standard SW technologies.

2.1 The RSP Query Language

The RSP Community Group is in the process of standardizing the RSP query language (RSP-QL). One important difference with regard to other RSP extensions is that RSP-QL assumes the streaming of annotated RDF graphs rather than triples. A model defining the semantics of RSP-QL has been proposed that makes some of the hidden assumptions of some RSP implementations explicit [8]. This model captures the query model and the operational semantics of existing RSP systems.

With regard to the query syntax of RSP-QL the current draft has been compared to other RSP languages [7]. This comparison is useful as a starting point, though the requirements document draft suggests that not all features are yet captured in the query language examples, for example, that of referring to named graphs in windows. Here the assumption appears to be that all graphs in a window are added to the window's default graph, while any information captured by the graph structure in the stream is lost. This would make it difficult to support Complex Event Processing (CEP) [10] if named graphs are to be used to represent and encapsulate event objects. Adding this feature to RSP-QL would not pose any restrictions with regard to any of the current query examples, and this paper will assume that this will be supported.

The new language extends SPARQL 1.1 and reuses many of the features found in CQELS [9], C-SPARQL [4], and SPARQLstream [5]. RSP-QL supports named windows over streams, where windows can be either count-based or time-based. It also supports the declaration of the output stream operator of queries, where `ISTREAM` outputs data not present in the previous window, `RSTREAM` outputs all data, and `DSTREAM` outputs data present in the previous window but not the new one.

The example query in Listing 1.1 illustrates some of the features of RSP-QL. The query counts the rides that exceed a given distance every hour by referencing a named graph inside a named window clause. In the available RSP-QL sample queries[8] no distinction is made between queries and streams in the `REGISTER AS` clause, and `STREAM` will therefore be assumed to indicate any continuous query.

[8] https://github.com/streamreasoning/RSP-QL/

```
PREFIX : <http://debs2015.org/streams/>
PREFIX debs: <http://debs2015.org/onto#>

REGISTER STREAM :longTrips AS

SELECT ISTREAM  (count(?ride) AS ?rideCount)
FROM NAMED WINDOW :w ON :trips [RANGE PT1H STEP PT1H]
WHERE
  { WINDOW :w
      { GRAPH ?g
          { ?ride debs:distance ?distance
            FILTER(?distance > 2)
          }
      }
  }
```

Listing 1.1. RSP-QL query counting the rides that exceed a given distance.

2.2 The SPIN Modeling Vocabulary

The SPARQL Inferencing Notation (SPIN)[9] provides mechanisms to capture reusable SPARQL queries in RDF. This can be used to construct rules and constraints implemented as SPARQL queries. Rules allow new information to be inferred from existing data, while constraints can be used to check, for example, violations in data. The RDF representation enables queries to be represented and shared using standard Semantic Web formats.

The vocabulary defines a lightweight collection of classes and properties that can be used to represent SPARQL queries as RDF. The model also allows parameterized templates to be defined on top of queries. The SPIN API provides methods for working with queries, constraints, rules, and templates. The API can also be used to validate parameter bindings for query templates.

There are two main ways of expressing queries in SPIN; as query strings, or as decomposed RDF. The former simplifies the maintenance of queries by making them human-readable, while the latter provides a syntax agnostic representation that allows multiple templates to be defined over the same query, supports the sharing of partial queries, and which can be queried using standard SPARQL.

3 Extending SPIN for RSP-QL

In this paper we propose the RSP-SPIN Vocabulary[10] as an extension of SPIN to support the features of RSP-QL. More specifically, the vocabulary extension

[9] http://spinrdf.org/spin.html
[10] http://w3id.org/rsp/spin

adds classes and properties for the definition of named windows over streams, window clauses, output stream operators, and register clauses.

RSP-QL provides a way of specifying the output stream operator for a query, that is, whether to produce the `ISTREAM`, `RSTREAM`, or `DSTREAM`. The `REGISTER AS` clause lets the name of the produced stream be specified as part of the query. Both features are illustrated in Listing 1.2.

```
@prefix :       <http://w3id.org/rsp/spin#> .
@prefix sp:     <http://spinrdf.org/sp#> .
@prefix stream: <http://debs2015.org/streams/> .

<#query>  a                sp:Select ;
          :registerAs      stream:longTrips ;
          :streamOperator  :Istream .
```

Listing 1.2. Representing `REGISTER AS` and the declaration of the output stream operator in RSP-SPIN.

RSP-QL supports three types of named windows over streams: logical windows (defined in terms of a time range), logical windows in the past (defined with an upper and lower time bound), and physical windows (defined in terms of number of streamed elements). Examples of each window type represented in RSP-SPIN are shown in Listing 1.3.

```
@prefix :       <http://w3id.org/rsp/spin#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix stream: <http://debs2015.org/streams/> .

<#query>
   :fromNamedWindow [ a              :LogicalWindow ;
                      :range         "PT1H"^^xsd:duration ;
                      :logicalStep   "PT1H"^^xsd:duration ;
                      :streamIri     stream:trips ;
                      :windowIri     :w1
                    ] ;
   :fromNamedWindow [ a              :LogicalPastWindow ;
                      :from          "PT2H"^^xsd:duration ;
                      :to            "PT1H"^^xsd:duration ;
                      :logicalStep   "PT1H"^^xsd:duration ;
                      :streamIri     stream:trips ;
                      :windowIri     :w2
                    ] ;
   :fromNamedWindow [ a              :PhysicalWindow ;
                      :size          1000 ;
                      :physicalStep  10 ;
                      :streamIri     stream:trips ;
                      :windowIri     :w3
                    ] .
```

Listing 1.3. RSP-QL window types represented using RSP-SPIN.

Referencing named windows is analogous to how named graphs are referenced in SPIN. Listing 1.4 shows an example where a named window is defined with a basic graph pattern.

```
@prefix :   <http://w3id.org/rsp/spin#> .
@prefix sp: <http://spinrdf.org/sp#> .

<#query>
  sp:where (
    [ a                :NamedWindow ;
      :windowNameNode  :w
      sp:elements (
        [ sp:subject    [ sp:varName  "ride"^^xsd:string ] ;
          sp:predicate  debs:distance ;
          sp:object     [ sp:varName  "distance"^^xsd:string ]
        ] ) ;
] ) .
```

Listing 1.4. Referencing a named window in RSP-SPIN.

The meta-modeling mechanisms and templating vocabulary of SPIN still apply in the extended model, and the reader is referred to the official SPIN documentation for details[11]. For reference, Listing 1.5 shows a basic template with a single parameter defined over a query. A full example has been excluded for brevity but is available along with the RSP-SPIN vocabulary[12].

```
@prefix spin: <http://spinrdf.org/spin#> .
@prefix spl:  <http://spinrdf.org/spl#> .
@prefix arg:  <http://spinrdf.org/arg#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

<#template>
  a                spin:SelectTemplate ;
  spin:body        <#query> ;
  spin:constraint  [ a              spl:Argument ;
                     spl:predicate  arg:distance ;
                     spl:valueType  xsd:integer ;
                     spl:optional   true
                   ] .
```

Listing 1.5. A SPIN template stating that the variable `distance` in the reference query is an optional integer parameter.

RSP-QL extends the output of `CONSTRUCT` queries to support named graphs. While this is not supported in the SPIN API it is a limitation of the SPARQL 1.1 syntax, rather than the SPIN vocabulary as such. The API extension provided with RSP-SPIN adds support for RSP-SPIN as described above, and includes a parser and serializer for RSP-QL.

[11] http://spinrdf.org/spin.html
[12] http://w3id.org/rsp/spin

4 Evaluation of RSP-SPIN

In this section we demonstrate and evaluate the functionality of RSP-SPIN and the API extension. The API has been made available as open source[13] and contains convenience classes for managing RSP-SPIN templates and queries, as well as proof-of-concept serializers for three RSP language extensions.

Query templates in SPIN allow constants to be lifted to variables that can be bound when the template is instantiated, and parameter constraints can be applied to any variable specified in the query. The RSP-QL parser provided with the RSP-SPIN API supports variables for the output stream name, input stream names, window steps and ranges, as well as any standard SPARQL 1.1 query variable. The API does not support parameterization of the output stream operator or named windows since no scenario could be identified where this would be required. The modeling capabilities of SPIN with respect to templates have not been effected by the RSP-SPIN extension. For this reason the evaluation will focus on query representation only, and for full examples of templatized RSP-QL queries the reader is referred to the example and test code of the API.

4.1 RSP-QL Queries

The sample queries of the RSP Community Group were used as a reference in evaluating RSP-SPIN and the API. Most of the queries contained minor syntax errors, such as missing prefix declarations, miss-spellings, and unbalanced parentheses. For the purpose of the evaluation these errors were corrected prior to parsing. One query contained more severe errors but since the intention the query was clear it could be corrected accordingly.

All sample queries except one were successfully parsed into RSP-SPIN and serialized back as an equivalent RSP-QL query (i.e., the original queries matched the queries serialized from the RSP-SPIN representation). The query that failed was not parseable as RSP-QL at all since it contained a nested `CONSTRUCT` query in a named window clause, something that has not been considered in the development of RSP-SPIN since nesting `CONSTRUCT` queries is not supported in SPARQL 1.1.

One additional query was tested to illustrate a feature not present in any of the current example queries, namely the use of named graphs in window clauses and in `CONSTRUCT` results (see Listing 1.6). This type of query is useful when filtering a stream with an arbitrary structure without modifying the elements in the stream. All the referenced queries and their RDF representations are available in the RSP-SPIN repository.

[13] ibid.

```
PREFIX  :      <http://debs2015.org/streams/>
PREFIX  debs:  <http://debs2015.org/onto#>

REGISTER STREAM :filteredTrips AS

CONSTRUCT ISTREAM {
   GRAPH ?g { ?s ?p ?o }
   ?g ?p2 ?o2 .
}
FROM NAMED WINDOW :w ON :trips [RANGE PT1S]
WHERE
  { WINDOW :w
    { GRAPH ?g
        { ?ride debs:distance ?distance .
          FILTER ( ?distance > 2)
          ?s ?p ?o .
        }
      ?g ?p2 ?o2 .
    }
  }
```

Listing 1.6. RSP-QL query filtering a stream of named graphs.

4.2 RSP-SPIN Serialization

RSP-QL has been shown to capture the semantics of CQELS, C-SPARQL, SPARQLstream [7]. The RDF representation of RSP-SPIN is query language agnostic and could therefore be used to capture any of these three RSP languages. This could potentially be very useful since it would be possible to define a query once in RSP-SPIN and from it provide serializations in any of the languages. This would also mean that the template support for RSP-QL would propagate to any of the three languages.

As a proof-of-concept we implemented serializers for each of the three query languages above. The strategy for handling unsupported features present in RSP-QL but not the target language was to resort to either exclusion or simplification. The definition of the stream operator is only supported in SPARQLstream, while `REGISTER AS` is only supported by C-SPARQL. In both cases the languages that lack support would need to exclude the feature. On the other hand, only CQELS supports multiple windows over the same stream, a case in which simplification for C-SPARQL and SPARQLstream would be required. Here we instead identify the outer bounds of a single window to include all named windows and use the smallest step size provided in the definitions. Finally, named graphs in streams and output results are supported by neither language and are collapsed into the default graph.

To evaluate the viability of this approach we used the CSRBench queries as the starting point, since the benchmark provides equivalent queries expressed

for each of the relevant query languages. The seven queries of the benchmark were expressed once using RSP-QL and parsed as RSP-SPIN. The serializations for each language were then compared with the manually constructed queries provided in the benchmark. The full list of queries and RDF representations are available in the RSP-SPIN repository[14]. For each of the serializations the query string returned was equivalent, but not identical, to the expected query.

We then attempted to serialize the RSP-QL sample queries into the three RSP languages. The strategies employed for unsupported features resulted in queries that were in some respects "rough" approximations of the original query, but for this initial proof-of-concept implementation we made no further inquiries into how these approximation could be improved further.

5 Discussion

The dynamic nature of data in a streaming context makes it difficult to verify the correctness of any type of non-trivial query. When a query is defined to match rare patterns considerable time may pass before a match is found, which complicates matters further. Additionally, queries in RSP applications are often long lived and run against data that is constantly in motion. After long periods of time a query is not unlikely to become inactive (e.g., if an input stream becomes unavailable), or invalid (e.g., a relevant ontology is updated). This means that many RSP queries also need to be maintained over time.

The API extension was used to parse the RSP-QL sample queries into RDF, and each query could be serialized back into the original query. Since the proposed vocabulary extension is fully compatible with standard SPIN this also enables templates and parameter constraints to be defined for any query that can be represented using RSP-QL.

We also provided an implementation of serializers from RSP-SPIN to CQELS, C-SPARQL, and SPARQLstream. The serializations were verified against the CSRBench queries, where each serialization was compared with the reference query. This show that RSP-SPIN could, for example, be useful in the development and maintenance of queries that are used for benchmarks, as well as for users who wish to retain the option of switching between RSP systems, or who wishes to compare engine performance without requiring a lot of time to be spent on rewriting queries.

The representation of RSP-QL queries as RDF adds some complexity to the management of queries. As an alternative SPIN provides a basic property for attaching pure string based queries. Both solutions have their merits and both are compatible with RSP-SPIN and templates. However, the RDF representation provides a way of describing RSP-QL queries in a syntax agnostic way, allows its structure to be queried using standard SW technologies, and supports sharing of partial query structures between queries and templates. Purely string-based approaches are also at the risk of query injections, and complicate validation of input parameters.

[14] ibid.

6 Conclusion

Despite considerable energy spent on defining RSP queries little or no effort has been devoted to making the queries reusable. Instead, RSP queries are shared only as query strings, which have to be manually modified to fit a new use case. However, users of RSP-based systems cannot be expected to be experts at formulating RSP queries. Predefined queries, by contrast, allow regular users to leverage these technologies with minimal intervention. By providing a way of defining parameterized queries the flexibility of each prepared query is increased manifold. This process is very similar to how some types of stored procedures in RDMS are used and defined.

In this paper we have introduced RSP-SPIN, an extension to the SPIN Modeling Vocabulary for supporting encapsulation of RSP-QL queries as parameterized query templates. We demonstrated how the extension could be used to represent the sample queries of the RSP Community Group, and that RSP-SPIN can support serialization into different RSP languages. We argue that support for this type of templates is valuable not only within the RSP community but also in the adoptation of RSP in other domains.

Acknowledgments This work was supported by the EU FP7 project Visual Analytics for Sense-making in Criminal Intelligence Analysis (VALCRI) under grant number FP7-SEC-2013-608142.

References

1. Abadi, D., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Erwin, C., Galvez, E., Hatoun, M., Hwang, J., Maskey, A., Rasin, A., Singer, A., Stonebraker, M., Tatbul, N., Xing, Y., Yan, R., Zdonik, S.: Aurora: A Data Stream Management System. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. p. 666 (2003)
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: A Unified Language for Event Processing and St ream Reasoning. In: Proceedings of the 20th International Conference on World Wide Web (2011)
3. Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., Widom, J.: STREAM: The Stanford Data Stream Management System. Technical Report 2004-20, Stanford InfoLab (2004)
4. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF streams with C-SPARQL. SIGMOD Record 39(1), 20–26 (2010)
5. Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling Ontology-based Access to Streaming Data Sources. In: Proceedings of the 9th International Semantic Web Conference on The Semantic Web – Volume Part I. pp. 96–111. ISWC'10, Springer-Verlag, Berlin, Heidelberg (2010)
6. Dell'Aglio, D., Calbimonte, J.P., Balduini, M., Corcho, O., Della Valle, E.: On Correctness in RDF Stream Processor Benchmarking. In: Proceedings of the 12th International Semantic Web Conference - Part II. pp. 326–342. ISWC '13, Springer-Verlag New York, Inc., New York, NY, USA (2013)

7. Dell'Aglio, D., Calbimonte, J.P., Valle, E.D., Corcho, O.: Towards a Unified Language for RDF Stream Query Processing. In: Gandon, F., Guéret, C., Villata, S., Breslin, J.G., Faron-Zucker, C., Zimmermann, A. (eds.) ESWC (Satellite Events). Lecture Notes in Computer Science, vol. 9341, pp. 353–363. Springer (2015)
8. Dell'Aglio, D., Della Valle, E., Calbimonte, J.P., Corcho, O.: RSP-QL Semantics: A Unifying Query Model to Explain Heterogeneity of RDF Stream Processing Systems. Int. J. Semant. Web Inf. Syst. 10(4), 17–44 (October 2014)
9. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In: Proceedings of the 10th International Conference on the Semantic Web. pp. 370–388 (2011)
10. Luckham, D., Schulte, R.: Event Processing Glossary Version 2.0 (2011), http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/
11. Rinne, M., Nuutila, E., Törmä, S.: INSTANS: High-Performance Event Processing with Standard RDF and SPARQL. In: Proceedings of the ISWC 2012 Posters and Demonstrations Track. Boston, US (2012)
12. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 Requirements of Real-time Stream Processing. SIGMOD Rec. 34(4), 42–47 (December 2005)