Stream Reasoning: From Theory to Practice

Emanuele Falzone[1](✉), Riccardo Tommasini[2], and Emanuele Della Valle[1]

[1] Politecnico di Milano, DEIB, Milan, Italy
{emanuele.falzone,emanuele.dellavalle}@polimi.it
[2] DataSystem Group, University of Tartu, Tartu, Estonia
riccardo.tommasini@ut.ee

Abstract. Stream Reasoning is set at the confluence of Artificial Intelligence and Stream Processing with the ambitious goal to reason on rapidly changing flows of information. The goals of the lecture are threefold: (1) Introducing students to the state-of-the-art of Stream Reasoning, (2) Deep diving into RDF Stream Processing by outlining how to design, develop and deploy a stream reasoning application, and (3) Jointly discussing the limits of the state-of-the-art and the current challenges.

1 Introduction

We live in a streaming world [8]. Data are no longer just vast and various, they are also produced faster every day. Social networks, Internet of Things deployments for healthcare or smart cities, as well as modern news infrastructures provision data continuously in the form of *data streams*, i.e., infinite sequences of timestamped data.

Since more and more streams are becoming available, the underlying Web infrastructure evolved to include new protocols for real-time data provisioning, e.g., Web Sockets[1]. Figure 1 visualizes an example of a data stream and helps us introduce the running example that we will use along this manuscript.

Example 1 (Color Stream). In our running example, we will observe a stream of colors (or colored boxes). Each element is a timestamped color observation made by a sensor.

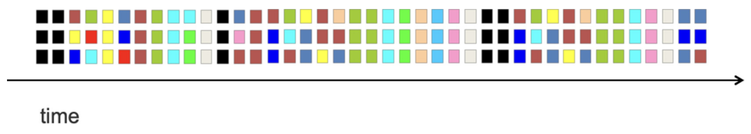


Fig. 1. An example of Stream Processing over a stream of colored boxes.

[1] https://tools.ietf.org/html/rfc6455.

M. Manna and A. Pieris (Eds.): Reasoning Web 2020, LNCS 12258, pp. 85–108, 2020.
https://doi.org/10.1007/978-3-030-60067-9_4

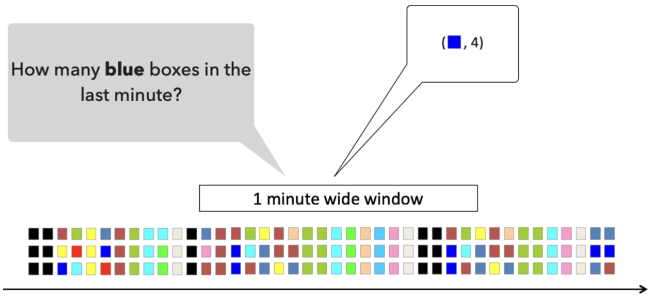


Fig. 2. An example of continuous query over the color stream. (Color figure online)

In these scenarios, data-intensive applications must deal with many new data challenges simultaneously, e.g., Data Volume and Data Velocity [17]. However, from new challenges, new opportunities arise: Stream Processing is the research field that studies how to analyze vast data streams over time. The related literature contains models, systems, and languages that lead to continuous queries and, in particular, to windowing operations [1,14].

Example 2 (Color Stream Processing (cont'd)). Continuous queries can be registered on the color stream, e.g., for counting the number of blue boxes in the last minute as shown in Fig. 2.

Additionally, data-intensive applications must deal with information coming from a variety of sources. Thus, a common problem is how to integrate data streams with background domain knowledge without violating specific time constrains. However, traditional data integration techniques are not adequate to tame variety and velocity at the same time. Indeed, such techniques are based on semantic technologies and require to put data at rest. Thus, the following research question arises.

How can we make sense, in real-time, of vast, noisy, heterogeneous data streams coming from complex domains?

The Stream Reasoning (SR) research field addresses the challenges unveiled by the research question above [8]. SR's ultimate goal is to design a new generation of systems capable of addressing Data Variety and Velocity simultaneously. To this extent, SR's research combines the lessons learned on Stream Processing with the extensive work of the Semantic Web community.

Since 2008, the results of Stream Reasoning research made an impact in both the Stream Processing and Semantic Web areas [9,18]. In particular, RDF Stream Processing (RSP) extends the Semantic Web Stack for continuous querying and reasoning upon rapidly changing information flows [12].

Example 3 (Color Stream Reasoning). Continuous queries over streams can be enriched with contextual domain knowledge. For instance, for counting the *cool*

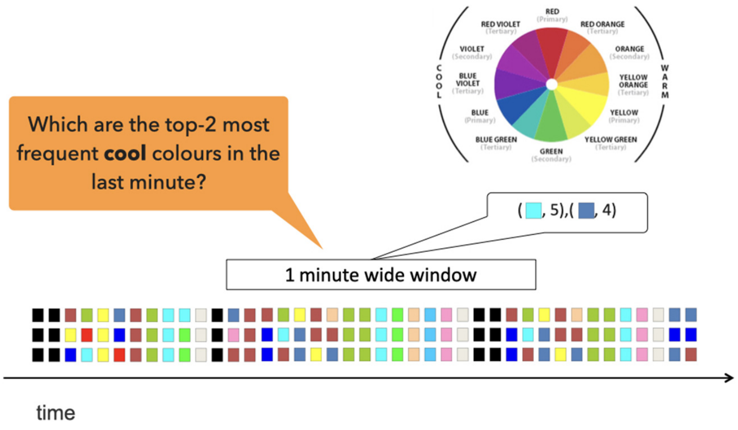


Fig. 3. An example of Stream Reasoning on a stream of colored boxes.

colors in the last minute window we must add an ontology of colors as shown in Fig. 3. Then, such ontology can be used for identifying cool colours.

In this lecture, we will walk through some prominent Stream Reasoning achievements. In particular, we focus on the area of RDF Stream Processing. We present theoretical results as well as practical guidelines and resources to build RDF Stream Processing applications.

The remainder of the paper is organized as follows: Sect. 2 presents preliminary knowledge about Continuous Processing (Sect. 2.1), RDF (Sect. 2.2) and SPARQL (Sect. 2.3). Section 3 presents the life-cycle of Streaming Linked Data. In particular, it digs into streaming data publications with five simple steps for practitioners to apply. Section 4 presents RDF Stream Processing, i.e., the last step of the Streaming Linked Data life-cycle. To this extent, the section gives an introduction to RSP-QL, i.e., the reference model for Web Stream Processing using RDF Streams. (Sect. 4.1). Moreover, the section presents canonical problems in Stream Processing on the Web. Section 5 presents the *Linear Pizza Oven*, i.e., an exercise for getting started with Streaming Linked Data and RDF Stream Processing using RSP-QL. Finally, Sect. 6 draws conclusions.

2 Preliminaries

Before providing guidelines to develop Stream Reasoning applications, we need to formalize the concepts necessary for querying and reasoning over data streams.

2.1 Continuous Queries

Continuous queries, a.k.a. persistent or standing queries, are a special class of queries that listens to updates and allow interested users to receive new results as soon as data becomes available. They are similar to traditional database queries, with the difference that they are issued once, but they run repeatedly over the input streams. This idea, known as *continuous semantics*, conveys the following intuition, i.e., the processing of an infinite input produces an infinite output [26].

Under continuous semantics, the result of a query is the set of results that would be returned if the query were executed at every instant in time.

Upon the intuition above, many query languages have been designed for writing continuous queries. Among them, the Continuous Query Language (CQL) [1] passed the test of time for what concerns relational data Stream Processing.

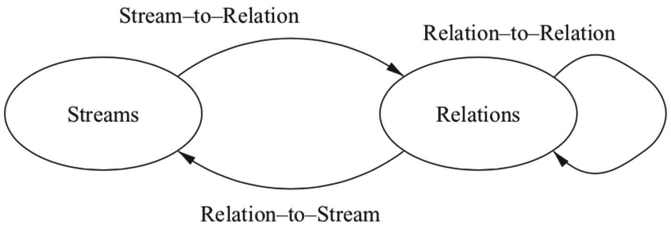


Fig. 4. The operator classes defined in CQL.

As shown in Fig. 4, CQL gives an abstract semantics to continuous queries in terms of two data types: *Streams* and *Relations*.

Definition 1. *Let T be the ordered time domain, e.g., the set of natural numbers $\mathcal{N}$. A data stream S is a possibly infinite multiset of elements $\langle o, \tau \rangle$, where o is a data item, e.g., a tuple, and $\tau \in T$ is a timestamp, e.g., a natural number.*

Definition 2. *A relation R is a mapping from each time instant $\tau \in T$ to a finite but unbounded bag of tuples belonging to a fixed schema consisting of a set of named attributes.*

Upon these two abstractions, CQL defines three classes of operators that, together, allow to write continuous queries over data streams:

- Stream-to-Relation (S2R) operators that produce a relation from a stream,
- Relation-to-Relation (R2R) operators that produce a relation from one or more other relations, and
- Relation-to-Stream (R2S) operators that produce a stream from a relation.

More specifically, S2R operators in CQL are operators that chuck a stream S into Windows. A window, denoted as W(S), is a set of elements extracted from a stream by a *window operator*. CQL describes time-based, tuple-based, and partitioned window operators. Nevertheless, in the following we will limit our scope to time-based sliding window operators.

Definition 3. *A time-based sliding window operator on a stream S takes a time interval I as a parameter and outputs at τ the relation R of S[Range I] defined as:*

$$R(\tau) = \{s | \langle s, \tau' \rangle \wedge \tau' \leq \tau \wedge \tau' \geq max\{\tau - I, 0\}\}$$

R2R operators correspond to relational operators adapted to handle time-varying relations. Assuming the familiarity of the readers with Relational Algebra (RA), in the following we focus on the R2S operators that include:

– The *insert stream* operator that streams out all new entries w.r.t. the previous instant as per Definition 4.
– The *delete stream* operator that streams out all deleted entries w.r.t the previous instant as per Definition 5.
– The *relation stream* operator that streams out all elements at a certain instant in the source relation as per Definition 6.

Given two consecutive time instants $\tau - 1$ and τ we formally define R2S operators as follows:

Definition 4. *The insert stream operator applied to a relation R emits an element $\langle s, \tau \rangle$ if and only if the tuple s is in R(τ) − R(τ − 1) at time τ:*

$$Istream(R) = \bigcup_{\tau \geq 0} ((R(\tau) - R(\tau - 1)) \times \{\tau\}$$

Definition 5. *The delete stream operator applied to a relation R emits an element $\langle s, \tau \rangle$ if and only if the tuple s is in R(τ − 1) − R(τ) at time τ:*

$$Dstream(R) = \bigcup_{\tau \geq 0} ((R(\tau - 1) - R(\tau)) \times \{\tau\}$$

Definition 6. *The relation stream operator applied to a relation R emits an element $\langle s, \tau \rangle$ if and only if the tuple s is in R(τ) at time τ:*

$$Rstream(R) = \bigcup_{\tau \geq 0} (R(\tau)) \times \{\tau\}$$

2.2 Resource Description Framework

The *Resource Description Framework* (RDF) is a semistructured data model that fosters data interchange and allows publishing machine-readable representation of resources [32]. RDF requires to organize information as triples, i.e., *(subject,predicate,object)* statements.

Definition 7. *An RDF statement t is a triple*

$$(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$$

where I is the set of IRIs (Internationalized Resource Identifiers). B is the set of all the blank nodes, i.e., representing identifiers of anonymous resources. L is the set of literals, i.e., string values associated with their datatype. I, B, and L are disjoint from each other.

RDF triples are organized in RDF graphs (cf. Definition 8), and RDF graphs can be organized into datasets (cf. Definition 9).

Definition 8. *An RDF graph is a set of RDF triples.*

Definition 9. *An RDF Dataset DS is a set:*

$$DS = \{g_0, (u_1, g_1), (u_2, g_2), ..., (u_n, g_n)\}$$

where g_0 and g_i are RDF graphs, and each corresponding u_i is a distinct IRI. g_0 is called the default graph, while the others are called named graphs.

RDF is a data model and it can be serialized using different data formats. According to W3C, the default RDF serialization is RDF/XML[2]. However, it supports other serialization formats, e.g., Turtle[3] (TTL), JSON for Linked Data (JSON-LD)[4], NTriples[5], and TriG. In the following, we adopt mostly Turtle and JSON-LD as the default serialization because they are more readable.

```
1 PREFIX p: <http://example/>
2 <http://example/sbjct1> <http://example/pred1> <http://example/objct1> .
3
4 <http://example/subject2>
5     <http://example/predicate2> <http://example/object2> ;
6     <http://example/predicate3> <http://example/object3> .
7
8 p:subject4 p:predicate4 p:object4 , p:object5  .
```

Listing 1.1. Example illustrating Turtle Syntax for RDF triples.

Listing 1.1 shows at Line 2 an example of an RDF triple. Turtle syntax allows grouping triples with the same subject, separating different predicates using a semicolon (cf. Lines 5 and 6). It allows separating alternative objects for the same predicate using a comma (cf. Line 8). It allows defining prefixes for making documents more succinct. Line 1 declares a prefix IRI, which is used by the triple in the following line.

2.3 SPARQL

SPARQL is a graph query language and a protocol for RDF data. Listing 1.2 shows an example of SPARQL query which consists of three clauses, i.e., Dataset Clause, Query Form, and Where Clause. The language also includes solution modifiers such as DISTINCT, ORDER BY and GROUP BY. Finally, it is possible to associate a prefix label to an IRI.

SPARQL's Dataset Clause, indicated in Fig. 10 (a), determines the RDF dataset in the scope of the query evaluation. This clause employs the *FROM*

[2] https://www.w3.org/TR/rdf-syntax-grammar/.
[3] https://www.w3.org/TR/turtle/.
[4] https://www.w3.org/TR/json-ld11/.
[5] https://www.w3.org/TR/n-triples/.

```
 1 PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
 2 SELECT ?name ?email
 3 FROM <http://www.w3.org/People/Berners-Lee/card>
 4 WHERE {
 5     ?person foaf:name ?name ;
 6             foaf:age ?age .
 7     FILTER (?age > 18)
 8 }
 9 ORDER BY ?name
10 LIMIT 10
11 OFFSET 10
```

Listing 1.2. Example SPARQL query.

operator to load an RDF graph into the Dataset's default graph. Moreover, the *FROM NAMED* operator allows loading an RDF graph into a separated RDF graph identified by its IRI.

Definition 10. *A triple pattern t_p is a triple (sp, pp, op) s.t.*

$$(sp, pp, op) \in (I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$$

where I, B, and L are defined as in Definition 7, while V is the infinite set of variables.

SPARQL's Where Clause allows defining the patterns to match over an RDF dataset. During the evaluation of a query, the graph from the dataset used for matching the graph pattern is called *Active Graph*. By default, the *Active Graph* is set to the default graph of the dataset. The basic match unit in SPARQL is the Triple pattern (cf. Definition 10). Listing 1.2 presents two triple patterns (cf. Lines 5–6). A set of triple patterns is called a Basic Graph Pattern (BGP). BGPs can be named or not, and in the first case they are identified by either a variable or an IRI. BGPs can include other compound patterns defined by different algebraic operators [22]. Listing 1.2 shows an example of FILTER clause that checks whether age is higher than 18.

Last but not least, SPARQL's Query Form Clause allows specializing the query results. SPARQL includes the following forms: (i) The *SELECT* form includes a set of variables and it returns the corresponding values according to the query results. (ii) The *CONSTRUCT* form includes a graph template and it returns one or more RDF graphs filling the template variables according to the query results. (iii) The *DESCRIBE* form includes a resource identifier and it returns an RDF graph containing RDF data that describe a resource. (iv) The *ASK* form just returns a boolean value telling whether or not the query pattern has at least a solution.

Having introduced the syntax, we can now give more details about SPARQL semantics. A SPARQL query is defined as a tuple (E, DS, QS) where E is a SPARQL algebraic expression, DS an RDF dataset, and QF a query form.

The evaluation semantics of a SPARQL query algebraic expression w.r.t. an RDF dataset is defined for every operator of the algebra as $eval(DS(g), E)$ where E denotes an algebraic expression and $DS(g)$ a dataset DS with active graph g. Moreover, the evaluation semantics relies on the notion of solution mappings.

Definition 11. *A solution mapping μ is a partial function*

$$\mu : V \rightarrow I \cup B \cup L$$

from a set of Variables V to a set of RDF terms. A solution mapping μ is defined over a domain $dom(\mu) \subseteq V$, and $\mu(x)$ indicates the application of μ to x.

Following the notation of [11], we indicate with Ω a multiset of solution mappings, and with Ψ a sequence of solution mappings.

Definition 12. *Two mappings, μ_1 and μ_2, are said to be compatible, i.e., $\mu_1 \backsim \mu_2$ iff:*

$$\forall x \in (dom(\mu_1) \cap dom(\mu_2)) \Rightarrow \mu_1(x) = \mu_2(x)$$

Given an RDF graph, a SPARQL query solution can be represented as a set of solution mappings, each assigning term of RDF triples in the graph to variables of the query.

The evaluation function $[\![.]\!]_D$ defines the semantics of graph pattern expressions. It takes graph pattern expressions over an RDF Dataset D as input and returns a multiset of solution mappings Ω. As in [22], the evaluation of triple patterns t, and a compound graph pattern $P_i\ X\ P_j$ is defined as follows:

$$[\![t]\!]_D = \{\mu | dom(\mu) = var(t) \in D\}$$

$$[\![P_1\ AND\ P_2]\!] = \Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 | \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \mu_1 \backsim \mu_2\}$$

$$[\![P_1\ UNION\ P_2]\!] = \Omega_1 \cup \Omega_2 = \{\mu_1 \cup \mu_2 | \mu_1 \in \Omega_1 \vee \mu_2 \in \Omega_2 \mu_1 \backsim \mu_2\}$$

$$[\![P_1\ OPTIONAL\ P_2]\!] = \Omega_1 \,\overline{\bowtie}\, \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 /\ \Omega_2)$$

where

$$\Omega_1 /\ \Omega_2 = \{\mu | \mu \in \Omega_1 \wedge \nexists \mu' \in \Omega_2 \mu \backsim \mu'\}$$

3 Streaming Linked Data Life-Cycle

The mechanisms to publish and consume data streams on the Web has gained attention [13] thanks to the progresses in Stream Reasoning systems and approaches. Figure 5 summarises the publication life-cycle for streaming linked data, which consists of five steps described in the following.

Name. The Step (0) aims at designing IRIs that identify the stream itself and the other resources it may contain. Linked Data best practices for good IRIs design prescribe the usage of HTTP IRIs to identify Web resources. Indeed,

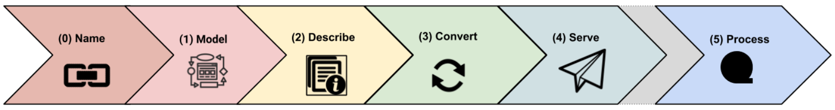


Fig. 5. Streaming Linked Data Lifecycle

despite streaming extensions of the Web architecture existing (e.g., WebSocket) identification of stream as resources should still rely on HTTP IRIs [28].

To this extent, Sequeda and Corcho [25] suggested a mechanism to identify sensors and their observations. They suggested three innovative IRI schemes to identify sources, temporal, spatial, and spatio-temporal metadata [25]. Barbieri and Della Valle recommended to identify streams using IRIs that resolve a named graph containing all the relevant metadata. These named graphs also describe the current content of the window over the stream using the properties *rdfs:seeAlso* and *:receivedAt* [4]. The latter design was further developed by Mauri et al. [19] and is the approach we will adopt in the following.

To capture the essence of what a Web Stream is we provide Definition 13, which also explains how the stream itself and the element it contains are valid Web resources, i.e., they are identifiable via IRIs.

Definition 13. *A Web data stream is a Web Resource that identifies an unbounded ordered collection of pairs* (o, i)*, where* o *is a Web resource, e.g., a named graph, and* i *is a metadata that can be used to establish an ordering relation, e.g., a timestamp.*

Example 4. (cont'd) Carrying on our running example for Step (0), the color stream that we are going to use for our running experiments is identified by the base URL http://linkeddata.stream. Moreover, we will apply the following URI schemas to identify Web Resources that are relevant for this paper:

1. `http://linkeddata.stream/ontologies/{ontologyname}`.
2. `http://linkeddata.stream/resource/{streamname}`.
3. `http://linkeddata.stream/resource/{streamname}/{graphid}`.

Model. Step (1) aims at describing the application domain from which data comes. To this extent, ontologies and models are designed and reused in order to capture the domain knowledge in machine-understandable way. During this step it is also critical to identify relevant resources, collect data samples, and formulate canonical information needs.

In the Stream Reasoning literature, several vocabularies have been designed, used, and adapted for representing the domain of streaming data. State-of-the-art vocabularies include but are not limited to FrAPPE [2], SAO [15], SSN [7] or SOSA [16], and SIOC [21].

Example 5. (cont'd) Carrying on our running example for Step (1), we must model the application domain for our color stream. Figure 6 exemplifies domain knowledge about colors. The role of a knowledge engineer is to design a formal model that allows a machine to understand such information.

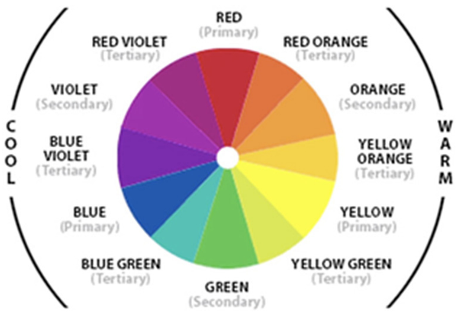


Fig. 6. Domain knowledge about colors.

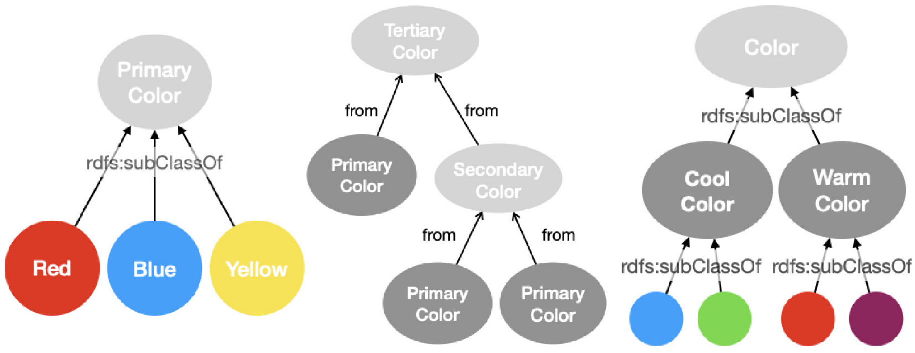


Fig. 7. Graphical representation of the color ontology.

Figures 7 exemplifies a possible way to represent the domain knowledge as a taxonomy. Notably, due to the lack of space, we represent only a sub-portion of the taxonomy. We decided to model the colors as classes in order to provide different hierarchies based on their compositions and "temperature". Consequently, individuals are instances of the colors to be identified. The OWL 2 version of such ontology is also available[6].

Describe. Step (2) aims at providing useful representations of the data streams to be consumed by humans and/or machines. It recommends to use standard vocabularies to include relevant metadata that eases the discovery of and the access to the data in the stream. For instance, during this step the data publisher shall choose an appropriate license.

Recently, Schema.org included two concepts that are relevant for stream representation, i.e., DataFeed[7] and DataFeedItem[8]. However, their adoption has not been estimated, yet.

[6] https://linkeddata.stream/ontologies/colors.owl.
[7] https://schema.org/DataFeed.
[8] https://schema.org/DataFeedItem.

On the other hand, Tommasini et al. [31] designed the Vocabulary & Catalog of Linked Streams (VoCaLS) with the goal of fostering interoperability between streaming services on the Web. VoCaLS consists of three modules that enable (i) Publishing streaming data following Linked Data principles, (ii) Describing streaming services, and (iii) Tracking the provenance of Stream Processing.

Example 6. (cont'd) Carrying on our running example for Step (2), we describe the color stream using an appropriate vocabulary. VoCaLS is our choice and Listing 1.3 provides an example of a stream descriptor, i.e., an RDF graph containing information about the color stream.

```
PREFIX : <http://linkeddata.stream/resource/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX frmt: <http://www.w3.org/ns/formats/>
PREFIX vocals: <http://w3id.org/rsp/vocals#>
PREFIX vsd: <http://w3id.org/rsp/vocals-sd#>
<> a vocals:StreamDescriptor .

:colorstream a vocals:RDFStream ;
 dcat:title "Color Stream"^^xsd:string ;
 dcat:description "Stream of primary colors"^^xsd:string ;
 dcat:license <https://creativecommons.org/licenses/by-nc/4.0/> ;
 dcat:publisher <http://linkeddata.stream> ;
 dcat:landingPage <http://linkeddata.stream/page/colorstream> ;
 vocals:hasEndpoint :ColorEndpoint  .
```

Listing 1.3. Publishing color stream with Vocals and RSP-QL

Convert. Step (3) recommends converting streaming data into a machine-readable format. As for Linked Data, RDF is the data model of choice to publish data on the Web. For streaming data, we recommend RDF Streams which are formalized by Definition 14.

Definition 14. *An RDF Stream is a data stream (cf Definition 1) where the data items o are RDF graphs, i.e., a set of RDF triples of the from (subject, predicate, object).*

Typically, streaming data is not produced directly as RDF streams. For instance, sensor push observations using a tabular format like CSV. Often, compression is also used to save bandwidth and reduce costs.

In these cases, a conversion mechanism must be set up to transform the data stream into an RDF stream. The conversion pipeline should make use of the domain ontologies designed at Step 0. Additionally, streaming data may be enriched with contextual domain knowledge, capturing the domain information collected in an ontological model.

Technologies like R2RML are adequate to set up static data conversion pipelines. Nevertheless, they present some limitations when having to deal with data streams. In particular, due to the infinite nature of streaming data, the conversion mechanism can only take into account the stream one element at a time. Alternatively, one can use a window-based Stream Processing engine to transform the input stream by means of a continuous query.

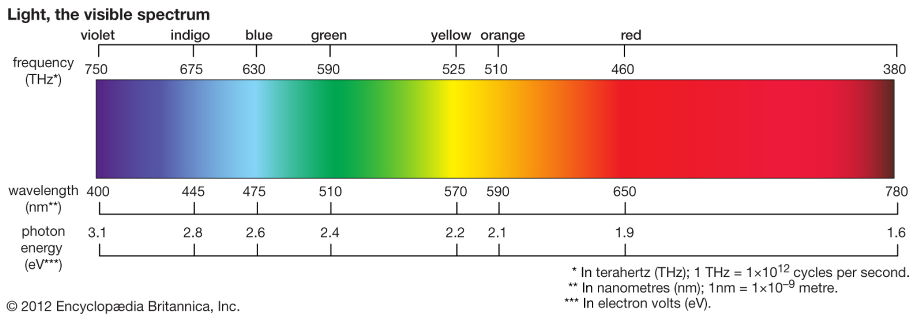


Fig. 8. Light spectrum overview.

Example 7 (cont'd). Carrying on our running example for Step (3), we exemplify an annotation process that makes use of the Color ontology to transform a sensor stream that samples the light spectrum into an RDF Stream of color instances.

According to Fig. 8, we can derive the following rules to map the sensor measurement into symbolic representations of colors:

– If the sensed frequency is between 650 and 605, the light is perceived as blue.
– If the frequency is between 605 and 545, the light is perceived as green.
– If the frequency is between 450 and 480, the light is perceived as yellow.
– If the sensed frequency is between 480 and 380, the light is perceived as red.

Table 1 shows some observation made by a sensor network about the light spectrum. Finally, Listing 1.4 displays a set of corresponding timestamped RDF graphs, denoting sample occurrences of colors.

Table 1. Sensor observation in tabular form.

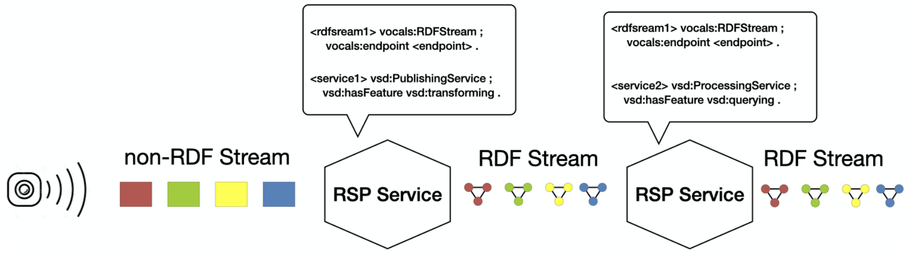
ObservationID	Sensed frequency	Sensor	Timestamp
507f704d-dfb8-4fa3-97ca-64a93e56abb0	460	S1	1588684149
6e852dd4-cd3a-4516-9665-842483d5c22f	630	S2	1588684150
84b0350a-2422-4538-8689-f0e58ce1c485	590	S1	1588684152
80350a4b-4fa3-9665-8689-g46842483ds5	525	S2	1588684399

Before describing the next steps, i.e., Publish and Process, we must introduce the notion of *RSP Service*. An RSP Service is a special kind of Web service

```
{
    "@id":"http://linkeddata.stream/streams/colorstream/1588684149",
    "@context":{
        "color":"http://linkeddata.stream/ontologies/colors#",
        "rdf":"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        "rdfs":"http://www.w3.org/2000/01/rdf-schema#"
    },
    "@graph":{
        "@id":"507f704d-dfb8-4fa3-97ca-64a93e56abb0",
        "@type":"color:Red"
    }
},
{
    "@id":"http://linkeddata.stream/streams/colorstream/1588684150",
    "@context":{
        "color":"http://linkeddata.stream/ontologies/colors#",
        "rdf":"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        "rdfs":"http://www.w3.org/2000/01/rdf-schema#"
    },
    "@graph":{
        "@id":"6e852dd4-cd3a-4516-9665-842483d5c22f",
        "@type":"color:Blue"
    }
},
{
    "@id":"http://linkeddata.stream/streams/colorstream/1588684152",
    "@context":{
        "color":"http://linkeddata.stream/ontologies/colors#",
        "rdf":"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        "rdfs":"http://www.w3.org/2000/01/rdf-schema#"
    },
    "@graph":{
        "@id":"84b0350a-2422-4538-8689-f0e58ce1c485",
        "@type":"color:Green"
    }
},
{
    "@id":"http://linkeddata.stream/streams/colorstream/1588684399",
    "@context":{
        "color":"http://linkeddata.stream/ontologies/colors#",
        "rdf":"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
        "rdfs":"http://www.w3.org/2000/01/rdf-schema#"
    },
    "@graph":{
        "@id":"80350a4b-4fa3-9665-8689-g46842483ds5",
        "@type":"color:Yellow"
    }
}
```

Listing 1.4. The RDF stream of color.

that manipulates Web streams and, in particular, RDF Streams. We identify three main types of RSP services that are relevant for the streaming linked data lifecycle: (i) *Catalogs* that provide metadata about streams, their content, query endpoints and more. (ii) *Publishers* that publish RDF streams, possibly following a Linked Data compliant scheme (e.g. TripleWave in Listing 1.5). (iii) *Processors*, which model a Stream Processing service that performs any kind of transformation on streaming data, e.g. querying (by RSP engines like CSPARQL engine [3] or CQELS [23]) or reasoning (by Stream Reasoners like RDFFox [20], TrOWL [27], or MASSIF [5]), or detection (Semantic Complex Event Processors [10, 29]).



<rdfsream1> vocals:RDFStream ; vocals:endpoint <endpoint> .
<service1> vsd:PublishingService ; vsd:hasFeature vsd:transforming .
<rdfsream1> vocals:RDFStream ; vocals:endpoint <endpoint> .
<service2> vsd:ProcessingService ; vsd:hasFeature vsd:querying .
non-RDF Stream
RSP Service
RDF Stream
RSP Service
RDF Stream

Fig. 9. Converting and publishing the sensor stream with TripleWave.

Intuitively, the first two are services relevant within streaming data publication, while the latter is relevant for processing. Due to the lack of space, in the following we focus on Publishers and Processors.

Serve. Step (4) aims at making the streaming data accessible on the Web. The goal of this step is serving the data to the audience of interest, i.e., making them available for processing.

RSP Publishers like TripleWave [19] are deployed to provision the streaming data content. TripleWave is a reusable and generic tool that enables the publication of RDF streams on the Web. It can be invoked through both pull- and push-based mechanisms, thus enabling RSP engines to automatically register and receive data from TripleWave.

```
<http://linkeddata.stream> a vsd:PublishingService ;
    vsd:hasFeature vsd:transforming .
```

Listing 1.5. The Publisher of the Color Stream.

Additionally, RSP publishing services carry on the conversion process and make the Web stream findable by publishing stream descriptors like the one presented in Listing 1.3. To this extent, VoCaLS provides additional modules for describing the provisioning service (e.g., Listing 1.5) as well as tracking the provenance of the stream transformation.

Finally, streaming data access is made possible through the publication of Stream Endpoints that refer to the appropriate protocols to the continuous or reactive consumption of Web data (e.g., WebSockets or Server-Sent Events).

```
:ColorEndpoint a vocals:StreamEndpoint ;
               dcat:format frmt:Turtle ;
               dcat:accessURL "ws://colorstream:8080" .
```

Listing 1.6. An access point to the color stream.

Example 8. (cont'd) Carrying on our running example for Step (4), Listing 1.5 and Listing 1.6 complete the stream descriptor presented in Listing 1.3 with the specification of the corresponding publishing service and access point. Additionally, Fig. 9 shows how to carry on the lifecycle consuming the sensor streams, converting it, and at the same time making the stream descriptor available. This pipeline can be realised using TripleWave [19] and the CSPARQL engine [4].

Process. Finally, the aim of Step (5) is consuming published stream for analysis. We provide details on how this is possible in the next section, making use of RSP-QL, i.e., the reference language for RDF Stream Processing [10].

4 Processing

In this section, we dig into processing Web Streams using RSP-QL. To this extent, we present an RSP-QL primer that will introduce the reader to the syntax and RSP-QL main functionalities. Furthermore, we provide a comprehensive set of analytics that builds on our running example, i.e., the color stream.

4.1 RSP-QL Primer

RSP-QL extends SPARQL 1.1 syntax and semantics to define streaming transformations over RDF Streams. The RSP-QL semantic is inspired by CQL and SECRET [14]. The formal framework includes operator classes (like CQL), and primitives to describe the operational semantics (like SECRET). In particular, it is worth mentioning that as RA corresponds to R2R operators in CQL (cf. Sect. 2), so SPARQL algebra corresponds to R2R operators in RSP-QL.

Figure 10 presents the anatomy of an RSP-QL query. As explained in Sect. 2.3, a SPARQL query consists of a Dataset Clause, a Query Form, and a Where Clause. RSP-QL extends the Dataset Clause to include Window Operators (Line 5 in Listing 1.7); it introduces the `WINDOW` keywords for referring to stream in the Where Clause (Line 10 in Listing 1.7), and it adds the REGISTER clause to name the R2S operators and the output streams (Line 2 in Listing 1.7). Notably, the output of an RSP-QL query with the register clause (named RSP-QL query) is not necessarily an RDF Stream. It depends on the Query Form.

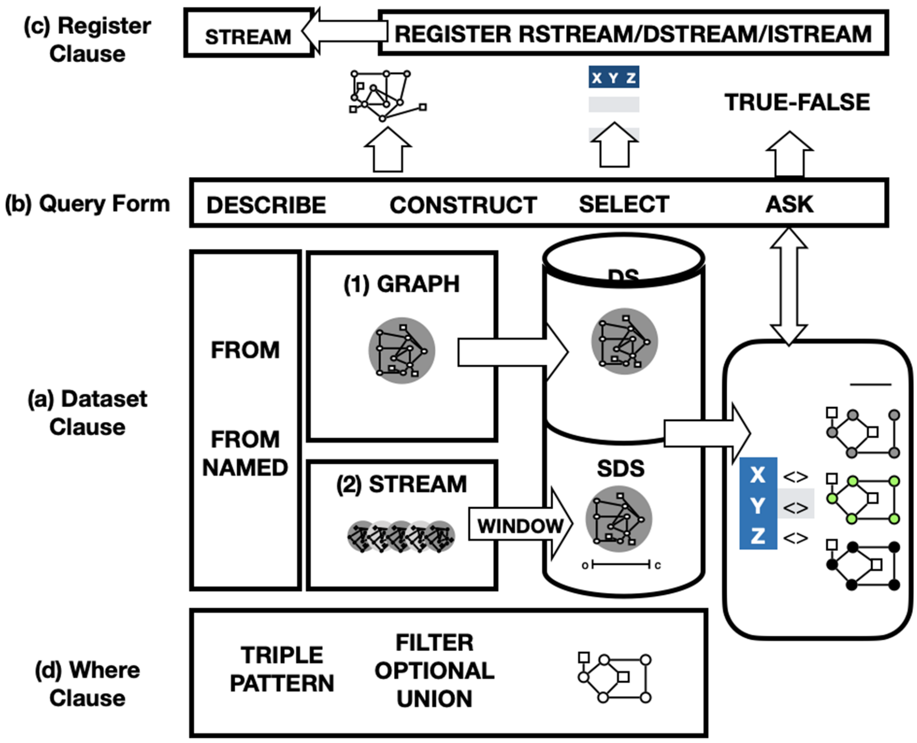


Fig. 10. The anatomy of an RSP-QL query.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
REGISTER RSTREAM <sout> AS
SELECT ?name ?email
FROM <http://www.w3.org/People/Berners-Lee/card>
FROM NAMED WINDOW <w> ON <stream1> [RANGE PT15S STEP PT5S]
WHERE {
    ?person foaf:name ?name ;
            foaf:age ?age .
    FILTER (?age > 18)
    WINDOW <w> { ?person ?p ?o .}
}
ORDER BY ?name
```

Listing 1.7. Example SPARQL query.

Formally speaking, RSP-QL extends CQL for processing RDF Streams. It generalizes the concept of RDF dataset into the idea of a streaming RDF dataset called SDS (cf. Definition 17). It adds operators for continuous processing of RDF

Streams, using Time-Based Sliding Windows (cf. Definition 15), while reusing the R2S operators as defined in CQL.

Definition 15. *A Time-Based Sliding Window $\mathbb{W}$ is an S2R operator that takes as input an RDF stream S (cf. Definition 14) and three more parameters: (i) t_0: a time instant indicating when the operator starts processing; (ii) α: a window width; (iii) β: the window slide. These parameters characterize how the window operator divides the stream. $\mathbb{W}(\alpha, \beta, t_0)$ identifies a set of window intervals $(o, c] \in \mathbb{W}$ such that $o, c \in T$ and are respectively the opening and closing time instants, i.e., $|o_i - c_j| = \alpha$.*

Once applied to an RDF Stream, an RSP-QL time-based window operator produces a *Time-Varying Graph*, i.e., a function that takes a time instant as input and produce an RDF Graph as output.

Definition 16. *A Time-Varying Graph is a function $G_{\mathbb{W}}$ such that*

$$G_{\mathbb{W}}(t) = \{g|(g,t) \in S \wedge (o,c] \in \mathbb{W} \wedge t \in (o,c]\}$$

The domain of the time-varying graph is the set of time instants t where time-based window operator is defined. The co-domain definition is more subtle. Indeed, the window operator chunks the RDF Stream into finite portions each containing a number of RDF Graphs. Thus, as per RSP-QL semantics, the co-domain of the time-varying graph consists of all the RDF graphs resulting from the UNION of each RDF Graph inside a window.

Definition 17. *An RSPQL dataset SDS consists of an (optional) default graph G_{def} and $n (n \geq 0)$ named Time-Varying Graphs resulting from applying Time-Based Sliding Window operators over $m \leq n$ RDF streams.*

Using this operator the RSP-QL users can open a time-based window over an RDF Stream and load the time-varying content into a (named) graph. To represent such concepts, RSP-QL syntax extends the SPARQL Dataset clause with a time-based Window operator. Using the following syntax (for an example see Listing 1.7 at Line 5).

FROM [NAMED] WINDOW [graph IRI] ON (stream IRI) (RANGE,STEP).

Finally, let's briefly discuss how RSP-QL extends the SPARQL evaluation semantics. To this aim, we define an RSP-QL query as follows:

Definition 18. *An RSP-QL query Q is defined as (SE, SDS, ET, QF) where:*

- *SE is an RSP-QL algebraic expression*
- *SDS is an RSP-QL dataset*
- *ET is a sequence of time instants on which the evaluation of the query occurs*
- *QF is the Query Form like in SPARQL (e.g. Select or Construct)*

Intuitively, the evaluation semantics of an RSP-QL query depends on time and is defined as

$$eval(SDS(G), SE, t)\ \forall t \in ET$$

where, SDS is a streaming dataset having G as active Time-Varying Graph, SE is an algebraic expression and t is a time instant.

The evaluation is equivalent to the SPARQL evaluation computed over the instantaneous graphs $G(t)$, i.e.,

$$eval_{SPARQL}(SDS(G(t)), SE)\ \forall t \in ET$$

The set ET of evaluation time instants is determined by the reporting policy adopted by the engine executing the query. This is known as execution semantics [14] and can in RSP-QL be described in terms 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.

4.2 Putting RSP-QL into Practice

In the following, we present a comprehensive set of stream reasoning tasks and their implementation using RSP-QL as a query language for Web stream analytics. For the sake of clarity, we continue working with our running example on the color stream.

Stream Filtering. The aim of this stream reasoning task is identifying the sub-portion of the input stream that is relevant for the analysis.

Example 9 (cont'd). The previously registered color stream can be queried to retrieve the number of blue boxes in the last minute. This requires the RSP to ingest the data stream and to register a continuous query. Listing 1.8 shows the corresponding RSP-QL query.

```
PREFIX color: <http://linkeddata.stream/ontologies/colors#>
PREFIX :      <http://linkeddata.stream/resource/>
SELECT (COUNT(?b) as ?numBlues)
FROM NAMED WINDOW <bw> ON :colorstream [RANGE PT15S STEP PT5S]
WHERE {
    WINDOW ?bw { ?b a color:Blue .}
}
```

Listing 1.8. Counting Blue Occurrences

Stream Enrichment. The aim of this stream reasoning task is joining the streaming data with contextual static knowledge to the extent of enriching the input stream and raising the level of analysis.

Example 10 (cont'd). Exploiting the sentiment ontology, we enrich the colorstream relating each color with the corresponding sentiment. Then, we count the number of color occurrences that are related to positive sentiments. The RSP-QL query is shown in Listing 1.9.

```
PREFIX color: <http://linkeddata.stream/ontologies/colors#>
PREFIX sentiment: <http://linkeddata.stream/ontologies/sentiment#>
PREFIX :      <http://linkeddata.stream/resource/> .
SELECT COUNT(?p) as ?pos
FROM NAMED <sentiment.rdf>
FROM NAMED WINDOW <rw>  ON :colorstream [RANGE PT30S STEP PT5S]
WHERE {
    GRAPH <sentiment.rdf> {
        ?c sentiment:relates [ ?s a sentiment:Positive ] .
    }
    WINDOW <rw> {
        ?p a [ ?c rdfs:subClassOf color:Color .]
        }
}
```

Listing 1.9. Enriching the color stream with sentiment.

Stream Crossing. The aim of this stream reasoning task is to perform analysis across two distinct RDF streams. Stream-to-Stream joining typically requires expressing window operators that provide the scope of the join execution. RSP-QL allows (i) To define different named windows over multiple stream or (ii) To define commonly named windows over different streams.

Example 11 (cont'd). Exploiting colorstream, a new stream can be built, creating new green occurrences every time yellow and blue occurs in a defined time interval. Listing 1.10 shows the corresponding RSP-QL query.

```
PREFIX color: <http://linkeddata.stream/ontologies/colors#>
PREFIX :      <http://linkeddata.stream/resource/> .
CONSTRUCT { ?g a color:Green ; color:from ?b, ?y . }
FROM NAMED WINDOW <bw> ON :bluestream [RANGE PT30S STEP PT5S]
FROM NAMED WINDOW <yw> ON :yellowstream [RANGE PT15S STEP PT5S]
WHERE {
    WINDOW <bw> { ?b a color:Blue .}
    WINDOW <yw> { ?y a color:Yellow .}
    BIND( UUID() as ?g )
}
```

Listing 1.10. Cooking Green streams.

```
PREFIX color: <http://linkeddata.stream/ontologies/colors#> .
PREFIX :      <http://linkeddata.stream/resource/> .
SELECT (COUNT(?w) AS ?cntWarm) (COUNT(?c) AS ?cntCold) ((?cntWarm>?
    cntCold) as ?moreWarmThanCold)
FROM NAMED WINDOW <cw> ON :colorstream [RANGE PT15S STEP PT5S]
WHERE {
    { WINDOW <cw> { ?w a color:Warm. } }
    UNION
    { WINDOW <cw> { ?c a color:Cold .} }
}
```

Listing 1.11. Comparing warm and cold colors.

Stream Abstraction. The aim of this stream reasoning task is leveraging background domain knowledge to raise the analysis to a higher level of abstraction. To this extent, the query answering relies on deductive reasoning.

Example 12 (cont'd). Reasoning capabilities can be exploited to continuously compare the number of cold colors versus the number of warm colors. Listing 1.11 shows the corresponding RSP-QL query.

The queries above, written in RSP-QL syntax, can be executed through YASPER [30], an RSP engine that implements RSP-QL semantics. YASPER is not just a reference implementation of RSP-QL but also a library for rapid prototyping of RSP Engines[9]. It includes different runtimes that allow for a systematic and comparative analysis of RSP-QL performance. In particular, YASPER builds on the lesson learned by realizing several working prototypes like C-SPARQL engine [3], CQELS engine [24], and Morph$_{Stream}$ [6].

5 Exercise - Linear Pizza Oven

Consider the scenario depicted in Fig. 11. The company DreamPizza bought an oven with the aim of increasing their pizza's quality. The oven is a smart one. It makes use of a conveyor belt that takes pizzas in and out of the oven. Moreover, it has several sensors that continuously produce data. Carl, a data scientist working for DreamPizza, is particularly focused on two sensors:

– **S1.** The sensor S1 is positioned at the entrance of the oven. It has a camera that, using image recognition algorithms, is capable of detecting pizza toppings. Since it is the first sensor to analyze a pizza, it also assigns to each pizza a unique id $< pid >$. The sensor S1 exploits the Pizza Ontology[10] to produce an RDF stream. An example of such a stream is shown in Listing 1.12.

[9] Find the most recent version at https://github.com/riccardotommasini/yasper.
[10] https://protege.stanford.edu/ontologies/pizza/pizza.owl.

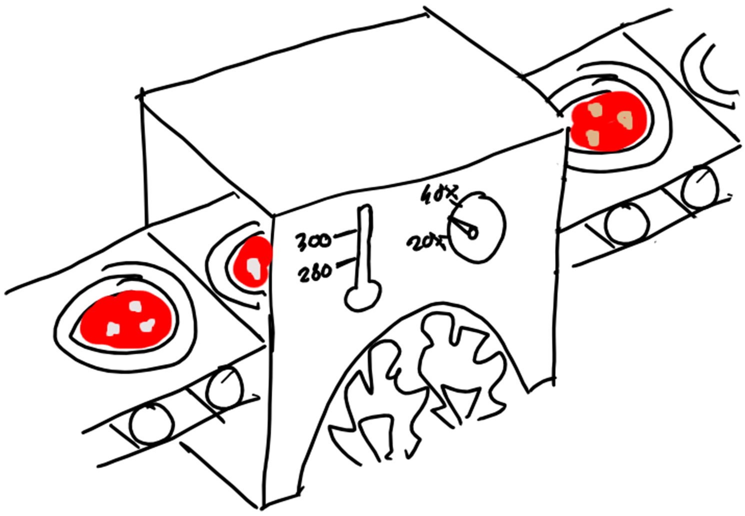


Fig. 11. Linear pizza oven.

– **S2.** The sensor S2 is positioned inside the oven. It senses the temperature and, exploiting the Sensor-Observation-Sampling-Actuator ontology (SOSA[11]), outputs an RDF stream with the $< pid >$ of the pizza and the oven's temperature. Multiple measures for the same $< pid >$ are provided, denoting the temperatures of the oven during the cooking process. An example of the RDF stream produced by S2 is shown in Listing 1.13.

```
@base <http://linkeddata.stream/streams/pizza-S1/1588684149> .
@prefix : <http://www.co-ode.org/ontologies/pizza/pizza.owl#> .

<#507f704d-dfb8-4fa3-97ca-64a93e56abb0> a :Pizza ;
                                        :hasTopping :MozzarellaTopping
                                            ,
                                                    :TomatoTopping .
```

Listing 1.12. The RDF stream produced by S1.

Observing the RDF streams from S1 and S2, Carl is able to observe, for each pizza, the toppings and the temperature of the oven during the cooking process. Carl wonders if it is possible to exploit such data to automatically detect the name of each pizza starting from its toppings and, analyzing the oven temperature during its cooking process, assert on the quality of such pizza. In other words, he wants to know if a pizza has been cooked correctly or not. A colleague of Carl, Frank, has a relational database containing the cooking temperature for each named pizza. A small dump of the table Cook from such database is shown in Table 2.

[11] https://www.w3.org/TR/vocab-ssn/.

```
@base <http://linkeddata.stream/streams/pizza-S2/1588684151> .
@prefix : <http://linkeddata.stream/streams/pizza-S1/1588684149#> .
@prefix sosa: <http://www.w3.org/ns/sosa/> .
@prefix qudt: <http://qudt.org/1.1/schema/qudt#> .
@prefix qudt-unit: <http://qudt.org/1.1/vocab/unit#> .

<#575d1e00-1b70-4a3b-9ed4-961990aaead8> a sosa:Observation ;
   sosa:hasFeatureOfInterest :507f704d-dfb8-4fa3-97ca-64a93e56abb0 ;
   sosa:hasResult [
      a qudt:QuantityValue ;
      qudt:unit qudt-unit:DegreeCelsius ;
      qudt:numericValue 280.4 ] .
```

Listing 1.13. The RDF stream produced by S2.

Table 2. Table cook from Frank's database.

NamedPizza	TempAvg	TempStd
Margherita	280	10
American	275	5

Carl needs your help to develop and deploy a Stream Reasoning application that, exploiting the data streams from the sensors and the relational database from Frank, is able to detect whether each pizza passing thought the oven is cooked correctly or not.

6 Conclusion

Social networks, sensors for Industry 4.0, smart homes, and many other devices connected to the Internet produce an ever-growing stream of data. In this context, Stream Reasoning applications are needed to integrate data streams with complex background knowledge, while addressing time-specific constraints.

In this lecture, we provide basic knowledge about data streams, continuous processing, RDF and SPARQL. We illustrate the guidelines to develop and deploy Steams Reasoning applications, analyzing the Streaming Linked Data Lifecycle and presenting concrete realizations of each step with respect to a running example based on colors. In particular, we focused on the interoperability between stream services on the Web, giving an introduction to the Vocabulary & Catalog of Linked Streams (VoCaLS), designed with the goal of solving such problems. We later depicted RSP-QL, an extension of SPARQL able to cope with data streams and that, besides introducing new operator classes, it also formalizes the operational semantics. Finally, we presented a full-stack exercise for getting started with Streaming Linked Data and RDF Stream Processing using RSP-QL.

Acknowledgments. Dr. Tommasini acknowledges support from the European Social Fund via IT Academy programme.

References

1. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB J. **15**(2) (2006)
2. Balduini, M., Valle, E.D.: FraPPE: a vocabulary to represent heterogeneous spatio-temporal data to support visual analytics. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 321–328. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_21
3. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: a continuous query language for RDF data streams. Int. J. Semant. Comput. **4**(1), 3–25 (2010)
4. Barbieri, D.F., Della Valle, E.: A proposal for publishing data streams as linked data - a position paper. In: LDOW. CEUR Workshop Proceedings, vol. 628. CEUR-WS.org (2010)
5. Bonte, P., Tommasini, R., Della Valle, E., Turck, F.D., Ongenae, F.: Streaming MASSIF: cascading reasoning for efficient processing of IoT data streams. Sensors **18**(11), 3832 (2018)
6. Calbimonte, J.-P., Mora, J., Corcho, O.: Query rewriting in RDF stream processing. In: Sack, H., Blomqvist, E., d'Aquin, M., Ghidini, C., Ponzetto, S.P., Lange, C. (eds.) ESWC 2016. LNCS, vol. 9678, pp. 486–502. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-34129-3_30
7. Comton, M., et al.: The SSN ontology of the W3C semantic sensor network incubator group. J. Web Semant. **17**, 25–32 (2012)
8. Della Valle, E., Ceri, S., van Harmelen, F., Fensel, D.: It's a streaming world! reasoning upon rapidly changing information. IEEE Intell. Syst. **24**(6), 83–89 (2009)
9. Della Valle, E., Dell'Aglio, D., Margara, A.: Taming velocity and variety simultaneously in big data with stream reasoning: tutorial. In: DEBS, pp. 394–401. ACM (2016)
10. Dell'Aglio, D., Dao-Tran, M., Calbimonte, J.-P., Le Phuoc, D., Della Valle, E.: A query model to capture event pattern matching in RDF stream processing query languages. In: Blomqvist, E., Ciancarini, P., Poggi, F., Vitali, F. (eds.) EKAW 2016. LNCS (LNAI), vol. 10024, pp. 145–162. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49004-5_10
11. 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. Semant. Web Inf. Syst. **10**(4), 17–44 (2014)
12. Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: a survey and outlook. Data Sci. **1**(1–2), 59–83 (2017)
13. Dell'Aglio, D., Le Phuoc, D., Le-Tuan, A., Ali, M.I., Calbimonte, J.P.: On a web of data streams. In: ISWC DeSemWeb (2017)
14. Dindar, N., Tatbul, N., Miller, R.J., Haas, L.M., Botan, I.: Modeling the execution semantics of stream processing engines with SECRET. VLDB J. **22**(4), 421–446 (2013)
15. Gao, F., Ali, M.I., Mileo, A.: Semantic discovery and integration of urban data streams. In: Proceedings of the Fifth Workshop on Semantics for Smarter Cities a Workshop at the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, October 19, 2014, pp. 15–30 (2014). http://ceur-ws.org/Vol-1280/paper5.pdf

16. Janowicz, K., Haller, A., Cox, S.J.D., Phuoc, D.L., Lefrançois, M.: SOSA: a lightweight ontology for sensors, observations, samples, and actuators. J. Web Semant. **56**, 1–10 (2019)
17. Laney, D.: 3d data management: controlling data volume, velocity and variety. META Group Res. Note **6**(70), 1 (2001)
18. Margara, A., Urbani, J., van Harmelen, F., Bal, H.E.: Streaming the web: reasoning over dynamic data. J. Web Semant. **25**, 24–44 (2014)
19. Mauri, A., et al.: Triplewave: spreading RDF streams on the web. In: ISWC (2016)
20. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: RDFox: a highly-scalable RDF store. In: Arena, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 3–20. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_1
21. Passant, A., Bojārs, U., Breslin, J.G., Decker, S.: The SIOC project: semantically-interlinked online communities, from humans to machines. In: Padget, J., et al. (eds.) COIN -2009. LNCS (LNAI), vol. 6069, pp. 179–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14962-7_12
22. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (2009)
23. Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 370–388. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_24
24. Phuoc, D.L., Dao-Tran, M., Tuán, A.L., Duc, M.N., Hauswirth, M.: RDF stream processing with CQELS framework for real-time analysis. In: DEBS (2015)
25. Sequeda, J.F., Corcho, Ó.: Linked stream data: a position paper. In: SSN. CEUR Workshop Proceedings, vol. 522, pp. 148–157. CEUR-WS.org (2009)
26. Terry, D.B., Goldberg, D., Nichols, D.A., Oki, B.M.: Continuous queries over append-only databases. In: Stonebraker, M. (ed.) Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, 2–5 June 1992, pp. 321–330. ACM Press (1992). https://doi.org/10.1145/130283.130333
27. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: tractable OWL 2 reasoning infrastructure. In: Aroyo, L., et al. (eds.) ESWC 2010. LNCS, vol. 6089, pp. 431–435. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13489-0_38
28. Tommasini, R.: Velocity on the web - a PhD symposium. In: WWW (Companion Volume), pp. 56–60. ACM (2019)
29. Tommasini, R., Bonte, P., Della Valle, E., Ongenae, F., De Turck, F.: A query model for ontology-based event processing over RDF streams. In: Faron Zucker, C., Ghidini, C., Napoli, A., Toussaint, Y. (eds.) EKAW 2018. LNCS (LNAI), vol. 11313, pp. 439–453. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03667-6_28
30. Tommasini, R., Della Valle, E.: Yasper 1.0: towards an RSP-QL engine. In: Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks Co-located with 16th International Semantic Web Conference (ISWC) (2017)
31. Tommasini, R., et al.: VoCaLS: vocabulary and catalog of linked streams. In: Vrandečić, G., et al. (eds.) ISWC 2018. LNCS, vol. 11137, pp. 256–272. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00668-6_16
32. Wood, D., Lanthaler, M., Cyganiak, R.: RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/