RSP-QL*: Enabling Statement-Level Annotations in RDF Streams

Robin Keskisärkkä[1], Eva Blomqvist[1,2], Leili Lind[1,2], and Olaf Hartig[1]

[1] Linköping University, Linköping, Sweden
firstname.lastname@liu.se
[2] RISE Research Institutes of Sweden AB / Division ICT SICS East, Linköping, Sweden
firstname.lastname@ri.se

Abstract. RSP-QL was developed by the W3C RDF Stream Processing (RSP) community group as a common way to express and query RDF streams. However, RSP-QL does not provide any way of annotating data on the statement level, for example, to express the uncertainty that is often associated with streaming information. Instead, the only way to provide such information has been to use RDF reification, which adds additional complexity to query processing, and is syntactically verbose. In this paper, we define an extension of RSP-QL, called RSP-QL*, that provides an intuitive way for supporting statement-level annotations in RSP. The approach leverages the concepts previously described for RDF* and SPARQL*. We illustrate the proposed approach based on a scenario from a research project in e-health. An open-source implementation of the proposal is provided and compared to the baseline approach of using RDF reification. The results show that this way of dealing with statement-level annotations offers advantages with respect to both data transfer bandwidth and query execution performance.

Keywords: RSP-QL* · RDF* · RDF Stream Processing · e-health

1 Introduction

Recent years have seen an increasing interest in processing and analyzing streaming information as it is generated by applications, services, sensors, and smart devices. RDF Stream Processing (RSP) leverages the principles of Linked Data and the Semantic Web to cope with heterogeneity in data, but employs strategies inspired from stream processing to cope with high velocity data streams. During the last decade, several RSP systems and models have been proposed, which have all provided their own syntax, semantics, and underlying assumptions about the nature of RDF streams [6,7]. The RSP community group[3] was formed to define a common model for producing, transmitting and continuously querying RDF streams. The first version of this common query model (RSP-QL)

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

was proposed by Dell'Aglio et al. in 2014 [7], and the draft of the abstract syntax and semantics was published by the RSP community group in 2016 [2].

Data generated by sensors is almost always coupled with provenance information, or a level of uncertainty representing, for instance, lack of precision or a knowledge gap. For example, all values reported by a temperature sensor may be associated with some error describing a probability distribution. The RDF specification provides a vocabulary that allows metadata to be represented about RDF triples using *RDF reification* [11]. In practice, however, this is not widely adopted as a standard for representing and managing such metadata on the Semantic Web [8]. RDF* was recently proposed as a way to support a concise representation of statement-level metadata, while remaining backwards compatible with standard RDF [9,10]. By enclosing a triple using the strings '<<' and '>>', the extension allows it to be used in the subject or object position of other triples. This allows statement-level metadata to be provided directly. For example, the triple *:bob :knows :alice* could be annotated with the source *wikipedia* as follows: <<*:bob :knows :alice*>> *:source :wikipedia*. Similarly, the authors' propose SPARQL* as an extension of SPARQL for querying RDF* data, where SPARQL* supports similar nesting of triple patterns.

We propose an extension to RSP-QL that leverages RDF*/SPARQL* for annotating and querying streaming data. We show that the proposed approach has several benefits over RDF reification when it comes to statement-level annotations. The approach is motivated based on a use case from a current research project, where we attempt to detect abnormal situations in an e-health scenario.

The rest of the paper is organized as follows. Section 2 briefly discusses the relevant related work, while Section 3 describes a use-case scenario that both motivates the proposed approach and exemplifies the requirements addressed by the proposal. Section 4 describes the proposed approach informally, and Sections 5–6 provide the necessary formal definitions, where Section 5 defines the data model and Section 6 defines the syntax and semantics of the proposed RSP-QL extension. Section 7 provides an application-based evaluation of the approach. Section 8 describes a prototype implementation and a performance evaluation of the implemented system. Section 9 discusses the impact of the presented work and Section 10 summarizes the main conclusions of the paper.

2 Related Work

Over the past decade, there has been a growing interest in providing models and languages for combining the principles of the Semantic Web with streaming information. RDF Stream Processing (RSP) systems aim to provide extensions to RDF and SPARQL for representing and querying streaming data. However, though several RSP systems have emerged that provide extensions and operators for this purpose [1,3,4,13,18], they typically provide different languages, constructs, operators, and evaluation semantics [7]. The W3C RSP community group was formed to define a common model for representing and querying streaming RDF data. The proposed model and language, RSP-QL [7], can be

used to model the behavior of most of the current RSP systems, and provides well-defined semantics for explaining query execution. However, none of the existing RSP approaches have given much attention to aspects related to representing metadata in streams, such as uncertainty or provenance. The RSP-QL stream model allows such annotations to be provided on the graph level, but annotations on the triple level are not supported.

The term *statement-level metadata* refers to data that captures information about a single statement or fact. The RDF specification includes the notion of *RDF reification* that lets a set of RDF triples describe some other RDF triple [11]. The approach requires the inclusion of four additional RDF triples for every statement where metadata is to be provided. Another approach is to leverage *named graphs*, where the identifier of the graphs can be used to attach metadata to statements [12]. However, this approach has the disadvantage of inhibiting the application of named graphs for other uses. Finally, *singleton properties* have been proposed as an alternative approach, where a distinct property is provided for each triple to be annotated [15]. The singleton properties proposal introduces a large number of unique predicates, which is atypical for RDF data, and disadvantageous for common SPARQL optimization techniques [19]. Additionally, these approaches result in verbose queries [9]. For standard RDF, there therefore exists no convenient way of annotating data with metadata on the statement level [10]. The RDF⋆/SPARQL⋆ approach was proposed as a way of supporting a more intuitive representation, by allowing triples in the subject and object positions of RDF statements [9,10]. In this paper, we propose to extend RSP-QL based on this approach.

3 Use-Case Scenario

In this section, we describe a use-case scenario to exemplify the kinds of requirements that may be addressed by combining RSP-QL with RDF⋆/SPARQL⋆. The scenario originates from an ongoing research project, E-care@home[4], in which the aim is to develop privacy-preserving AI-solutions for home care of elderly patients. In addition to developing technical solutions, the project has put great emphasis on studying the requirements of stakeholders. These requirements have been documented in a project deliverable [14]. As part of this deliverable, a number of personas and use-case scenarios were also developed, including the following description of a scenario involving the patient Rut who has advanced chronic obstructive pulmonary disease (COPD) and is multimorbid.

"The system can automatically sense abnormal situations, e.g. when certain health parameters deviate from the normal values, or when the overall situation as assessed by a multitude of sensors appears abnormal. When the system detects such situations, it sends out an alarm to a suitable recipient based on the severity of the deviation (e.g., emergency dispatch for a life-threatening deviation, the patient's physician if no immediate action is required, or next-of-kin if suitable). [...] Today the system has detected an abnormal state. Rut appears to have been

[4] http://ecareathome.se/

sitting in the same position in a chair in the living room for an unusually long time given that there are no entertainment devices turned on at the moment. Her heart rate is above normal, but her breathing is slower than normal. Small motions indicate that she is not asleep, yet she is not moving much. Her oxygen levels are about normal. The system decides to classify this as a low-emergency abnormal state. The system also knows that Rut's partner has left the house a few hours ago. It therefore sends an alert to him [...] the alert reaches Rut's partner as he is already on his way home. He hurries home and opens the door only to find out that Rut is in good health and has been enjoying a paperback copy of the latest crime novel by a famous Swedish author for the past few hours." [14]

Like any health-care system, the one envisioned by E-care@home sets high requirements in terms of patient safety, system reliability, and transparency. To this end, all the data that the system uses to draw conclusions and to generate suggestions, or even to take action, must be accompanied by some assessed confidence. For instance, in the scenario above, to put patient safety first the system cannot afford to miss an abnormal and highly dangerous situation, but on the other hand it needs to be able to disregard observations that are not reliable. As an example, whenever a pulse oxymeter reports the oxygen saturation of a patient, the system also needs to know the confidence that the system can put in this value. The sensor may have a fixed confidence value, but the system may also derive an adjusted value that takes into account contextual factors of the measurement, such as the position of the sensor and the activity of the patient at measurement time. Regardless of how the confidence value is derived, it needs to be reported as part of the reported observation.

4 Overview of RSP-QL⋆

The main difference between RSP and traditional RDF/SPARQL processing is that the former introduces a time dimension to processing [6]. The time dimension in RSP-QL is managed by allowing *windows* to define discrete subsets over *RDF streams*, and at any point in time, a window can be queried as a regular RDF dataset. The approach proposed in this paper extends RSP-QL in two fundamental ways: RDF streams are extended to support RDF⋆, and the supported graph patterns in RSP-QL are extended to support those in SPARQL⋆. The example in Listing 1.1 shows an RSP-QL⋆ query that illustrates the main features and language constructs.

The registered query is evaluated every 10 seconds. It defines a time-based window with a width of 1 minute that slides every 10 seconds over the heart-rate stream. The query then matches the heart-rate value and confidence of each observation in the window using an RDF⋆ pattern [9]. This is the only difference between RSP-QL and RSP-QL⋆ in this query. The results are then filtered based on a threshold, and the heart-rate value and timestamp of the matched observations are reported. There are conceptually no limitations on the complexity of the provided annotations, and they can, e.g., instead be represented as confidence intervals or distributions rather than single values.

```
PREFIX ex: <http://www.example.org/ontology#>
PREFIX sosa: <http://www.w3.org/ns/sosa/>
REGISTER STREAM <heart-rate/alert> COMPUTED EVERY PT10S AS
SELECT ?hr ?time
FROM NAMED WINDOW <window/1> ON <http://stream/heart-rate> [RANGE PT1M STEP PT10S]
WHERE {
    WINDOW <window/1> {
        GRAPH ?g {
            <<?obs sosa:hasSimpleResult ?hr>> ex:Confidence ?confidence .
            FILTER(?confidence > 0.9 && ?hr > 120)
        }
        ?g <generatedAt> ?time .
    }
}
```

Listing 1.1: Example of an RSP-QL* query.

5 Data Model

This section defines the concepts that capture the notion of streams considered by our approach. We begin with the basic notions of RDF and RDF*.

As usual [5,16], we assume three pairwise disjoint, countably infinite sets $\mathcal{I}$ (IRIs), $\mathcal{B}$ (blank nodes), and $\mathcal{L}$ (literals). Then, an *RDF triple* is a tuple $(s, p, o) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, and an *RDF graph* is a set of RDF triples. For such a triple (s, p, o), s is called the *subject*, p the *predicate*, and o the *object*.

RDF* extends this notion of triples by allowing the subject or the object to be another triple [9]. This form of nesting of triples, which may be arbitrarily deep, allows for statements to capture metadata about other statements. Formally, an *RDF* triple* is defined recursively as follows [9]: (i) any RDF triple is an RDF* triple, and (ii) given two RDF* triples t and t', and the RDF terms $s \in (\mathcal{I} \cup \mathcal{B})$, $p \in \mathcal{I}$, and $o \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, the tuples (t, p, o), (s, p, t), and (t, p, t') are RDF* triples. Furthermore, a set of RDF* triples is called an *RDF* graph*.

The concept of an *RDF dataset* has been introduced to represent collections of RDF graphs [5]. We extend this concept to cover RDF* graphs.

Definition 1. *A **named RDF* graph** is a pair $(n, G^\star)$ where $n \in (\mathcal{I} \cup \mathcal{B})$, which is called the* graph name, *and $G^\star$ is an RDF* graph. An **RDF* dataset** is a set $D = \{G_0^\star, (n_1, G_1^\star), (n_2, G_2^\star), ..., (n_i, G_i^\star)\}$, where $G_0^\star$ is an RDF* graph, called the* default graph *of D, and $(n_k, G_k^\star)$ is a named RDF* graph for all $k \in \{1, 2, ..., i\}$.*

While the RDF model is atemporal, the notion of an RDF stream has been introduced to capture the dynamic nature of streaming RDF data [7]. Along the same lines, we define an *RDF* stream* as a time-ordered sequence of elements that are captured by a specific form of RDF* datasets.

Definition 2. *Let p be an IRI that denotes a predicate to capture timestamps for named RDF* graphs. Then, an **RDF* stream element** E is an RDF* dataset that consists of a default graph $G_o^\star$ and exactly one named RDF* graph $(n, G^\star)$ such that the default graph $G_o^\star$ contains one RDF triple of the form (n, p, τ), where τ is a timestamp. To denote this timestamp τ in E we write $\tau(E)$.*

Definition 3. *An **RDF⋆ stream** S is a potentially unbounded sequence of RDF⋆ stream elements such that for every pair of such elements E_i and E_j, where E_i comes before E_j (i.e., $S = (..., E_i, ..., E_j, ...)$), the following properties hold:*

1. *$\tau(E_i) \leq \tau(E_j)$, and*
2. *the names of the single named RDF⋆ graph $(n_i, G^\star_i)$ in E_i and of the single named RDF⋆ graph $(n_j, G^\star_j)$ in E_j are different (i.e., $n_i \neq n_j$).*

*A **named RDF⋆ stream** is a pair (n, S) where $n \in \mathcal{I}$ and S is an RDF⋆ stream.*

We also need to define a notion of windows over such streams as a way of referencing discrete portions of potentially infinite data streams [7].

Definition 4. *A **window** W over an RDF⋆ stream S is a finite set of RDF⋆ stream elements from S.*

In this paper, we focus explicitly on temporal window operators (other window operators, such as count-based windows, can be defined in a similar manner). To this end, we define a *time-based window* of an RDF⋆ stream as a contiguous set of elements from the stream whose timestamp is in a given interval.

Definition 5. *Given a time interval $[l, u)$, the **time-based window** over an RDF⋆ stream S for $[l, u)$, denoted by $\mathcal{W}(S, l, u)$, is a window over S that is defined as follows: $\mathcal{W}(S, l, u) = \{E \mid E \text{ is in } S \text{ and } l \leq \tau(E) < u\}$.*

Finally, we shall need a function that represents any window as an RDF⋆ dataset. Informally, this dataset consists of all the named RDF⋆ graphs of all RDF⋆ stream elements within the window, and the default graph of this dataset is constructed from the default graphs in all these RDF⋆ stream elements.

Definition 6. *Let $\mathcal{W} = \{E_1, E_2, ..., E_n\}$ be a window over some RDF⋆ stream. The **dataset representation** of $\mathcal{W}$, denoted by $DS(\mathcal{W})$, is the RDF⋆ dataset that is constructed as follows:*

- *the default graph of $DS(\mathcal{W})$ is $G^\star_0 = \bigcup_{\{G^\star_{dflt}, (n, G^\star)\} \in \mathcal{W}} G^\star_{dflt}$, and*
- *the set of named RDF⋆ graphs in $DS(\mathcal{W})$ is $\{(n, G^\star) \mid \{G^\star_{dflt}, (n, G^\star)\} \in \mathcal{W}\}$.*

6 Syntax and Semantics of RSP-QL⋆

This section defines RSP-QL⋆, which is an RDF⋆-aware extension of RSP-QL. RSP-QL, in turn, is an extension of SPARQL. Hence, our definitions in this section extend RSP-QL [7] along the lines of how SPARQL⋆ extends SPARQL [9,10], and by also taking into account the abstract syntax and semantics draft of the W3C RSP community group [2]. For the SPARQL-specific constructs we adopt the algebraic SPARQL syntax introduced by Pérez et al. [16]. Due to space constraints, we limit ourselves to presenting only the core concepts of the language.

6.1 Syntax of RSP-QL* Queries

RSP-QL is an extension of SPARQL [17], and the basic building block is a *basic graph pattern* (BGP), that is, a finite set of *triple patterns*. A triple pattern is a tuple $(s, p, o) \in (\mathcal{V} \cup \mathcal{B} \cup \mathcal{I}) \times (\mathcal{V} \cup \mathcal{I}) \times (\mathcal{V} \cup \mathcal{B} \cup \mathcal{I} \cup \mathcal{L})$, where $\mathcal{V}$ is a countably infinite set of query variables that is disjoint from $\mathcal{B}$, $\mathcal{I}$, and $\mathcal{L}$, respectively.

Like SPARQL* [9,10], RSP-QL* extends these notions further by supporting the concept of *triple* patterns*, which add the possibility to nest triple patterns (arbitrarily deep), and which are defined recursively as follows [9,10]:

- any triple pattern is a triple* pattern, and
- given two triple* patterns tp and tp', and $s \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{V})$, $p \in (\mathcal{I} \cup \mathcal{V})$, and $o \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$, then (tp, p, o), (s, p, tp), and (tp, p, tp') are triple* patterns.

A finite set of triple* patterns is referred to as a *BGP**.

On top of BGPs, RSP-QL supports all the other forms of graph patterns that have been introduced for SPARQL, and RSP-QL adds a new form to match data within windows of streaming data. We define a corresponding notion of patterns for RSP-QL*, but for brevity we here focus only on the core constructs.

Definition 7. *An* ***RSP-QL* pattern*** *is defined recursively as follows:*

1. *Any BGP* is an RSP-QL* pattern.*
2. *If $n \in (\mathcal{V} \cup \mathcal{I})$ and P is a RSP-QL* pattern, then (WINDOW n P) and (GRAPH n P) are RSP-QL* patterns.*
3. *If P_1 and P_2 are RSP-QL* patterns, then (P_1 AND P_2), (P_1 OPT P_2), and (P_1 UNION P_2) are RSP-QL* patterns.*

In addition to such patterns, every RSP-QL* query may declare windows over named RDF* streams, which we capture by the concept of window declarations.

Definition 8. *A* ***window declaration*** *is a tuple $(u_S, \alpha, \beta, \tau_0)$ where $u_S \in \mathcal{I}$ is an IRI (representing the name of a named RDF* stream), α is a time duration (representing a window width), β is a time duration (representing a slide parameter), and τ_0 is a timestamp (representing a start time).*

We now have everything required to define RSP-QL* queries, which consist of an RSP-QL* pattern and window declarations that are associated with IRIs to serve as names for the corresponding windows in the query.

Definition 9. *An* ***RSP-QL* query*** *is a pair (ω, P) where ω is a partial function that maps some IRIs in $\mathcal{I}$ to a window declaration, respectively, and P is an RSP-QL* pattern such that for every sub-pattern (WINDOW n P') in P it holds that if $n \in \mathcal{I}$, then ω is defined for n, i.e., $n \in \mathrm{dom}(\omega)$.*

6.2 Semantics of RSP-QL* Queries

We now define the semantics of RSP-QL* queries, for which we have to introduce some concepts used to define the query semantics of SPARQL and of SPARQL*.

The query semantics of SPARQL is based on the notion of *solution mappings* [16] that map query variables to blank nodes, IRIs, or literals. For SPARQL*, this notion has been extended to also be able to map to RDF* triples. That is, a *solution** *mapping* is a partial function $\eta : \mathcal{V} \to (\mathcal{T} \cup \mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ where $\mathcal{T}$ denotes the set of all RDF* triples [9,10]. The standard notions of *compatibility*, *merging* and *application* of solution mappings can then be adapted as follows.

Definition 10. *Two solution** *mappings* η, η' *are* ***compatible*** *if* $\eta(v) = \eta'(v)$ *for every variable* $v \in dom(\eta) \cap dom(\eta')$*.*

Definition 11. *The* ***merge*** *of two compatible solution** *mappings* η *and* η'*, denoted by* $\eta \cup \eta'$*, is a solution** *mapping* η'' *with the following three properties:*

- $dom(\eta'') = dom(\eta) \cup dom(\eta')$,
- $\eta''(v) = \eta(v)$ *for all* $v \in dom(\eta)$*, and*
- $\eta''(v) = \eta'(v)$ *for all* $v \in dom(\eta)' \setminus dom(\eta)$*.*

Definition 12. *The* ***application*** *of a solution** *mapping* η *to an RSP-QL** *pattern* P*, denoted by* $\eta[P]$*, is the RSP-QL** *pattern obtained by replacing all variables in* P *according to* η*.*

We now define the corresponding algebra operators *join*, *union*, and *left join*.

Definition 13. *Let* Ω_1 *and* Ω_2 *be sets of solution* mappings.*

$\Omega_1 \bowtie \Omega_2 = \{\eta_1 \cup \eta_2 \mid \eta_1 \in \Omega_1, \eta_2 \in \Omega_2,\ \eta \text{ and } \eta' \text{ are compatible}\}$
$\Omega_1 \cup \Omega_2 = \{\eta \mid \eta \in \Omega_1 \text{ or } \eta \in \Omega_2\}$
$\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup \{\eta \in \Omega_1 \mid \forall \eta' \in \Omega_2 : \eta \text{ and } \eta' \text{ are not compatible}\}$

Based on these algebra operators, RSP-QL* patterns are evaluated over a background dataset and a set of named windows at a given timestamp.

Definition 14. *Let* W *be a partial function that maps some IRIs in* $\mathcal{I}$ *to a window over some RDF** *stream, respectively, and* P *be an RSP-QL** *pattern such that for every sub-pattern (WINDOW* n P'*) in* P *with* $n \in \mathcal{I}$*, it holds that* W *is defined for* n*, i.e.,* $n \in \mathrm{dom}(W)$*. Furthermore, let* D *be an RDF** *dataset,* G *be an RDF** *graph, and* τ *be a timestamp. Then, the* ***evaluation of P*** *over* D *and* W *at* τ *with* G*, denoted by* $[\![P]\!]_G^{D,W,\tau}$*, is defined recursively as follows:*

1. *If* P *is a triple** *pattern* tp*, then* $[\![P]\!]_G^{D,W,\tau} = \{\eta \mid dom(\eta) = var(tp)$ *and* $\eta(tp) \in G\}$ *where* $var(tp)$ *denotes the set of variables occurring in* tp*.*
2. *If* P *is (GRAPH* u P'*), then* $[\![P]\!]_G^{D,W,\tau} = [\![P']\!]_{G'}^{D,W,\tau}$ *where* $(u, G') \in D$
3. *If* P *is (GRAPH ?x* P'*), then* $[\![P]\!]_G^{D,W,\tau} = \bigcup_{(u,G') \in D} [\![\mathit{GRAPH}\ u\ P']\!]_{G'}^{D,W,\tau}$
4. *If* P *is (WINDOW* u P'*), then* $[\![P]\!]_G^{D,W,\tau} = [\![P']\!]_{G'}^{DS(\mathcal{W}),\emptyset,\tau}$ *where* $\mathcal{W} = W(u)$ *and* G' *is the default graph of the RDF** *dataset* $DS(\mathcal{W})$
5. *If* P *is (WINDOW ?x* P'*), then* $[\![P]\!]_G^{D,W,\tau} = \bigcup_{u \in \mathrm{dom}(W)} [\![\mathit{WINDOW}\ u\ P']\!]_G^{D,W,\tau}$
6. *If* P *is (P1 AND P2), then* $[\![P]\!]_G^{D,W,\tau} = [\![P1]\!]_G^{D,W,\tau} \bowtie [\![P2]\!]_G^{D,W,\tau}$
7. *If* P *is (P1 UNION P2), then* $[\![P]\!]_G^{D,W,\tau} = [\![P1]\!]_G^{D,W,\tau} \cup [\![P2]\!]_G^{D,W,\tau}$
8. *If* P *is (P1 OPT P2), then* $[\![P]\!]_G^{D,W,\tau} = [\![P1]\!]_G^{D,W,\tau} ⟕ [\![P2]\!]_G^{D,W,\tau}$

It remains to define the semantics of RSP-QL* queries, which contain window declarations in addition to an RSP-QL* pattern (cf. Definition 9).

Definition 15. *Let $\mathcal{S}$ be a finite set of named RDF* streams and $q = (\omega, P)$ be an RSP-QL* query such that for every IRI $u_S \in \mathrm{dom}(\omega)$ there exists a named RDF* stream $(u_S, S) \in \mathcal{S}$. Furthermore, let D be an RDF* dataset and τ be a timestamp. The* ***evaluation of q*** *over D and $\mathcal{S}$ at τ, denoted by $[\![q]\!]^{D,\mathcal{S},\tau}$, is defined as $[\![q]\!]^{D,\mathcal{S},\tau} = [\![P]\!]_G^{D,W,\tau}$ where G is the default graph of D and W is a partial function such that $\mathrm{dom}(W) = \mathrm{dom}(\omega)$ and for every IRI $u \in \mathrm{dom}(W)$, it holds that $W(u)$ is the time-based window $\mathcal{W}(S, x - \alpha, x)$ with $(u_S, S) \in \mathcal{S}$, $(u_S, \alpha, \beta, \tau_0) = \omega(u)$ and $x = \tau_0 + \alpha + \beta \times i$ for the greatest possible $i \in \mathbb{N}$ for which $x < \tau$.*

7 Application-Based Evaluation

In this section, we evaluate RSP-QL* based on the application use-case scenario introduced in Section 3. To this end, we make three assumptions: First, we assume that all parameters about the patient are provided in separate streams. Second, the thresholds for the physiological parameters are context dependent, and we assume that the background data contains information about Rut's expected values with respect to some activity. Third, we assume that all physiological parameters are reported with a confidence value representing some inherent uncertainty of the sample.

Listing 1.2 illustrates a typical query for the application scenario. For the sake of readability, we have simplified the query slightly compared to the actual project application. Additional optimization strategies would also be employed in practice to provide improved scalability.

The inputs to the query are 5 different streams that report data about the patient's current heart rate, breathing rate, oxygen saturation, location (of both Rut and Rut's partner), and current activity, respectively. The activity stream might have been created by another reasoning mechanism in the system, which infers activities of daily life based on sensor inputs and the context. For each window, the values are filtered for specific values or a confidence threshold, and then the aggregated data is checked against the threshold values specific to the current context of the patient (e.g., including the type of activity). If these conditions are met, we consider it a low-emergency situation, as described in the scenario outlined in Section 3. The resulting event is pushed to another stream upon which the system can act appropriately. In our use-case scenario, the system would first contact Rut's partner. Similar queries could be set up to deal with other situations that the system should be able to detect.

The application of RSP-QL* to this project use case shows that it is possible to express the queries needed, and that the proposed language thereby fulfills our use-case based requirements. In particular, it is worth noting the compactness and relative readability of the query in Listing 1.2, as compared to the corresponding RDF reification query[5] (excluded to to space constraints).

[5] https://github.com/keski/RSPQLStarEngine/tree/master/publications/semantics2019

```
BASE <http://base/>
PREFIX ex: <http://www.example.org/ontology#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX sosa: <http://www.w3.org/ns/sosa/>

REGISTER STREAM <alert/lowEmergencyAbnormalState> COMPUTED EVERY PT10S AS

SELECT ?activity (AVG(?hr) AS ?avgHr) (AVG(?br) AS ?avgBr) (AVG(?ox) AS ?avgOx)
FROM NAMED WINDOW <w/1> ON <s/activity>  [RANGE PT10M STEP PT10S]
FROM NAMED WINDOW <w/2> ON <s/location>  [RANGE PT10M STEP PT10S]
FROM NAMED WINDOW <w/3> ON <s/heart>     [RANGE PT1M STEP PT10S]
FROM NAMED WINDOW <w/4> ON <s/breathing> [RANGE PT1M STEP PT10S]
FROM NAMED WINDOW <w/5> ON <s/oxygen>    [RANGE PT1M STEP PT10S]
WHERE {
    ?person a foaf:Person ;
       foaf:name "Rut" ;
       ex:home ?home ;
       ex:partner ?partner .

    [] a ex:NormalSituation ;
       ex:forPerson ?person ;
       ex:forActivity ?activity ;
       ex:expectedHeartRate [ ex:upperBound ?hrMax ] ;
       ex:expectedBreathingRate [ ex:lowerBound ?brMin ] ;
       ex:expectedOxygenSaturation [ ex:lowerBound ?oxMin ; ex:upperBound ?oxMax ] .

    WINDOW <w/1> { # Current activity, reported by the system
        GRAPH ?g1 {
            [ a sosa:Observation ;
              sosa:featureOfInterest ?person ;
              sosa:hasSimpleResult ?activity ] .
        }
    }
    WINDOW <w/2> { # Location of Rut's partner
        GRAPH ?g2 {
            [ a sosa:Observation ;
              sosa:featureOfInterest ?partner ;
              sosa:hasSimpleResult ?loc ] .
            FILTER(?loc != ?home)
        }
    }
    WINDOW <w/3> { # Heart rate
        GRAPH ?g3 {
            ?o3 a sosa:Observation ;
                sosa:featureOfInterest ?person .
            <<?o3 sosa:hasSimpleResult ?hr>> ex:confidence ?c3 .
            FILTER(?c3 > 0.95)
        }
    }
    WINDOW <w/4> { # Breathing rate
        GRAPH ?g4 {
            ?o4 a sosa:Observation ;
                sosa:featureOfInterest ?person .
            <<?o4 sosa:hasSimpleResult ?br>> ex:confidence ?c4 .
            FILTER(?c4 > 0.95)
        }
    }
    WINDOW <w/5> { # Oxygen saturation
        GRAPH ?g5 {
            ?o5 a sosa:Observation ;
                sosa:featureOfInterest ?person .
            <<?o5 sosa:hasSimpleResult ?ox>> ex:confidence ?c5 .
            FILTER(?c5 > 0.95)
        }
    }
}
GROUP BY ?activity ?hrMax ?brMin ?oxMin ?oxMax
HAVING(?avgHr > ?hrMax && ?avgBr < ?brMin && ?oxMin <= ?avgOx && ?avgOx <= ?oxMax)
```

Listing 1.2: The RSP-QL* query used in the use-case evaluation.

8 Performance Evaluation

In this section, we begin by briefly describing a prototype implementation of the proposed approach. We then report on the effects of the proposed RDF stream model with respect to data bandwidth, and compare it with a baseline approach of using RDF reification. Finally, we compare the query execution performance of the prototype when using RDF* as opposed to RDF reification, while varying the number of annotated triples per streamed element.

All experiments were run on a MacBook Pro with 16 GB 1600 MHz DDR3 memory, and a 2.8 GHz Intel Core i7. The experiments were run using Java 1.8.0 with 2048 MB allocated for the JVM. All experiments were preceded by warm-up runs and averages for execution times were collected only after memory usage had stabilized.

8.1 Prototype implementation

We implemented the prototype using Apache Jena[6] and RDFstarTools[7], where the latter provides a collection of Java libraries for processing RDF* data and SPARQL* queries. Additionally, we implemented a separate RSP-QL* query parser and integrated it with the standard Jena architecture, along with an extension of Jena's query class to support the additional syntax elements defined in RSP-QL*.

For the query execution, the implementation provides an extension of Jena's query engine and query execution, supporting the new query operators. During query execution, all windows over streams are materialized as individual RDF* datasets. The execution's active dataset then changes as needed when a window operation is evaluated. To improve evaluation efficiency, all parsed nodes are encoded as integers in one of two dictionaries: the *node dictionary* or the *reference dictionary*. Regular RDF nodes are added to the node dictionary, while triple nodes are added to the reference dictionary, which (recursively) encodes each separate node of the triple. All nodes, regardless of type, are internally represented as an integer, where the most significant bit signals whether the ID represents a regular node or a reference triple. This allows the system to quickly check how a node should be decoded. Encoding and decoding iterators are provided to support moving between ID-based iterators, and Jena's standard iterator implementations.

The prototype is provided as open-source[8] under the MIT License. The underlying data structures can easily be changed by providing alternative implementations for the corresponding interfaces.

8.2 Serialization Overhead

One of the side-effects of using RDF reification to annotate triples is that it increases the size of the dataset, since for each reification triple four additional

[6] https://jena.apache.org/ (version 3.8.0)
[7] https://github.com/RDFstar/RDFstarTools (version from 2019-02-28)
[8] https://github.com/keski/RSPQLStarEngine

triples have to be added. Thus, one of the benefits of the proposed extension is the reduced overhead involved in transferring statement-level annotations in data streams. To compare the impact on bandwidth requirements, we compared the overhead in terms of bytes for each of the two approaches. The data was serialized using TriG⋆, which is an extension of Turtle⋆ [9] for supporting named graphs, and compressed[9].

The amount of metadata per annotated triple impacts the relative overhead of the two approaches. For this evaluation, the TriG⋆ serialization of each RDF⋆ stream element contains declarations of one prefix, a base IRI, and a single metadata statement per annotated triple. Fig. 1 shows the bandwidth required by the approaches as a function of the number of annotated triples per streamed element. The results show that the amount of bytes required when using RDF⋆ is around half of what is required when using RDF reification.

8.3 Query Execution Performance

The performance of the approach was evaluated on the prototype implementation. The streamed elements contained a single confidence annotated triple, where the number of additional triples annotated with some other metadata predicate varied between experiments runs. A single evaluation query was used to match and filter all triples annotated with the confidence value. We compared query execution times when representing the metadata using RDF⋆ and querying it using RSP-QL⋆ versus representing the metadata using RDF reification and querying it using pure reification-based RSP-QL queries. The prototype applies no specific optimization techniques for the queries; thus, the two approaches differ only with respect to how statement-level metadata is represented internally. The RDF reification approach simply uses regular triple-pattern matching, whereas the RDF⋆ approach represents the annotated triples as resources on the physical level. For the RDF reification query, we provided an additional version of the query optimized based on the heuristics described by Tsialiamanis et al. [19], where the order of the matched triple patterns was determined based on selectivity. Fig. 2 presents the average query execution times. The results show that the advantage of the proposed approach grows with the number of distinct triples annotated in each streamed element, but that this difference can potentially be reduced by applying established optimization heuristics.

9 Discussion

The proposed approach provides a compact and intuitive way for both representing and querying annotated triples. Other approaches that could be considered for this purpose include single-triple named graphs [12], singleton properties [15], and RDF reification [11], but these approaches come with various drawbacks.

[9] Compression here included the removal of excessive whitespace characters, the use of prefixes, and the use of predicate lists where appropriate.

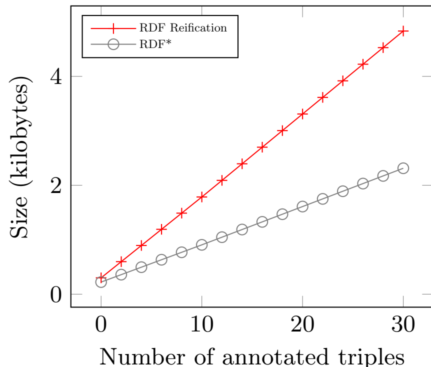

Fig. 1: Byte size of a serialized RDF⋆ stream element as a function of the number of triples in it, where each triple is annotated with exactly one metadata triple.

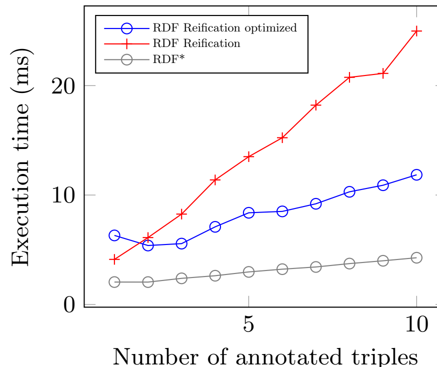

Fig. 2: Average query execution times of the prototype implementation when using either RDF⋆, reification, or reification with query optimization based on selectivity.

The application of named graphs inhibits the use of the graph name for other purposes, which means it is not compatible with the structure of RDF stream elements. Singleton properties introduce large numbers of unique predicates, which can adversely affect query execution performance. RDF reification, on the other hand, is both part of the RDF standard and can be supported in RSP-QL. However, RDF reification is verbose, both with respect to representing and querying data.

We note that RDF⋆ and SPARQL⋆ may be understood simply as syntactic sugar on top of RDF and SPARQL [9], and by extension this applies to the approach presented in this paper. However, the evaluation of the prototype implementation illustrates that representing annotated triples as resources on the physical level can have positive effects on the query execution level. When matching a single RDF reification triple, a total of four additional triple patterns have to be evaluated. In fact, due to this inefficiency, many RDF stores implement specific strategies for representing annotated triples [8]. For example, Virtuoso[10] encodes RDF reification statements as quads, Apache Jena[11] provides an implementation of a node type with direct access to the statement it reifies, and Blazegraph[12] uses an approach similar to the one implemented in our prototype.

RDF⋆ and SPARQL⋆, and thus RSP-QL⋆, simplifies the representation of complex scenarios, both from the perspective of modeling and of querying annotated metadata. For example, we may want to treat an RDF statement differently depending on whether the uncertainty associated with it has been automatically

[10] https://virtuoso.openlinksw.com/
[11] https://jena.apache.org/
[12] https://wiki.blazegraph.com/

generated by a sensor, or if it originates from a physician. Querying this using RSP-QL⋆ simply involves having a triple⋆ pattern with two layers of nesting.

As part of future work, we plan on relaxing some of the assumptions made in the semantics, and add support for additional features defined in RSP-QL, such as count-based window operators and output stream operators.

10 Conclusion

In this paper, we have presented a novel way of annotating and querying statement-level metadata in RDF Stream Processing (RSP), and formally defined the new continuous query language RSP-QL⋆. The approach extends RDF streams to allow triples to directly use other triples in the subject and object positions, and similarly extends the current version of RSP-QL to query these, by leveraging and building on the concepts previously proposed for RDF⋆ and SPARQL* [9,10].

The proposed approach was applied in a use case from an e-health research project, where multiple data streams have to be queried in parallel, and over extended periods of time, to detect possibly abnormal situations. The results show that RSP-QL⋆ meets all our use-case requirements, and provides a compact and intuitive way of expressing and querying statement-level metadata, compared with the baseline approach of using RDF reification. Furthermore, the prototype implementation presented in the paper, which is provided as open-source, demonstrates benefits over the baseline approach, both with respect to the bandwidth required for data transfer and with respect to query execution performance over statement-level annotations. RDF⋆ is a syntactically more compact way to express metadata annotations, and our experiments show that this difference is large enough to have an impact in deployed real-world systems and applications, where bandwidth may be limited. Although our prototype implementation is not optimized for query performance, we were able to demonstrate that the approach was faster with respect to query execution performance, when compared to using standard RDF reification.

This is the first work on RSP that has focused on supporting annotations on the statement level. We believe that the proposed approach provides an intuitive and compact way for representing and querying statement-level metadata, and that this work provides a good foundation for future research on efficient management of, e.g., uncertainty and provenance, in RDF data streams.

Acknowledgements. This research was supported by E-care@home, a "SIDUS – Strong Distributed Research Environment" project, funded by the Swedish Knowledge Foundation (KK-stiftelsen, Dnr: 20140217). Project website: http://ecareathome.se/. Olaf Hartig's work on this paper has been funded by the CENIIT program at Linköping University (project no. 17.05).

References

1. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning. In: Proceedings of the 20th International Conference on World Wide Web (2011)
2. Athan, T., Anderson, J., Ortner, P.W.B.: RDF Stream Abstract Syntax and Semantics: Draft Community Group Report 22 August 2016 (2016)
3. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF Streams with C-SPARQL. ACM SIGMOD Record **39**(1), 20–26 (Sep 2010)
4. Calbimonte, J.P., Corcho, O., Gray, A.J.G.: Enabling Ontology-Based Access to Streaming Data Sources. In: 9th International Semantic Web Conf. (ISWC) (2010)
5. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax. Tech. rep., W3C (2014), http://www.w3.org/TR/rdf11-concepts/
6. Dell'Aglio, D., Calbimonte, J.P., Della Valle, E., Corcho, O.: Towards a Unified Language for RDF Stream Query Processing. In: Revised Selected Papers of the ESWC 2015 Satellite Events on The Semantic Web: ESWC 2015 Satellite Events - Volume 9341. pp. 353–363. Springer-Verlag New York, Inc., New York, NY, USA (2015). https://doi.org/10.1007/978-3-319-25639-9_48, http://dx.doi.org/10.1007/978-3-319-25639-9_48
7. 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. Journal on Semantic Web and Information Systems **10**(4), 17–44 (2014)
8. Frey, J., Müller, K., Hellmann, S., Rahm, E., Vidal, M.E.: Evaluation of metadata representations in RDF stores. Semantic Web Journal **10**, 205–229 (2017)
9. Hartig, O.: Foundations of RDF* and SPARQL* – An Alternative Approach to Statement-Level Metadata in RDF. In: Proc. of the 11th Alberto Mendelzon Int. Workshop on Foundations of Data Management and the Web (2017)
10. Hartig, O., Thompson, B.: Foundations of an Alternative Approach to Reification in RDF. CoRR **abs/1406.3399** (2014)
11. Hayes, P.J., Patel-Schneider, P.F.: RDF 1.1 Semantics. Tech. rep., W3C (2014), https://www.w3.org/TR/rdf11-mt/
12. Hernández, D., Hogan, A., Krötzsch, M.: Reifying RDF: What Works Well With Wikidata? In: Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS) (2015)
13. 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 Semantic Web Conference (ISWC) (2011)
14. Lind, L., Prytz, E., Lindén, M., Kristoffersson, A.: Use cases unified description. E-care@home project Milestone Report MSR5.1b (Project Internal) (2017)
15. Nguyen, V., Bodenreider, O., Sheth, A.: Don't Like RDF Reification?: Making Statements About Statements Using Singleton Property. In: Proceedings of the 23rd International Conference on World Wide Web (2014)
16. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. ACM Trans. Database Syst. **34**(3) (2009)
17. Prud'hommeaux, E., Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. Tech. rep., W3C (2013), http://www.w3.org/TR/sparql11-query
18. Rinne, M., Nuutila, E., Törmä, S.: Instans: High-performance event processing with standard rdf and sparql. In: ISWC 2012 Posters & Demos Track (2012)
19. Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., Boncz, P.: Heuristics-based Query Optimisation for SPARQL. In: Proceedings of the 15th International Conference on Extending Database Technology (2012)