Challenges & Opportunities of RSP-QL Implementations

Riccardo Tommasini and Emanuele Della Valle

Politecnico di Milano, DEIB, Milan, Italy
{riccardo.tommasini,emanuele.dellavalle}@polimi.it

Abstract. The RSP-QL model explains and unifies RDF Stream Processing (RSP) approaches into a more general semantics. It was successfully applied to model C-SPARQL, CQELS-QL and SPARQL$_{stream}$ taking into account the operational semantics of the existing implementations. In this paper, we present Yasper 1.0, an RSP engine that implements RSP-QL semantics. Moreover, we present the challenges we found during our implementation experience and we discuss the research opportunities they imply. For each of these challenges, we also formulate some hypotheses that will drive our future work empirical studies on RSP.

1 Introduction

The Stream Reasoning (SR) community is reaching an agreement on RSP-QL .– the reference model proposed by Dell'Aglio et al [9] – as a unifying semantics of existing continuous SPARQL extensions and the operational semantics. Moreover, RSP-QL was successfully applied to explain the operational semantics of three popular RSP engines, i.e., C-SPARQL engine, CQELS, and Morph$_{stream}$ and the semantics of the continuous extensions of SPARQL, they implement, i.e., respectively C-SPARQL, CEQLS-QL, and SPARQL$_{stream}$.

These RSP engines had a crucial role in fostering the popularity of SR research. Their adoption also supported empirical comparative research and consequently more foundational investigations [10] (§ 2). Similarly, we believe that an RSP-QL engine that fully supports RSP-QL will push the model towards its limits, unveiling new challenges and opportunities.

In this paper, we introduce Yasper 1.0 (Yet Another RSP Engine)[1], i.e. an RSP engine that implements RSP-QL semantics. We present the challenges that we found during our implementation experience and we discuss research opportunities that these challenges unveiled. Moreover, for each challenge we formulate some hypotheses that we aim at empirically testing as future work.

The remainder of the paper is organized as follow. Section 2 presents the background information required to understand this work. Section 3 discusses Yasper architecture and implementation details. Section 4 presents the challenges and Section 5 discusses the research opportunities w.r.t. the presented challenges and Section 6 concludes the paper.

[1] https://github.com/streamreasoning/yasper.

2 Background

In this section, we present the background information required to understand the content of the paper.

We adopt a generic definition of **RDF Stream**, i.e. a sequence of pairs (O_i, t_i), where t_i is a non-decreasing timestamp and O_i is either a named RDF Graph or a timestamped RDF triple.

RSP-QL [9] is a unifying model for RSP dialects and engines. RDF Stream are pair of named RDF Graph as O_i and timestamps.

A time-based sliding window operator $\mathbb{W}$ is a Stream-to-Relation (S2R) operator [2] defined by the triple (α, β, t^0). $\mathbb{W}$ determines a series of windows of width (α) and sliding parameter (β) starting from t^0. A Time-Varying Graph (TVG) is a function that takes a time instant as input and outputs an Instantaneous RDF Graph. The application of $\mathbb{W}$ on a RDF Stream is a Time-Varying Graph that for any given time instant t at which $\mathbb{W}$ is defined coalesces the RDF Graphs in the current window[2] into an Instantaneous RDF Graph.

A streaming dataset SDS is a set composed by an optional default element A_0, n ($n \geq 0$) named Time-Varying Graphs and m ($m \geq 0$) named sliding window operators over k ($k \leq m$) streams.

RSP-QL evaluation semantics is defined as eval($SDS(G)$, SE, t), where SDS is a streaming dataset having G as active Time-Varying Graph, SE is an algebraic expression and t is a time instant. Accordingly, the extension to multi-window queries involves more than one active Time-Varying Graphs, i.e. eval($SDS(G_i$ ... $G_n)$, SE, t). The evaluation is computed over the instantaneous graphs $G_i(t)$, i.e., eval($SDS(G_i, t)$, SE). An RSP-QL query is continuously evaluated against a SDS by an RSP engine. The set ET of evaluation time instants is determined by the RSP engine according to a reporting policy that, in RSP-QL, consists of the composition of the following strategies: **CC** Content Change – the engine reports if the content of the current window changes –, **WC** Window Close – the engine reports if the current window closes –, **NC** Non-empty Content – the engine reports if the current window is not empty –, and **P** Periodic – the engine reports at regular intervals.

The output of the RSP-QL query evaluation is an instantaneous multiset of solution mappings for each evaluation time instant in "ET". Time-aware operators called Relation-to-Stream (R2S) are required to transform the instantaneous multiset of solution mappings into a Time-Varying multiset of solution mappings. RSP-QL comprises the following R2S operators: the **RStream**, which emits each solution mappings; the **IStream**, which emits the difference between the current solution mappings and previous ones, and; the **DStream**, which emits the difference between the previous solution mappings and the current ones.

C-SPARQL [6] is a continuous extension of SPARQL for registering queries over RDF Streams represented as timestamped RDF triples, i.e., O_i is an RDF

[2] The current window identified by $\mathbb{W}$ with the oldest closing time instant at t

Triple. Queries written C-SPARQL are executed by an RSP engine[3] that pipes a Data Stream Management System (DSMS) with a SPARQL engine. The C-SPARQL engine delegates the window execution to the DSMS and the remainder of the query to the SPARQL engine. C-SPARQL reporting policy is on window close and non-empty content (**WCNC**). C-SPARQL provides only the RStream operator.

SPARQL$_{stream}$ **[7]** is an extension of SPARQL to query virtual RDF streams where O_i is an RDF triple. Morph$_{stream}$ translates SPARQL$_{stream}$ queries into several DSMSs queries by means of mappings. The mapping language is S2O, and ad-hoc extension of R2O to include the time semantics (windows). Morph$_{stream}$ reporting policy is on window close and non-empty content (**WC.NC.**). Morph$_{stream}$ provides all the R2S operators.

CQELS-QL [12] extends SPARQL with continuous semantics adding special operators to deal with streams and define windows. RDF streams are represented by timestamped RDF triples as O_i. CQELS ports DSMS concepts into a SPARQL engine and, thus, it not only executes CQELS-QL queries but also applies optimizations by adapting the execution of the query to the stream velocity. CQELS reports results on content change (**CC**). CQELS provides only the IStream operator.

3 YASPER: Yet Another RDF Stream Processing Engine

In this section, we introduce Yasper's architecture and how it implements RSP-QL concepts. Figure 1 shows the following modules of Yasper 1.0 Streams, Windowing, SDS, Querying and Reasoning. Modules that exposes Yasper as a REST service [3] are also available but not discussed.

The Stream module, Figure 1 (a), contains the classes to represent a Stream. Yasper adopts a generic data stream model [5]. Each stream is identified by an URI and its content is a generic stream item $si{=}< t_i, t_e, O >$ where t_i is the time when s entered the system (*ingestion time*); t_e is the time when s occurred (*event time*) and O is a generic data element.

Thanks to this representation, Yasper can replicate the two RDF stream models[4] studied in RSP, i.e. unbounded sequence of timestamped RDF triples as in C-SPARQL vs. unbounded sequence of RDF graphs as in SLD [4].

The Windowing module, Figure 1 (b), is based on Esper[5], an open-source DSMS that relies on the Event Processing Language (EPL) and special objects called *listeners* that continuously receive the EPL queries outputs. We represent the RSP-QL Time-Based Sliding Window Operator (henceforth referred to as window operator) as an EPL statement on an RDF Stream plus a listener that continuously receives its results. Each window operator maintains a

[3] https://github.com/streamreasoning/CSPARQL-engine
[4] The current implementation uses Apache Jena 3.
[5] www.espertech.com

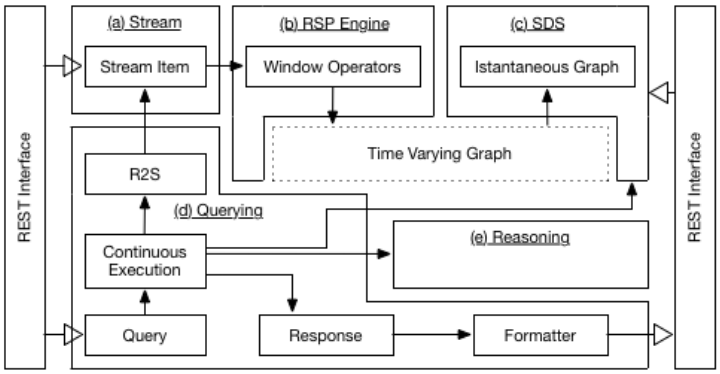

Fig. 1: Yasper's Modules: (a) Streams, (b) Windowing, (c) SDS, (d) Querying and (e) Reasoning. Only relevant architectural details are represented.

Time-Varying Graph that is responsible for generating an Instantaneous RDF Graph. The windowing is performed by means of temporal annotations of the StreamItems. Therefore Yasper can refer either to the ingestion–time – the time of arrival of a Stream Item – or to the event–time – the time each Stream Item is annotated with. Yasper works by default using Event Time that, notably, does not guarantee total ordering of Stream Items and, thus, requires further synchronization mechanisms in an open Web environment.

A streaming dataset SDS, Figure 1 (c), contains all the Time-Varying RDF Graphs. For each evaluation time t, each window operator w pushes StreamItems into a Time-Varying RDF Graph tvg, which reactively generates an Instantaneous RDF Graph g The streaming dataset SDS can be consolidated into an equivalent instantaneous dataset DI, i.e. a set of instantaneous RDF Graphs. The content of an instantaneous RDF graph g_i belongs to the current window identified by their window operator w_i at time t; w_i might be a sub-window [2]. Notably, slowly evolving RDF graphs are represented as a (named) Time-Varying Graph too. The SDS implementation extends Apache Jena 3 in-memory Dataset.

As described in Section 2, RSP-QL comprises four strategies to define an RSP engine reporting policy. As of the time of the submission, Yasper reports the results on Window Close (**WC**) and Non-empty Content (**NC**). However, how policies work with multi window queries is not obvious. RSP-QL [9] implies that any reporting policies could be mixed and introduces also the notion of sub-windows to support this claim. It is worth to notice that the set ET of evaluation time instants is tightly coupled with the arrival of new data, unless the RSP engine exploits an internal clock. In a real streaming setting, this is true even if the reporting policy define ET a-priori, e.g. periodic. For this reason, the creation of the SDS and the query evaluation are computed reactively to the arrival of a new Stream Item. This simple difference unveils several issues that we will discuss in the next section.

```
REGISTER STREAM <example> AS CONSTRUCT ISTREAM {?s ?p ?o}
FROM NAMED WINDOW :w1 [RANGE 5s, SLIDE 2s] ON STREAM :stream1
FROM NAMED WINDOW :w2 [RANGE 5s, SLIDE 5s] ON STREAM :stream2
WHERE { WINDOW ?w1 {?s ?p ?o}
        WINDOW ?w2 {?s ?p ?o}   FILTER (?w1 != ?w2) }
```

Listing 1.1: An example of RSP-QL Query.

The Querying module, Figure 1 (d), contains the elements for query instantiation and continuous execution. At this stage of development, Yasper accepts only SELECT and CONSTRUCT queries; the R2S operators R-, I- and DStream are available [2] and it also supports multi-window queries. Listing 1.1 presents the currently supported syntax, which is inspired to [8] and the W3C RSP CG[6].

The reasoning module, Figure 1 (e), supports continuous query answering under entailment regime. For each evaluation time instant t, before evaluating an RSP-QL query q, Yasper computes the closure under the entailment e of each instantaneous RDF Graph in SDS(t) using the Jena Generic Rule Reasoner. Future work in this direction comprise the integration of RDFox and Ontop.

4 Challenges

In this section, we present the challenges unveiled by our implementation experience. Yasper relies on some design decisions that require to be discussed with the SR/RSP community. In particular, we found issues about the streams item representation, stream item order, and multi-window queries evaluation, which relates to the notion of RSP-QL dataset SDS. We summarize these issues with the following questions:

(Q1) What is the best way to model the stream content?
(Q2) How does the time model impact the processing?
(Q3) What does define the *current* SDS at time t?
(Q4) Is there an efficient way to maintain the SDS?

(C1) Stream Content. RSP-QL assumes that stream items are timestamped (named) RDF Graphs. Actually, this assumption contrasts with how most of the existing RSP approaches work in practice. Indeed, [6, 7, 12] adopt timestamped RDF triples and only SLD [4] uses timestamped named graphs. Moreover, Balduini et al. [5] recently highlighted the benefits of a generic data model combined to a lazy-transformation approach.

Although the theoretical equivalence between the two approaches has been proved [9], we still have to identify their impact on performance. (Q1) highlights the need to identify an optimum if any. To support this investigation, we propose the following hypotheses:

- *Hp.0 Streams represented as RDF statement and those represented as Named graph are equal under the assumptions of non-decreasing timestamps for the RDF statement stream.*

[6] https://github.com/streamreasoning/rsp-ql

– *Hp.1 The cost of translating a stream of timestamped RDF statement to a stream of named RDF graph is negligible.*

(C2) Stream Ordering. RSP approaches typically assume the stream items arrive with non-decreasing timestamps. Although this assumption is reasonable for monolithic infrastructure, it does not hold for distributed or federated architectures [1]. (Q2) highlights the need to study the consequences of alternative time representations [1], especially in those settings where is not possible to guarantee synchronization nor absence of out-of-order arrivals. To support this investigation, we propose the following hypotheses:

– *Hp.2 Event–time and ingestion–time are equivalent for **partially** ordered streams.*
– *Hp.3 Event–time and ingestion–time are not equivalent for **totally** ordered streams.*

(C3) SDS Consolidation & Maintenance. RSP-QL assumes that an Instantaneous Graph IG can be generated by a Time-Varying Graph TVG for any instant t for which the window operator $\mathbb{W}$ is defined. This assumption implies that IG can be accessed even if the window is not closed. Although this approach theoretically promises always up-to-date results, it requires the RSP engine to maintain sub-windows accessible [1]. This surely has an impact on performance in terms of operations and memory. Moreover, if we consider existing RSP dialect syntaxes or RSP-QL proposals [8], the sub-window definition is transparent to the system user. This introduces ambiguity in the intended semantics. (Q3) highlights this issue from the point of view of the dataset SDS.

We name the problem *SDS Consolidation* and we introduce the notion of Active SDS – i.e. the SDS against which an RSP-QL query q is evaluated at time t. Dell'Aglio et al. [9] defined an SDS that relies on the notion of sub-windows. Since this consolidation semantics always considers the current status of all the windows, we refer to it as *Current* Active SDS. We identified two consolidations semantics that do not rely on sub-windows: (1.) the *Closed* Active SDS, as the name suggests, considers only the instantaneous RDF graphs that were derived from closed active windows. (2.) The *Cached* Active SDS considers the instantaneous RDF graphs that were derived from the latest closed active window. To support this investigation we formulate the following hypotheses:

– *Hp.4 The Current Active SDS is always more efficient in terms of resource usage than Closed and the Cached.*
– *Hp.5 The Current Active SDS is not more responsive than the Closed and the Cached ones.*

In order to better understand the impact on the results of the different SDS consolidations semantics, let us consider the example illustrated in Figure 2: :w1 and :w2 are two window operators applied to two streams. By definition, they define many windows with opening and closing time instants (o,c). Let us consider now the SDS at three relevant time instants: $t = 30$ where only a window defined by :w1 is closed; $t = 35$ where both :w1 and :w2 define two

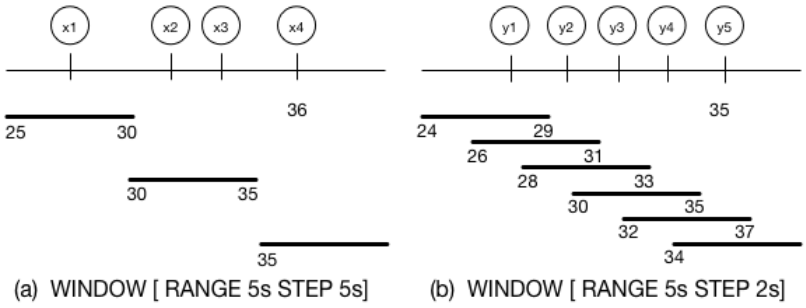

Fig. 2: Windowing Examples: (a) Window [RANGE 5s STEP 5s], (b) Window [RANGE 5s STEP 2s]. Cfr Listing 1.1.

closed windows, and; $t = 36$ where neither :w1 nor :w2 defines a closed windows Table 1 summarizes the results for each different SDS consolidations semantics and the reader can observe big differences for $t = 30$ and $t = 36$.

A second issues that arose from our implementation experience, summarized by (Q4), regards the problem of SDS maintenance. Once fixed the consolidation semantics, we need to identify an efficient[7] way to set up the Active SDS.

SDS(t)	Current		Closed		Cached	
	w1	w2	w1	w2	w1	w2
SDS(30)	x1	y1,y2	x1	$\emptyset$	x1	y1
SDS(35)	x2,x3	y3,y4	x2,x3	y3,y4	x2,x3	y3,y4
SDS(36)	x4	y5	$\emptyset$	$\emptyset$	x2,x3	y3,y4

Table 1: Differences between Active SDS respectively Current, Closed and Cached.

To this extent, we introduce the notion of *Maintenance Strategy*, i.e. the procedure to apply the SDS consolidation semantics. A maintenance strategy describes how a Time-Varying Graph generates and Instantaneous Graph. We identified two alternative maintenance strategies: (i) *Snapshot* – i.e. Time-Varying Graph the trashes the whole Instantaneous Graph content – and (ii) *Deltas* – i.e. the Time-Varying Graph updates the Instantaneous Graph considering the differences between the current content and the previous one in terms of additions and deletions. To support this investigation, we formulate the following hypotheses

- *Hp.6 Once fixed the SDS consolidation approach, the Maintenance Strategy does not influence the correctness of an RSP-QL query.*
- *Hp.7 the performance of the delta Maintenance Strategy is always comparable to those of the snapshot.*

[7] We consider efficient an approach that performs better than a naive solution that rebuild SDS.

	Challenges	RSP-QL	Variants	
C1	Stream Content	Named Graph	RDF Statement	Virtual
C2	Stream Ordering	Non-Decreasing	Monotonic	Late Items
C3	SDS Consolidation	Current	Closed	Cached
	SDS Maintenance	/	Snapshot	Deltas

Table 2: Five challenges in the RSP-QL model and variants to them.

5 Discussion

In this section we discuss the research opportunities that each of the presented challenges highlights.

Challenge C1, and partially C3, indicate that the RSP solution space is not yet systematically explored and there is room for optimization driven by RSP-QL insights. The community efforts on empirical research should continue, also supported by new resources like Yasper and the recent RSPLab [14]. Moreover, we need to define uniform evaluation methodologies based on choke-points [11], that guide the evaluation of new approaches.

Challenge C2 suggests that some typical SR/RSP assumptions are inadequate for distributed or federated environments. The management of out-of-order data arrival is an open challenge for mature systems like Spark[8] and Flink[9] and so should be for the SR/RSP community. Federated RSP is another appealing idea; networks of engines that exchange data and queries require to relax these assumptions and rethink the model.

We advocate the use of reasoning to deal with the challenges, i.e. out-of-order data arrival and federation and planning. Moreover, correctness must be redefined to include relevant notions such as maximum delay, watermark and multiple active windows [1].

Challenge C3 reveals that designing a syntax that takes into account all the aspects of RSP is hard. Especially because we, as a community, did not investigate what is the intended semantics of a query. A universal syntax might not exist, while task-specific syntaxes that capture fragments of RSP-QL semantics can better fit the user needs.

Specifically for querying, we must focus on usability. For a query language, we need to reduce ambiguity and improve efficiency as much as possible even if it means trading expressiveness and usability if necessary.

Our intuition is that some window operators are easier to use in combination with specific policies and our proposal is to reflect these preferred relations in the RSP-QL syntax. In Listing 1.2 we present three alternative syntaxes that we believe are tightly coupled. The window at line 1 suggests that the reporting will happen on window close (i.e. when the window slides). The window at line 2 is meaningful only when the system reports on content change. The window

[8] https://spark.apache.org/
[9] https://flink.apache.org/

at line 3 might be a possible syntax that highlights that sub-windows of a given size are allowed.

```
1 FROM NAMED WINDOW :a [RANGE 5s, STEP 2s] ON STREAM :str1
2 FROM NAMED WINDOW :b [RANGE 5s] ON STREAM :str2
3 FROM NAMED WINDOW :c [RANGE 5s, SLIDE 5s, SPLIT 1s] ON STREAM :str3
```

Listing 1.2: An example of RSP-QL Query.

6 Conclusion

In this paper we presented Yasper 1.0, i.e. an RSP-QL compliant RSP engine, and the challenges we faced during our implementation experience.

At the moment of writing, Yasper supports the proposed syntax [8] and can answer multi-stream SELECT and CONSTRUCT queries. The support of ASK queries is a work in progress whereas the DESCRIBE clause requires further investigation. Yasper supports all the RSP-QL streaming operators while it's reporting policy is On Window Close and Non-Empty Content. We are working to make the reporting policy also configurable in order to support all the combinations.

Regarding the presented challenges and our hypotheses, our goals is to perform a descriptive study on RSP engines. To this extent, we will formulate statistical tests to investigate our hypotheses and we will extensively evaluate Yasper 1.0 and the other relevant RSP engines [13, 9].

Acknowledgments. We thank Daniele Dell'Aglio for the supporting discussions and brainstorming regarding RSP-QL.

References

1. Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., Whittle, S.: The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. PVLDB 8(12), 1792–1803 (2015), http://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf
2. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB J. 15(2), 121–142 (2006), https://doi.org/10.1007/s00778-004-0147-z
3. Balduini, M., Della Valle, E.: A restful interface for RDF stream processors. In: Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013. pp. 209–212 (2013), http://ceur-ws.org/Vol-1035/iswc2013_poster_8.pdf
4. Balduini, M., Della Valle, E., Dell'Aglio, D., Tsytsarau, M., Palpanas, T., Confalonieri, C.: Social listening of city scale events using the streaming linked data framework. In: International Semantic Web Conference (2). Lecture Notes in Computer Science, vol. 8219, pp. 1–16. Springer (2013)

5. Balduini, M., Della Valle, E., Tommasini, R.: SLD Revolution: A Cheaper, Faster yet more Accurate Streaming Linked Data Framework. pp. 1–15. http://ceur-ws.org/Vol-1870/#paper-01
6. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: a continuous query language for RDF data streams. Int. J. Semantic Computing 4(1), 3–25 (2010), https://doi.org/10.1142/S1793351X10000936
7. Calbimonte, J., Corcho, Ó., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I. pp. 96–111 (2010), https://doi.org/10.1007/978-3-642-17746-0_7
8. Dell'Aglio, D., Calbimonte, J., Della Valle, E., Corcho, Ó.: Towards a unified language for RDF stream query processing. In: ESWC 2015 Satellite Events Portorož, Slovenia, May 31 - June 4, 2015, Revised Selected Papers. pp. 353–363 (2015)
9. Dell'Aglio, D., Della Valle, E., Calbimonte, J., Corcho, Ó.: RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. Int. J. Semantic Web Inf. Syst. 10(4), 17–44 (2014)
10. DellAglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science (Preprint), 1–24
11. Kotsev, V., Minadakis, N., Papakonstantinou, V., Erling, O., Fundulaki, I., Kiryakov, A.: Benchmarking RDF query engines: The LDBC semantic publishing benchmark. In: Proceedings of the Workshop on Benchmarking Linked Data (BLINK 2016) co-located with the 15th International Semantic Web Conference (ISWC), Kobe, Japan, October 18, 2016. (2016), http://ceur-ws.org/Vol-1700/paper-01.pdf
12. Phuoc, D.L., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I. pp. 370–388 (2011), https://doi.org/10.1007/978-3-642-25073-6_24
13. Tommasini, R., Della Valle, E., Balduini, M., Dell'Aglio, D.: Heaven: A framework for systematic comparative research approach for RSP engines. In: The 13th International ESWC, Heraklion, Crete, Greece, 2016, Proceedings. pp. 250–265 (2016)
14. Tommasini, R., Valle, E.D., Mauri, A., Brambilla, M.: Rsplab: Rdf stream processing benchmarking made easy. In: The Semantic Web 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II