Applying Semantic Interoperabiltiy Principles to Data Stream Management

Daniele Dell'Aglio, Marco Balduini, Emanuele Della Valle

1 Introduction

The cost vs. opportunity trade-off in ICT projects often pushes separate organisations and, even, departments of the some organisation to collect and manage data independently. This creates the so called *data silos*. However, early success of those ICT projects commonly results in requests for cross-silos usages of data and, therefore, for addressing the data integration problem [29]. Systems addressing this problem have to bridge the silos and produce a uniform query interface.

Data streams do not make an exception. Consider, for instance, the case of a large cultural heritage site where two successful ICT projects deployed Near Field Communication (NFC) sensors [32] to let the guide signal where they are and a Quick Response (QR) code[1] based system that tourists can use to get information about a part of the exposition. Add to this scenario the common habit of tourists to share on social network their impressions about the visiting experience. Those three streaming data sources are data silos. They serve the purpose of the application they were built for. However, it is easy to image the value of cross-silos usage of those data streams. For instance, it would be possible to recommend to a free guide to propose an attractive short guided tour to the tourists nearby based on the preferences they expressed by using the QR codes and sharing bits of their visits on social networks.

The problem of data integration has been studied for decades, but the experimental nature of the data acquisition solutions, which produce data streams, together with the poor attention to this type of data, which is often to consider too low level to be useful before aggregation, exacerbate the problem and requires a new generation of data integration solution tailored to data streams.

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico of Milano.
P.za L. Da Vinci, 32. I-20133 Milano – Italy
e-mail: {daniele.dellaglio},{marco.balduini},{emanuele.dellavalle}@polimi.it

[1] BS ISO/IEC 18004:2006. Information technology. Automatic identification and data capture techniques. Bar code symbology. QR Code. Geneva: ISO/IEC. 2000. p. 114.

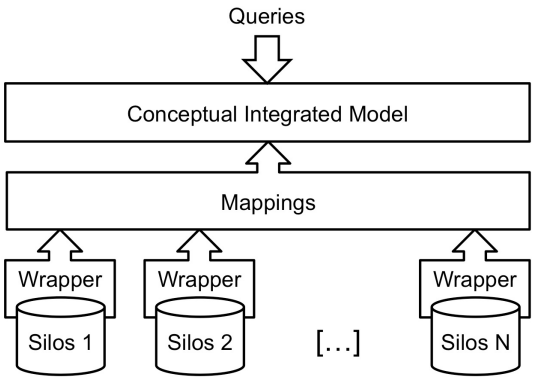

Fig. 1 The Architecture of Data Integration Systems.

Most data integration systems adopt the architecture outlined in Figure 1. Data in the silos can be organised in relational, XML or graph databases and in some case even in textual and multimedia repositories. In two separate silos the same information can appear with different syntaxes, data structures and conceptual models, raising respectively the syntactic, structural and semantic heterogeneity problem. For example, the notion of user is likely to be different for the QR code based system and a social network. The different business and legal requirements will influence their respective data representations. For example, a social network is unlikely to keep a record of what pieces in the site its users asked information about, while the QR code based system is unlikely to keep track of what impressions its users shared with their friends.

An **Integrated Conceptual Model** (ICM) is required to bridge the syntactic, structural and semantic heterogeneities of the data in the silos. In our example, an ICM of the user would consider all information about users available in the various silos. The problem of representing and querying incomplete information is extremely relevant in this context. Consider, for instance, a pieces in the site. It is explicitly named, when a user asks its description using the QR code-based application. On the contrary, if the a user checks-in the site using a social network, it is only possible to infer that the user has seen some pieces of the site without knowing exactly which one. In order to develop the ICM, an adequate data modelling language is needed. In literature, a variety of languages were proposed. They differ for their ability to capture relationship between terms used to describe the data in the silos. In the last decade, ontological languages like OWL [30] and OWL 2 [31] were often adopted for this purpose. They can adequately master the semantic heterogeneity between the silos.

The ICM defines a vocabulary to issue (or register in the streaming setting) queries across the silos, but in order to get results we have to link the terms describing the data in the silos to those in the ICM. Expressions to model those links are named **mappings** and they can assume multiple forms (i.e. Local-as-View, Global-

as-View and both [29]). Mappings only masters structural heterogeneity assuming the silos and the ICM to be requested in the same data model. This is often not the case, so a **wrapping** solution is needed to bridge the syntactic heterogeneity between the data in the silos (e.g. relational) and the one data model assumed by the mapping language. In some cases, hybrid solutions that wrap data in silos and map it to the ICM exists, e.g. to map a given data model (e.g. relational) into the data model of the ICM. In recent years, we observed a growing adoption of the Resource Description Format (RDF) [26] as data model of the ICM because it is adequate to represent data when the ICM is modelled in OWL or OWL 2. When those technologies are adopted and the data in the silos is relational, R2RML [19] is the associate mapping language to be used and SPARQL [35] (or its extensions to data streams [9]) is the language to declare queries with. Dialects of R2RML exist to map between data models that resemble the relational one, e.g. when data are stored in CSV files or are XML or JSON results of Web Service invocations. The Any23 solution[2] is a good example of flexible open source framework to wrap any type of data source as RDF.

The semantics of the data integration solution as a whole – i.e., to define which are the correct answers to a query on top of a given set of data in the silos – is determined by the semantics of the languages used to model the ICM, the mappings, the wrappers, the one of the data model to represent data in the ICM and the one of the query language used to declared the queries. Given the complexity of the task of defining the semantics for a given data integration solution as a whole, readers should not be surprised that current literature lacks a uniform and well-accepted theory that describes the various parts of data integration systems. Only a consistent theory for a particular data integration solution may be found.

This chapter considers only two extreme solutions: data-driven and query-driven. In data-driven solutions the ICM is populated with data before starting to answer a query declared on it. Solutions of this kind range from Materialised Views in databases [24] up to consequence-based reasoning [25, 39]. The limit of data-driven solutions is the need to duplicate in the ICM all the data in the silos even if only a limited fraction is required to answer a query. On the contrary, in the query-driven solutions a query expressed on the ICM is translated into a set of queries expressed in the query language (if any) of the silos [16, 23]. The queries are then evaluated in the silos, and the results are integrated into an answer to the original query. Unfortunately the query-driven approach is not always possible and depends on the ontological language used to model the ICM. In recent years, both approaches were explored to integrate data streams (see, for instance, [8] for a data-driven approach and [14] for a query-driven one). They are commonly referred as embodiments of the Stream Reasoning vision [20] and consensus is growing[3] around the idea to name them RDF Stream Processing (RSP) systems. It is worth to note that porting solutions known to work in the static setting to the streaming one is far from being trivial. Data-driven solutions are at risk of being impractical due to the very un-

[2] http://any23.apache.org/

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

bound nature of data streams while query-driven ones require more investigation to master the operation semantics heterogeneity that exists in data stream management systems [12].

The remainder of the chapter is organised as follows. Section 2 formalises the notion of RDF stream and the semantics of continuous query processing of RDF streams. Section 3 briefly surveys existing implementations. Section 4 assembles all elements introduced in the previous sections and puts them at work on a case study. Finally Section 5 concludes and casts some light on future developments of the field.

2 RSP models

The main feature of RDF Stream Processing is its data model: instead of traditional relational data, it uses the RDF data model. The rest of the section first introduces the RDF data model, then presents the different data and query models proposed in the literature.

2.1 The RDF data model

RDF is a W3C recommendation for data interchange on the Web [26]. RDF data are structured as directed labeled graphs, where the nodes are **resources**, and the edges represent relations among them. Each node of the graph is an **RDF term**: it can be a named **resource** (identified by an IRI), an anonymous resource (also known as **blank node**) or a **literal** (a number, a string, a date, etc.). Let us denote with I, B, and L the set of resources, blank nodes and litereals. The basic building block of RDF is the **RDF statement**, a triple $(s, p, o) \in (I \cup B) \times (I) \times (I \cup B \cup L)$. A set of RDF statements is an **RDF graph**. Let us consider the following chunck of RDF data, extracted by DBPedia[4]:

```
1  @prefix dbpedia: <http://dbpedia.org/resource/>
2  @prefix dbpprop: <http://dbpedia.org/property/>
3
4  dbpedia:Mona_Lisa  dbpprop:title    "Mona Lisa"@en .
5  dbpedia:Mona_Lisa  dbpprop:title    "La Gioconda"@it .
6  dbpedia:Mona_Lisa  dbpprop:artist   dbpedia:Leonardo_da_Vinci .
7  dbpedia:Mona_Lisa  dbpprop:museum   dbpedia:The_Louvre .
8  dbpedia:The_Louvre dbpprop:lat      "48.860339"^^xsd:double ;
9                     dbpprop:lng      "2.337599"^^xsd:double ;
10                    dbpprop:director dbpedia:Jean-Luc_Martinez .
```

[4] Cf. http://dbpedia.org.

This syntactic serialisation of RDF is named Turtle[5]. The first two lines are the **prefixes**: they can be declared to use a clearer representation of the data, e.g. given the prefix in Line 1, `dbpedia:Mona_Lisa` is a short version of the IRI `http://dbpedia.org/resource/Mona_Lisa`. The second block (Lines 4-10) contains the RDF statements: each of them is composed of three components, respectively the subject, the predicate and the object. The statements in the example describe the Mona Lisa: named Mona Lisa in English (Line 4), and La Gioconda in Italian (Line 5), it is a work of Leonardo Da Vinci (Line 6) and it is located at the Louvre (Line 7), a museum located at the coordinates $\langle 48.860339, 2.337599 \rangle$ (Lines 8 and 9) directed by Jean-Luc Martinez (Line 10). Notably, Lines 4-7 clearly show the triples separated by the character "`.`", whereas Lines 8-10 shows a convenience syntax that allows to separate with the character "`;`" pairs of property object sharing the same subject.

The following sections illustrate how the RDF data model has been used in the definition of data models for RSP, and how it can be processed by the engines. These analyses are used in Section 3 to present and classify the existing RSP solutions.

2.2 The RSP data model

In the previous chapters, the notion of data stream was introduced. We learnt that Data Stream Management System (DSMS) (Chapter 5 [38]) and Complex Event Processing (CEP) (Chapter 6 [18]) systems work on relational data streams. This section, to cope with the data integration problem, presents the **RDF data streams** (or RDF streams), introducing the existing definitions proposed in the previous years. The definition of RDF stream is built from the one of relational data stream. RDF streams are sequences of timestamped data items, where a data item is a self-consumable informative unit; moreover, the main characteristics of relational data streams hold [5]:

- They are **continuous**: new data items are continuously added to the stream;
- They are potentially **unbounded**: a data stream could be infinite;
- They are **transient**: it is not always possible to store data streams in secondary memory;
- They are **ordered**: data items are intrinsically characterised by recency. The order is partial, i.e. two data items can share the same temporal annotation (contemporaneity)

Different definitions of RDF streams have been proposed by RSP research. It is useful to introduce two axes to classify them: the data item (Section 2.2.1) and the time annotation (Section 2.2.2).

[5] Cf. http://www.w3.org/TR/turtle/.

2.2.1 Data item dimension

The data item is the minimal informative unit. Existing works in RSP consider two alternatives for this role: RDF statements – triples of RDF resources, and RDF graphs – set of RDF statements.

The simplest case is the one where the stream is composed of **RDF statements**: each data item is composed by a sequence of three RDF resources: subject, predicate and object. For example, let us consider this RDF stream on which each triple states the presence of a person in a room of The Louvre:

```
:alice  :detectedAt :monaLisaRoom  [1] .
:bob    :detectedAt :parthenonRoom [2] .
:conan  :detectedAt :monaLisaRoom  [5] .
:bob    :detectedAt :monaLisaRoom  [5] .
```

Turtle-like syntax is adopted: in each row there is an RDF statement enriched with the time annotation (the fourth element between square brackets). When adopting this data model, each statement may contain enough information to be processed as informative unit.

Even if the RDF statement stream model is easy to be managed, the amount of information that a single RDF statement carries may be not enough when modelling real use cases. For this reason, recent works (e.g. [7]) propose to use as informative unit **RDF graphs**, i.e. set of RDF statements. Let us consider the following RDF stream, expressing check-in operations in a social network:

```
:g1 [1]
  {
    :alice :posts :c1 .
    :c1    :where :monaLisaRoom .
    :c1    :with  :conan .
  }

:g2 [3]
  {
    :bob   :posts :c2 .
    :c2    :where :parthenonRoom .
    :c2    :with  :diana .
  }

:g3 [3]
  {
    :conan :posts :c3 .
    :c3    :where :monaLisaRoom .
  }
```

This example adopts a Trig-like syntax[6]: RDF resources `:g1`, `:g2` and `:g3` identify three RDF graphs followed by the relative time annotation (the number between squared brackets). The blocks of RDF statements (enclosed in {}) are the contents

[6] Cf. http://www.w3.org/TR/trig/.

of the graphs. As it is possible to observe, in this example single RDF statements are not enough to represent a whole informative unit (e.g. a check-in post).

2.2.2 Time annotation dimension

The time annotation is a set of time instants associated with each data item. The choice to consider the time as an annotation on data items, and not as part of the schema, is inherited by the DSMS research [5], and it is motivated by both modelling and technical reasons: the time information could be part of the schema, but it should not be mandatory (i.e. there are scenarios on which it is not). Moreover, DSMSs and CEPs usually do not allow explicit accesses to the time annotations through the query languages: conditions are usually expressed with time relative constraints, e.g. select events that *happen before a given one*, or identify the events that *hold in the last five minutes*.

The term **application time** refers to the time annotation of a data item [12]. Usually, application time is represented through sets of timestamps, i.e. identifier of relevant time instants. The classification along the time annotation axis depends on the number of timestamps that compose the application time.

In the simplest case, the application time consists of **zero timestamps**: in other words, there is not explicit time information associated with the data items. It follows that the RDF stream is an ordered sequence of elements that arrive to the processing engine over time, like in the stream S represented in the Figure 2.

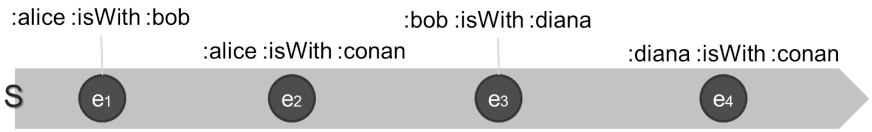

Fig. 2 Example of stream with zero timestamps per data item.

Rounds labelled with e_i (i in $[1,4]$) represent the data items of the stream; the time flows from left to right, e.g. item e_1 happens before item e_2. Even if the data items do not have explicit timestamps, it is possible to process those streams by defining queries that exploit the order of the elements, such as:

q_1. Does Alice meet Bob before Conan?
q_2. Who does Conan meet first?

Let us consider now the case on which the application time is modelled introducing a metric [37] and each data item has **one timestamp** like in Figure 3.

In most of the existing works, the timestamp used in the application time represents the time instant at which the associated event occurs, but other semantics are possible, e.g. time since the event holds. Due to the fact that data items are still

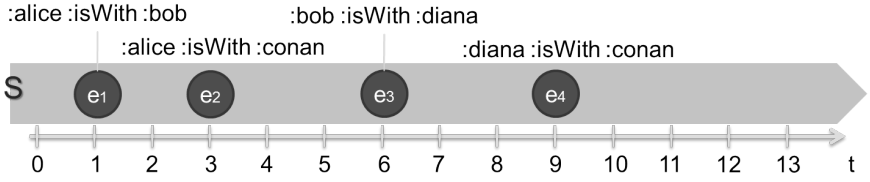

Fig. 3 Example of stream with one timestamp per data item.

ordered by recency (as in the previous case), it is possible to issue queries of the previous case, as q_1 and q_2. Additionally, it is possible to write queries that take into account the time, such as:

q_3. Does Diana meet Bob and then Conan within 5m?
q_4. How many people has Alice met in the last 5m?

It is worth to note that q_3 and q_4 do not refer to absolute time instants, but on relative ones w.r.t the time instant of another event (as in q_3) or the current time instant (as in q_4).

As a final case, let us introduce the application time composed of **two timestamps**. The semantics that is usually associated to the two timestamps is the time range $(from, to]$ on which the data item is valid, as shown in Figure 4.

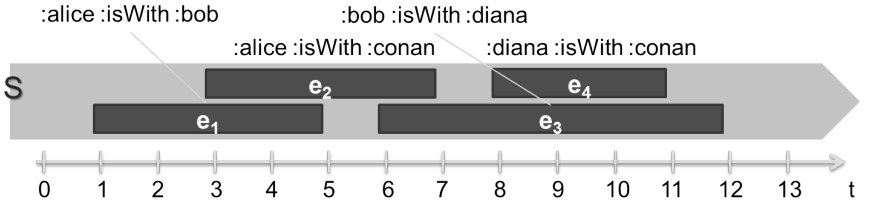

Fig. 4 Example of stream with two timestamps per data item.

Each square represents a data item, and the application time is represented by the initial and the final timestamps, e.g. e_1 has application time $(1,5)$, so it is valid from the time instants 1 to the time instants 5. Similarly to the previous case, it is still possible to process to the queries presented in the first two cases (e.g. $q_1, \ldots, q_4$), and additionally more complex constraints can now be written, such as:

q_5. Which are the meetings that last less than 5m?
q_6. Which are the meetings with conflicts?

Other cases exist, where the application time is composed of three or more timestamps, or where the application time semantics has other meanings. Even if they can

be useful in some use cases, they are still under investigation in RSP research and no relevant results have been reached, yet.

2.3 RSP query model

After the presentation of the RDF stream definitions, this section discusses the problem of processing those kind of data. As for the data model, the RSP query model is inspired by research in DSMS and CEP: existing works adapt the existing operators of those disciplines to process RDF data streams.

2.3.1 The CQL Model and its RDF stream adaptation

In Chapter 5 [38], we learnt that the CQL stream processing model (proposed by the DB group of the Stanford University [4]) defines a generic DSMS through three classes of operators (Figure 5.a).

The first class of operators manages the data stream: due to the fact that a stream is a potentially infinite bag of timestamped data items, those operators extract finite bags of data. CQL defines as **stream-to-relation** the operators able to transform streams in relations; among the available operators of this class, one of the most studied is the sliding window. Next, the relation may be transformed in another relation through a **relation-to-relation** operator. Relational algebraic expressions are a well-known cases of this class of operators. Finally, the transformed relation has to be set as output. There are usually two ways to set the answer: as a time-varying relation, or as a part of a stream. In the second case, an additional operator is required: the **relation-to-stream**.

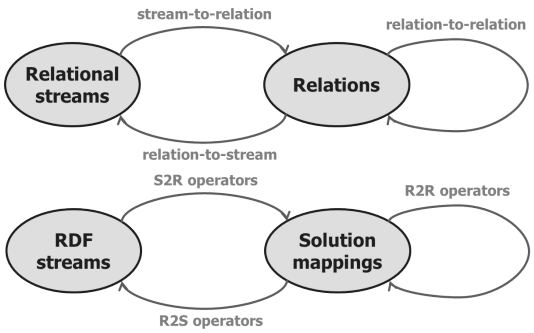

Fig. 5 The CQL model and its adaptation for RDF Stream Processing

As already mentioned in Chapter 5 [38], the CQL model inspired the design of different RDF Stream Processing engines [9, 27, 14], and currently there are different implementations (e.g. C-SPARQL, SPARQLstream, CQELS) of the adapted model, represented in Figure 5.b.

The stream and the relation concepts are mapped to RDF streams and to set of mappings (using the SPARQL algebra terminology[7]), respectively. To highlight the similarity of the RSP operators to the CQL ones, similar names are used: **S2R**, **R2R** and **R2S** to indicate the operators respectively analogous to stream-to-relation, relation-to-relation and relation-to-stream operators. The following sections describe the sliding window as example of S2R operators, SPARQL algebra as example of R2R operators, and the streaming operator as instance of R2S operators.

2.3.2 The sliding windows

Given a stream S, a sliding window dynamically selects subsets of the data items of S. The intuition behind this operator is that the most recent data are the most relevant, so it selects them from the stream and queries it, repeating this operations as the time goes ahead.

The basic elements of the sliding windows are the **windows**. Given a stream S, a **time-based window** W defined through two parameters o and c selects the data item d of S with associated timestamp in $(o,c]$. Given the stream in Figure 6, the window defined between the time instants 2 (included) and 5 (excluded) selects the elements e_3, e_4 and e_5.

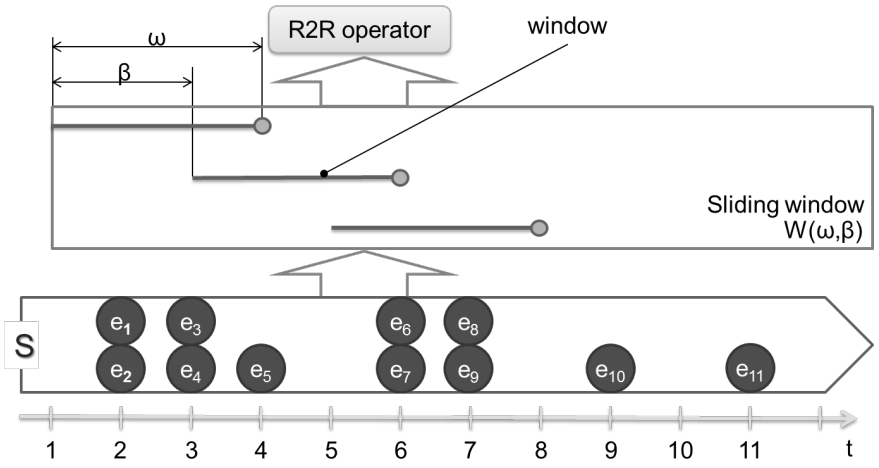

Fig. 6 Time-based sliding window

[7] Cf. http://www.w3.org/TR/sparql11-query/.

A sliding window generates multiple windows (at different time instants) to create a time-varying view over the stream. Two of the most famous kinds of sliding windows are the **time-based sliding windows** and the **triple-based sliding windows**. A time-based sliding window generates a sequence of windows at regular time intervals (e.g. a window each 2 seconds). Then, it selects the contents of the streams accordingly to each window $(o,c]$ (where o and c are the opening and the closing time instants). The time range changes periodically, modifying the content of the sliding window. A time-based sliding window is described through two parameters, the **width** ω (the dimension of the window, $c-o$) and the **slide** β (the distance between two consecutive windows).

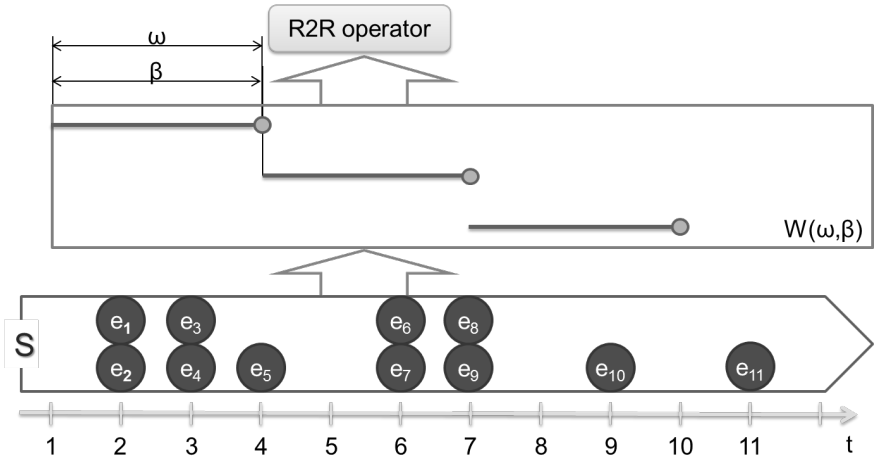

Fig. 7 Time-based tumbling window

Let us consider the example in Figure 6: it shows a time-based window W with width $\omega = 3$ and slide $\beta = 2$. The first window selects all the items with timestamp in (1,4], the second one selects the items with timestamps in (3,6], the third one selects the items with timestamps in (5,8]. As said above, the width of the window represented in the picture is 3 (4-1=6-3=8-5), while the slide parameter, defined as the distance between two consecutive windows, is 2 (3-1=5-3). When the slide parameter is equal to the width parameter, the sliding-window is also known as **tumbling window**, as depicted in Figure 7. This kind of sliding window is important because it partitions the stream: each data item would only in one window.

Tuple-based sliding windows select a fixed number of data items (Figure 8). Similarly to time-based sliding windows, they are described by the width and the slide parameters, but the width indicates the number of RDF statements that are collected in the current view, while the slide indicates how many RDF statements are removed/added at each window slide. Consequently, the difference between the clos-

ing time instant and the opening time instant of the generated windows is not constant: as it is possible to observe in the picture, windows can have different lengths.

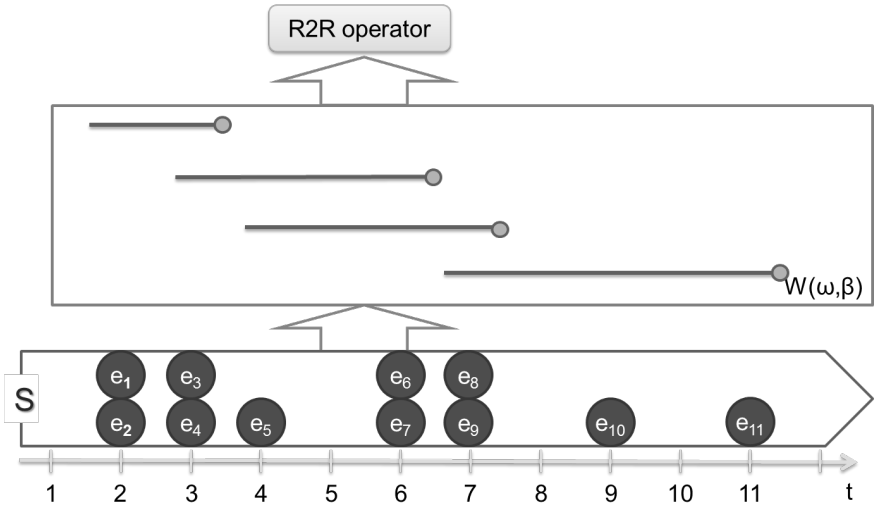

Fig. 8 Tuple-based tumbling window

2.3.3 The R2R operator

SPARQL is the query language of RDF [35]: it allows to query RDF models and to perform relation transformations. In the following, the main concepts of the SPARQL algebra [33] are briefly introduced, focusing on the `WHERE` clause, the clause that contains the criteria of selection of the data (similarly to the `WHERE` clause of SQL). In SPARQL, the `WHERE` clause contains a set of **graph pattern** expressions that can be constructed using the operators `OPTIONAL`, `UNION`, `FILTER` and concatenated via a point symbol " `.` " that means `AND`. Let us extends the sets of symbols introduced for RDF (i.e. the pairwise disjoint sets I (IRIs), B (Blank nodes), and L (literals)) with the set V (variables). A **graph pattern expression** is defined recursively as:

1. A tuple from $(I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$ is a graph pattern and in particular it is a triple pattern.
2. If P_1 and P_2 are graph patterns, then $(P_1 \text{ . } P_2)$, $(P_1 \text{ OPTIONAL } P_2)$ and $(P_1 \text{ UNION } P_2)$ are graph patterns.
3. If P is a graph pattern and R is a SPARQL built-in condition, then $(P \text{ FILTER } R)$ is a graph pattern.

A SPARQL built-in condition is composed of elements of the set $I \cup L \cup V$ and constants, logical connectives ($\neg$, $\wedge$, $\vee$), ordering symbols ($<, \leq, \geq, >$), the equality symbol (=), unary predicates like `bound`, `isBlank`, `isIRI` and other features. An important case of graph pattern expression is the **Basic Graph Pattern** (BGP): it is defined as a set of triple patterns and `FILTER` clauses that are connected by the "`.`" (i.e. the `AND`) operator.

The semantics of SPARQL queries uses as basic building block the **solution mapping**: let P be a graph pattern, *var(P)* denotes the set of variables occurring in P. A **solution mapping** μ is a partial function $\mu : V \rightarrow (I \cup L \cup B)$. The domain of μ, denoted by *dom*(μ), is the subset of V where μ is defined.

The relation between solution mappings, triple patterns and basic graph patterns is given in the following definition: given a triple pattern t and a solution mapping μ such that $var(t) \subseteq dom(\mu)$, $\mu(t)$ is the triple obtained by replacing the variables in t according to μ. Given a basic graph pattern B and a solution mapping μ such that $var(B) \subseteq dom(\mu)$, we have that $\mu(B) = \cup_{t \in B} \mu(t)$, i.e. $\mu(B)$ is the set of triples obtained by replacing the variables in the triples of B according to μ.

Using these definitions, [33] defines the semantics of SPARQL queries as an algebra. The main algebra operators are Join ($\bowtie$), Union ($\cup$), Difference($\setminus$) and Left Join ($\mathbin{⟕}$). The authors define the semantics of these operators on **sets of solution mappings** denoted with Ω. The evaluation of a SPARQL query is based on its translation into an algebraic tree composed of those algebraic operators.

The simplest case is the **evaluation of a basic graph pattern**: let G be an RDF graph over $I \cup L$ and P a Basic Graph Pattern. The evaluation of P over G, denoted by $[\![P]\!]_G$, is defined by the set of solution mappings:

$$[\![\mathrm{P}]\!]_G = \{\mu \mid dom(\mu) = var(P) \text{ and } \mu(P) \subseteq G\}$$

If $\mu \in [\![P]\!]_G$, μ is said to be a solution for P in G.

The evaluation of more complex graph pattern is compositional and can be defined recursively from the basic graph pattern evaluation .

2.3.4 The streaming operators

When the R2R operator transforms a set of mapping, the system can produce the output with the computation result. If the output of the query processor should be a stream, it is necessary to include a R2S operator. When applied, it appends the set of mappings to the output stream. At each time instant at which the continuous query is evaluated, the set of solution mappings is processed by the R2S operator, which is in charge determine which data items have to be streamed out. There are usually three R2S operators, depicted in Figure 9.

Rstream streams out the computed timestamped set of mappings at each step. Rstream answers can be verbose as the same solution mapping can be computed at different evaluation times, and consequently streamed out. It is suitable when it is important to have the whole SPARQL query answer at each step, e.g. discover museum attractions that are popular in a social network in the recent time period.

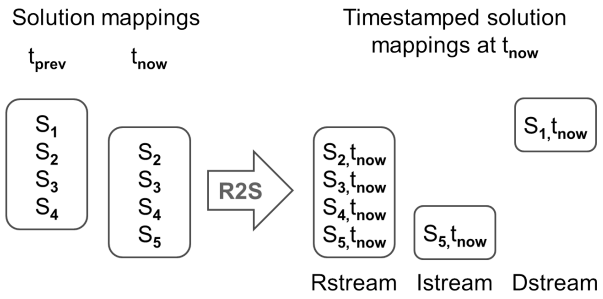

Fig. 9 Streaming operators

Istream streams out the difference between the timestamped set of mappings computed at the last step and the one computed at the previous step. Answers are usually short (they contain only the differences) and consequently this operator is used when data exchange is expensive. Istream is useful when the focus in on the new mappings that are computed by the system, e.g. discover emerging museum attractions in a social network.

Dstream does the opposite of Istream: it streams out the difference between the computed timestamped set of mappings at the previous step and at the last step. Dstream is normally considered less relevant than Rstream and Istream, but it can be useful, e.g. to retrieve attractions that are no more popular.

2.3.5 Temporal operators

So far only typical operators of DSMS were introduced. They allows to match graph patterns in the streaming data. However, practical scenarios (including the one illustrated in Section 4) may require to check if two graph patterns are observed at the same time, one after each other or when typical temporal relations between intervals holds.

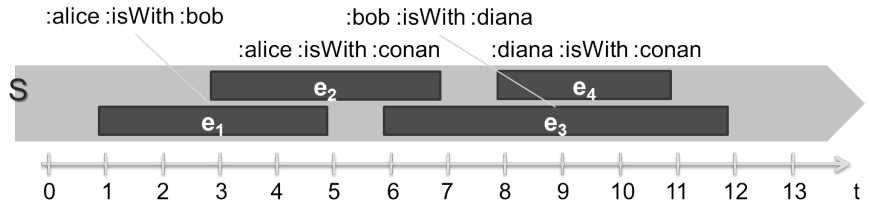

Fig. 10 Example of stream with two timestamps per data item.

For instance, given the situation illustrated in Figure 10, one may be willing to detect the people who are now with people who were observed together. At time 6, the answer is Conan and Diana, because they are with Alice and Bob who were observed together between 1 and 5.

Figure 11 informally introduces the temporal relations between two temporal intervals as defined in Allen's Algebra [2, 37]. Assume that compatible solution mappings exist for three graph patterns P_1, P_2, and P_3 in the time intervals shown in Figure 11, the horizontal bars represent the result of evaluating the operators on the solution mappings at the different time units depicted with vertical dashed lines.

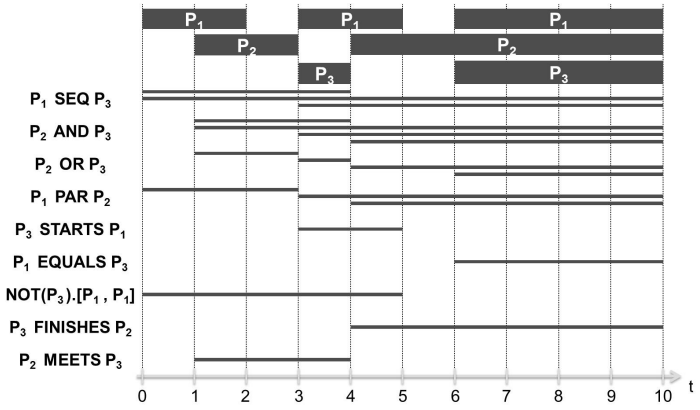

Fig. 11 Temporal relations between two temporal intervals as defined in Allen's Algebra.

A subset of those temporal operators can be added to the RSP query model as special types of joins that add to the boolean semantics of the join operator, the temporal semantics of the specific temporal relation of the operator. For instance, the solution mappings of P_1 `SEQ` P_2 from a membership perspective are those of (P_1 `JOIN` P_2), but the `SEQ` operator keeps only the solution mappings of the `JOIN` operator for which a solution mapping of P_2 starts after the end of a solution mapping of P_1.

3 RSP implementations

The previous section explains the data models and the query models used by RDF stream processing systems: putting together techniques and concepts from DSMS, CEP, and Semantic Web researches, RSP engines are systems able to cope with the semantic heterogeneity of the data. The remaining of this section presents an overview of existing systems.

Table 1 Data models adopted by RSP systems

	C-SPARQL	CQELS	SPARQL$_{stream}$	INSTANS	ETALIS	SLD
Data item	statement	statement	statement	statement	statement	graph
Applitaction time	1	1	1	0	2	1
Time-based sliding windows	✓	✓	✓			✓
Triple-based sliding windows	✓	✓				✓
S2R	RStream	IStream	RStream IStream DStream	RStream	RStream	RStream
SEQ					✓	
EQUALS					✓	

Table 1 lists the systems and classify them according to the RSP data and query models presented above. Analysing the table, it is possible to observe that most systems focus on RDF streams with RDF statement as data item and the application time is composed of one timestamp. C-SPARQL [9], CQELS [27], SPARQL$_{stream}$ [14] manage data streams where data items are RDF statements. Their query models are similar, but they are designed in different ways and they target different use cases. Sections 3.1, 3.2 and 3.3 discuss them.

The data model with one timestamped RDF statements is the one on which the initial RSP research focused, and novel trends started to consider data model variants. The SLD platform, described in Section 3.6, has features similar to C-SPARQL, CQELS and SPARQL$_{stream}$, but it is able to process RDF streams with RDF graphs as data items. INSTANS (Section 3.4) follows a completely different approach: it takes sequences of RDF statements without timestamps as input and processes them through the RETE algorithm. Finally, ETALIS and EP-SPARQL, presented in Section 3.5, work on RDF streams with application time composed of two timestamps: they use CEP concepts and are able to perform Stream Reasoning tasks.

3.1 C-SPARQL

Continuous SPARQL (C-SPARQL) [9] is a language for continuous queries over streams of RDF data that extends SPARQL 1.1. Is it implemented in the C-SPARQL engine, and it allows to register queries that are continuously evaluated over time.

Figure 12 presents a high-level description of the C-SPARQL Engine architecture. The engine is capable of registering queries and running them continuously, accordingly to the configuration of the engine. To this end, the C-SPARQL Engine . uses two subcomponents, a Data Stream Management System and a SPARQL engine. The former is responsible of executing continuous queries over RDF Streams, producing a sequence of RDF graphs over time, while the latter runs a standard SPARQL query against each RDF graph in the sequence, producing a continuous result. This result is finally formatted as specified in the query: if the SELECT or ASK forms are used, then the result is a relational data stream; if the CONSTRUCT form is used, the result is an RDF stream. Both the SPARQL engine and the DSMS are plug-ins of the C-SPARQL engine, and they can be changed. The binaries of the engines use Esper[8] as DSMS and Apache Jena-ARQ[9] as query engine.

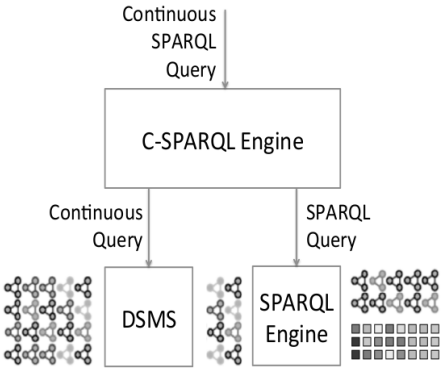

Fig. 12 Architecture of the C-SPARQL engine

3.2 CQELS

The Continuous Query Evaluation over Linked Streams (CQELS) accepts queries in CQELS-QL [27] – a declarative query language built from SPARQL 1.1 grammar. As C-SPARQL, it extends SPARQL with operators to query streams. The main difference between C-SPARQL and CQELS-QL is in the R2S operator supported: CQELS-QL supports only Istream, whereas C-SPARQL supports only Rstream.

Differently from the C-SPARQL Engine that uses a "black box" approach which delegates the processing to other engines, CQELS proposes a "white box" approach (see Figure 13) and implements the required query operators natively to avoid the overhead and limitations of closed system regimes. CQELS provides a flexible query execution framework with the query processor dynamically adapting to the changes in the input data. During query execution, it continuously reorders

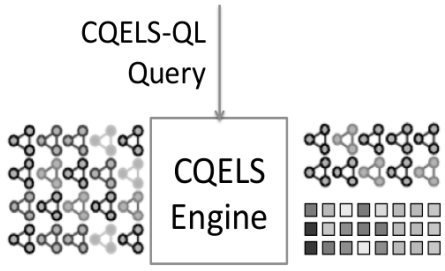

Fig. 13 Architecture of CQELS

[8] Cf. http://esper.codehaus.org/.

[9] Cf. http://jena.apache.org/documentation/query/.

operators according to heuristics that improve query execution in terms of delay and complexity. Moreover, external disk access on large Linked Data collections is reduced with the use of data encoding and caching of intermediate query results. It returns both data streams and RDF streams depending on the query form used.

3.3 SPARQL$_{stream}$

SPARQL$_{stream}$ [14] is another extension of SPARQL that supports operators over RDF streams such as time windows. Unlike CQELS and the C-SPARQL Engine, SPARQL$_{stream}$ supports all the streaming operators presented in Section 2.3.4. The language is adopted in morph-streams: it is an RDF stream query processor that uses Ontology-Based Data Access techniques [15] for the continuous execution of SPARQL$_{stream}$ queries against virtual RDF streams that logically represent relational data streams.

Morph-streams uses R2RML[10] to define mappings between ontologies and data streams. SPARQL$_{stream}$ queries are rewritten in continuous queries over the data streams. The rewriting process does not translate directly a SPARQL$_{stream}$ query in a continuous query in the target language of the underlying DSMS (Figure 14). It represents, instead, the query as a relational algebra expression extended with time window constructs. This allows performing logical optimizations (including pushing down projections, selections, and join and union distribution) and translating the algebraic representation into a target language or REST API request.

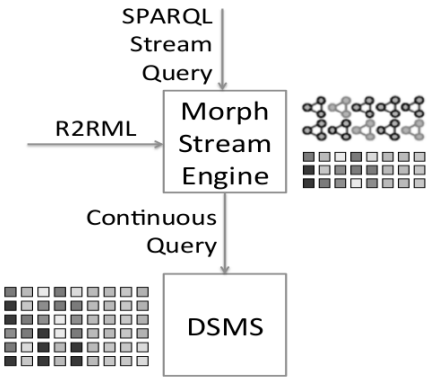

Fig. 14 Architecture of Morph-streams

The algebraic representation can be translated into both DSMS continuous queries, e.g. SNEE[11] or Esper; and Sensor Middleware REST API invocation, e.g. GSN[12] or Cosm/Xively[13].

[10] Cf http://www.w3.org/TR/r2rml/.

[11] Cf. http://code.google.com/p/snee/.

[12] Cf. http://lsir.epfl.ch/research/current/gsn/.

[13] Cf. https://xively.com/.

3.4 INSTANS

INSTANS (Incremental eNgine for STANding Sparql) [36] takes a different perspective on RDF Stream Processing. It asks the users to model their continuous query problem as multiple interconnected SPARQL 1.1 queries and rules. It performs continuous evaluation of incoming RDF data against the compiled set of queries, storing intermediate results. It has no notion of S2R operator; when all the conditions of a query are matched, the result is instantly available. For this reason it requires no extensions to RDF or SPARQL. As shown in Figure 15, to process the multiple interconnected queries it compiles them into a Rete-like structure.

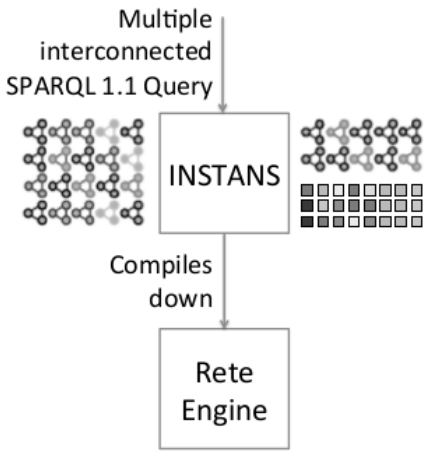

Fig. 15 Architecture of INSTANS

3.5 ETALIS and EP-SPARQL

ETALIS (Event TrAnsaction Logic Inference System) [3] is a CEP with Stream Reasoning features. As input data format it uses RDF statements annotated with two timestamps (see also Section 2.2). Users can specify event processing tasks in ETALIS using two declarative rule-based languages called: ETALIS Language for Events (ELE) and Event Processing SPARQL (EP-SPARQL) [3] . Both languages have the same semantics: complex events are derived from simpler events by means of deductive prolog rules (Figure 16).

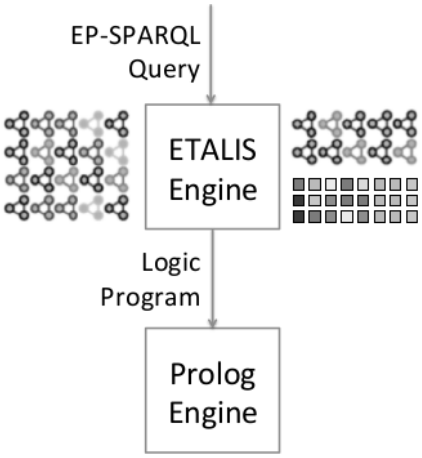

Fig. 16 Architecture of ETALIS

ETALIS not only supports typical event processing constructs (e.g. sequence, concurrent conjunction, disjunction and negation), but it also supports reasoning about events. For example, as CEPs, it allows to check for sequences such as $A \rightarrow B$ (an event of type A followed by an event of type B). However, it also allows stating that C is a subclass of A. Therefore the condition $A \rightarrow B$ will be matched also

if an event of type *C* is followed by an event of type *B*, because all events of type *C* are also events of type *A*.

Moreover, the two languages support all operators from Allen's interval algebra (e.g. during, meets, starts, finishes, as described in Section 2.3.5); count-based sliding windows; event aggregation for `COUNT`, `AVG`, `SUM`, `MIN`, `MAX`; event filtering, enrichment, projection, translation, and multiplication; processing of out-of-order events (i.e. events that are delayed due to different circumstances e.g. network anomalies etc.); and event retraction (revision). ETALIS is a pluggable system that can use multiple prolog engines such as YAP[14], SWI[15], SICStus[16], XSB[17], tuProlog[18] and LPA Prolog[19].

3.6 SLD

SLD is not a proper RSP engine, but it wraps the C-SPARQL engine in order to add support features, such as extensible means for real-time data collection, Linked Data publishing, and data and query result visualising. As depicted in Figure 17, the SLD framework offers: a set of adapters that transcode relational data streams in streams of RDF graphs (e.g. a stream of micro-posts as an RDF stream using the SIOC vocabulary [13], or a stream of weather sensor observation using the Semantic Sensor Network vocabulary [17]), a publish/subscribe bus to internally transmit RDF streams, facilities to record and replay RDF streams, an extendable component to decorate RDF streams (e.g. adding sentiment annotations to micro-posts), and a linked data server to publish results following the Streaming Linked Data Format [11].

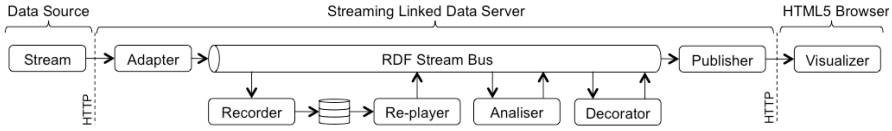

Fig. 17 Architecture of SLD

SLD is realised in order to ease the task of deploying the C-SPARQL engine in real-world applications: it is at the basis of works of real time data analysis of city scale events (i.e. group of events located in multiple venues around a city)

[14] Cf. http://www.dcc.fc.up.pt/ Evsc/Yap/.

[15] Cf. http://swi-prolog.org/.

[16] Cf. http://www.sics.se/isl/sicstuswww/site/index.html.

[17] Cf. http://xsb.sourceforge.net/.

[18] Cf. http://www.alice.unibo.it/xwiki/bin/view/Tuprolo.

[19] Cf. http://www.lpa.co.uk/win.html.

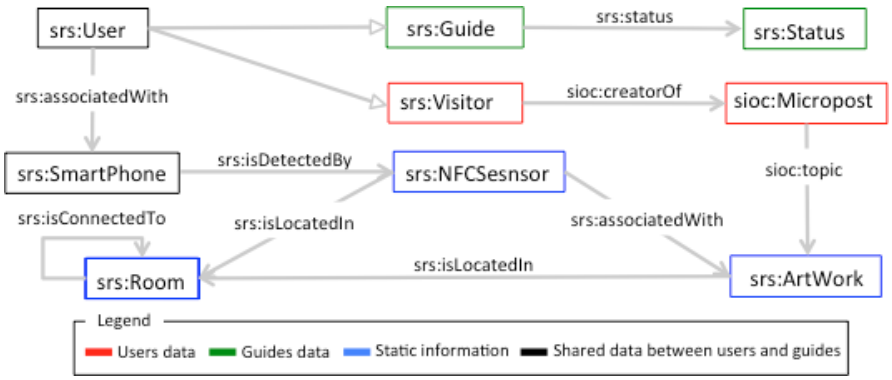

Fig. 18 Data model of the Blitz Guided Thematic Tour case study

for London Olympic Games 2012, Milano Design Week 2013 and Milano Design Week 2014 [7].

4 Case study

This section puts at work, in a realistic cultural heritage case study, the elements introduced in the previous sections. It proposes Blitz Thematic Guided Tours as an engaging cyber-physical-social service to experience a museum. This service allows free guides for offering a short and focused guided tour that matches the interests of the people nearby with the artworks exposed in the rooms at hand and with their own expertise.

The section starts describing the requirements such a service has to implement. Then, it presents an overview of an RSP based implementation. The emphasis is posed on the data (both streaming and static) and on the continuous queries that process it.

The services has to address the following **requirements**:

- R.1 It must be able to merge information from multiple heterogeneous data streams.
- R.2 It must allow to detect where the visitors are in the museum with a minimum delay.
- R.3 It should understand what the visitors are interested in.
- R.4 It should recommend free guides to organise a thematic tour.

The **data** involved in the Blitz Guided Thematic Tour is collected in different ways and from different data sources. It is worth to note that the visitors are not only service consumers, but also data producers. The means to collect the data are:

- User Mobile App: the visitors, when entering the museum, are offered to download the official app of the museum. This app offers information about

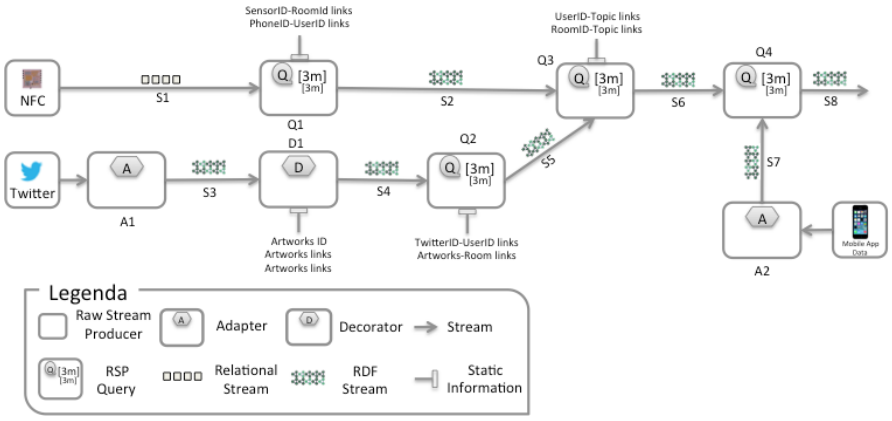

Fig. 19 Overview of the architecture of the Blitz Guided Thematic Tour service

artworks using NFC technology and allows guides to invited visitors to join a Blitz Guided Thematic Tour.

- Guide Mobile App: the guides also have a mobile app that let them specify their status (e.g. if they are available to start a tour) and their position. This application represents the communication channel between the guides and the visitors involved in a Blitz Guided Thematic Tour.
- NFC Sensors: these sensors are placed aside every artwork. They identify the artwork and allow visitors to learn more about the artwork. As a side effect, they allow indoor positioning of the visitors.
- Social Networks: they are the only sources of the knowledge about the users. The official User Mobile App simplifies visitors' task of sharing online the artworks they liked by asking visitors to connect their social network accounts. For the users who do so, the service can learn the language(s) and the interests of the users analysing past micro-posts (e.g. using the technique described in [6]).
- Static Knowledge: information related to the artworks, the rooms they are displayed in, how the rooms are connected, and the expertise of the guides.

Figure 18 presents the data model of the case study. `Users` (i.e. `Visitors` and `Guides`) are identified in the system via the UUID of their `Smartphone` where the two official mobile applications are installed: the red part of the Figure presents the data related to each `Visitor`, in particular the information the service can obtain from the social network; the green part refers to the `Guides`' information, their positions and their statuses; finally the blue part models the static information about each `NFC sensor` that is `associatedWith` an `ArtWork` which is `locatedIn` a `Room` which `isConnectTo` one or more `rooms`.

Figure 19 offers an overview of the architecture of the Blitz Guided Thematic Tour service with a focus on data streams, static data and the components that transform the information into actionable knowledge (i.e. recommendations for free guides). The visual language is the one proposed for the Streaming Linked Data

framework [7]. Each component represents a step in the data transformation process, while the input and output data are depicted in a different way depending on whether they are static or streaming. The remainder of the section details this picture step by step referring to the numbering of the elements.

Let us start from the NFC sensor. Smart phones equipped with NFC represent the cheapest way for indoor positioning. In the case study the visitors' positions are acquired as a result of the action of learning more about artworks exhibited in the museum. The NFC reader produces a relational stream (see component S1 in Figure 19) presented in the Listing 1 containing the tuples <phoneID,sensorID, timestamp>.

Listing 1 Two Events of Stream S1 produced by NFC sensors and consumed by the query Q1. For simplicity we omit Timestamps in the following listings.

```
phone1, monnalisaSensor [τ1]
phone2, theKissSensor [τ2]
```

Query Q1 addresses the requirement R.2. It continuously processes this relational stream treating it as a *virtual RDF stream* by using OBDA approach briefly introduced in Section 1. A mapping file, written in R2RML, defines mappings between the ontology illustrated in Figure 18 and the relational data streams. It maps the relational data in the streams to the objects in the ontology. It is worth to note that in order to save space, the prefixes clauses are shown only the first time the prefix is required and they are omitted in all the listings that appear after.

Listing 2 contains the R2RML file to map the detection of a smart phone by a sensor. Line 9 declares how to produce the subject of the triple using the template URL along with the phoneID parameters extracted from the stream S1 specified at Line 8. The predicate of the triples, srs:detectedBy, is created at Line 10 along with the object, specified at Line 12 using the template URL for the NFC sensors.

Listing 2 R2RML file to map the detection of a smartphone by a sensor

```
1  @prefix rr: <http://www.w3.org/ns/r2rml#> .
2  @prefix srs:<http://sr.org/ontologies/2014/7/srs#> .
3  @prefix : <http://sr.org/R2RMapping#> .
4  @prefix phone:<http://sr.org/data/smartphones/>
5  @prefic sensor:<http://sr.org/data/sensors/>
6
7  :smartphoneMap a rr:TriplesMap ;
8    rr:logicalTable [ rr:tableName "S1" ];
9    rr:subjectMap [ rr:template "phone:{smartphoneID}" ];
10   rr:predicateObjectMap [
11     rr:predicate srs:detectedBy;
12     rr:objectMap [ rr:template "sensor:{sensorID}" ] ] ].
```

Listing 3 presents the query that processes the relational data stream S1 treating it as a virtual RDF stream. It is worth to note that we are in a streaming setting, the queries are registered and periodically run against the flowing data. The REGISTER STREAM clause at Line 1 denotes the registration process for the query and tells the

system to produce a as output. It must be coupled with the `CONSTRUCT` clause, specified at Line 3, to tell the system the format of the output stream. The `FROM STREAM` clause at Line 4 specifies the query input stream. The unbound nature of a stream requires the ability to specify a criteria to select a part of flowing data (e.g. a time based window sorting out the most recent triples). The `RANGE` and `STEP` clauses identify the time-based sliding window containing the data to analyze, at Line 3 we specified a 10 minutes width window with a slide of 1 minute. The `FROM` clause identifies the online source of RDF static data. The `WHERE` clause at Line 5 opens the triples pattern specifications. The triple pattern at Line 5 matches all the triples with the `srs:detectedBy` predicate from the current window over the stream to find the links between smart phones and sensors. Then, the triple patterns at Line 6 and Line 7 are matched against the static knowledge base, to identify the users associated with the smartphones (`?userID`) and the room in which a sensor is located (`?roomID`).

Listing 3 Query Q1 that processes the stream S1 and produces stream S2

```
1 REGISTER QUERY Q1 AS
2 CONSTRUCT { ?userID srs:detectedAt ?roomID }
3 FROM STREAM <http://sr.org/streams/S1> [RANGE 10m STEP 1m]
4 FROM <http://sr.org/staticData/museum.rdf>
5 WHERE {  ?phoneID srs:detectedBy ?sensorID ;
6                   srs:associatedWith ?userID .
7          ?sensorID srs:locatedIn ?roomID . }
```

Let us assume that `user:123` and `user:345` respectively own `phone1` and `phone2`. The query Q1, observing in input the relational data stream in Listing 1, produces as output on the RDF stream S2 the triples shown in Listing 4.

Listing 4 An event in the Stream S2 produced by the query Q1 and consumed by the query Q3

```
@prefix rooms:<http://sr.org/data/rooms/>
@prefix topics:<http://sr.org/data/topics/>
@prefix users:<http://www.sr.org/data/users/>

users:123 srs:detectedAt  rooms:31 .
users:345 srs:detectedAt  rooms:42 .
```

Let us now focus on the application installed on the visitor's smartphone. It represents the communication channel between the museum and the users. Once installed, the application, via a configuration wizard, asks the users to insert their social network accounts credentials (e.g. twitter). Using twitter credentials a twitter adapter (see component A1 in Figure 19) creates a new RDF stream (S3) *wrapping* the twitter stream API. The information about the micro-posts of the users flows on this RDF stream. The Listing 5 shows an example of micro-post in RDF that flows on this stream. The vocabulary used to describe micro-posts is an extension of the SIOC ontology [13]. It states that the micro-post `@Louvre #monnalisa #wow` is posted by `user:543`, it contains the hashtags `monnalisa` and `wow`, and the smartphone was located in a given geo-position.

Listing 5 An event in the Stream S3 produced by the adapter A1 and consumed by the decorator D1

```
@prefix geo:<http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix sioc:<http://rdfs.org/sioc/ns#> .
@prefix mp:<http://www.sr.org/data/mp/>
@prefix tags:<http://www.sr.org/data/tags/>

mp:1234 sioc:content  "@Louvre #monnalisa #wow"^^xsd:string ;
  sioc:has_creator  user:543 ;
  sioc:topic tag:monnalisa, tag:wow ;
  geo:location [ a geo:SpatialThing ;
     geo:lat   "48.860816"^^xsd:double ;
     geo:long  "2.338320"^^xsd:double ] .
```

The twitter decorator (D1) enriches the stream S3 by semantically annotating the micro-post that flows on the stream with links that points to the artworks in the Static Knowledge. For instance, it adds the following triple to the micro-post in Listing 5.

```
mp:123456789 sioc:topic <http://sr.org/data/artworks/Monnalisa> .
```

Several techniques can be employed to discover those links. For instance, the Streaming Linked Data framework [6] uses an aggregation of a spatial metric (that checks the geo-position of the micro-post to make sure it was posted in the neighbourhood of the museum) and four lexical similarity metrics that compare content and hashtags of each tweet to the names of the artworks and the bags of words that describe them.

The query Q2 addresses the requirement R.2 (i.e. detecting the position of the users) using the stream of tweets to position the visitors. Notably, Q2 complements the positions obtained with the readings of the NFC sensor (see Q1). It does so by processing the stream S4 and positioning the users in the rooms based on the artworks they have been talking about in the last 10 minutes. This continuous query operates on a *native RDF stream*, thus it does not need a mapping.

Listing 6 Query Q2 that processes the stream S4 and produces stream S5

```
1  REGISTER QUERY Q2 AS
2  PREFIX srs:<http://streamreasoning.org/ontologies/2014/7/srs#>
3  PREFIX sioc:<http://rdfs.org/sioc/ns#>
4  CONSTRUCT { ?userID srs:detectedAt ?roomID }
5  FROM STREAM <http://sr.org/streams/S4> [RANGE 10m STEP 1m]
6  FROM <http://sr.org/staticData/museum.rdf>
7  WHERE {
8     ?mp sioc:has_creator ?userID ;
9     sioc:topic ?artworksID .
10    ?artworksID srs:isIn ?roomID .
11 }
```

Using the decorated micro-posts that flows on the stream S4, e.g. the one illustrated in Listing 5, query Q2 links the user to the room where the artwork is exhibited. On its output stream S5, it pushes triples such as:

```
users:543 srs:detectedAt rooms:31 .
```

It is worth to note that the output produced by Query Q1 and Q2 is made of triples of the same type. This is typical of OBDA systems where complementary (and sometimes even overlapping) information is obtained from multiple data sources.

Query Q3, presented in Listing 7, answers R.1 (i.e. merging information from multiple heterogenous streams) and R.3 (i.e. understand what the visitors may be interested in). For each room, it counts the number of visitors that may be interested in a given topic. It outputs the top 5 most numerous groups (see `ORDER BY` and `LIMIT` clauses at Lines 18 and 19) formed by at least 10 visitors (see `HAVING` clause at Line 17). It does so by consuming the RDF streams of visitors' positions produced by Q1 and Q2 (see the two different `FROM STREAM` clauses at Lines 4 and 5) and some static data describing the topology of the museum and the topics that may interest the visitors. The former is used to collect the visitors and the artworks in a given room or in the room nearby, the latter is computed using an off-line analysis of the tweets of the visitors (e.g. using the approach described in [6]) and it is used to match topics of interest to topics describing the artworks. Notably the match requires to reason on the relations between the topics of the artworks and the topics of interest of the visitors. The query does so by considering that the property `skos:transitiveBroader` (see Line 14) is transitive and by traversing the graph that links the two topics in order to find a matching.

Listing 7 Query Q3 that processes the streams S2 and S5 and produces stream S6

```
1  REGISTER QUERY Q3 AS
2  PREFIX srs:<http://streamreasoning.org/ontologies/2014/7/srs#>
3  CONSTRUCT { ?roomID srs:contains [ srs:userCount ?userTotalCount
       ; srs:topic ?topic] }
4  FROM STREAM <http://sr.org/streams/S2> [RANGE 10m STEP 1m]
5  FROM STREAM <http://sr.org/streams/S5> [RANGE 10m STEP 1m]
6  FROM <http://sr.org/staticData/museum.rdf>
7  WHERE {
8     SELECT ?roomID ?userTopic (COUNT(?userID) AS ?userTotalCount)
9     WHERE {
10       ?roomID srs:isConnectedTo ?roomID2 .
11       ?roomID2 srs:contains ?artworksID .
12       ?artworksID srs:topic ?artworkTopic .
13       ?userID srs:isInterestedIn ?userTopic .
14       ?artworkTopic skos:transitiveBroader ?userTopic.
15    }
16    GROUP BY ?roomID, ?topic
17    HAVING (?userTotalCount>10)
18    ORDER BY ?userTotalCount
19    LIMIT 5
20 }
```

The query produces as output the stream S6. A snapshot of the RDF triples flowing on S6 is presented in Listing 8.

Listing 8 An event in the Stream S6 produced by query Q3 and consumed by the query Q4

```
<http://www.sr.org/data/room/31>
   a  srs:Room ;
   srs:contains [ srs:userCount "57"^^xsd:integer ;
      srs:topic topics:Renaissance_art ] .
```

In the case study, the guides' positions and the status (i.e. being free or occupied in a tour) are volunteered. Every time they enter a room of the museum they can use a mobile app to let the system know their status. Stream S7, presented in Listing 9, shows the information regarding the position and the status of every guide.

Listing 9 An event in the Stream S7 produced by the Guide Mobile App and consumed by the query Q4

```
guides:54 srs:detectedAt  rooms:31 ;
          srs:status statuses:free .
```

Finally, the last query of the chain (Q4), presented in Listing 10, answers to requirement R.4, i.e. recommending the free guides to organise a guided tour. Q4 is the most complex query in the case study; not only it has multiple streams in input (Lines 4 and 5) that have to be process together with static knowledge (Lines 6), but it is also a *event based* query. The `DURING` clause, at line 9, allows to specify that the pattern preceeding the `DURING` clause must be valid while the pattern following the `DURING` clause is valid. In this specific case, it checks that while there is a large enough group of people interested in a topic in a room, a guide able to talk about such topic is present in the same room or in a room nearby.

Listing 10 Query Q4 that processes the streams S6 and S7 and produces stream S8

```
1  REGISTER QUERY Q4 AS
2  PREFIX srs:<http://streamreasoning.org/ontologies/2014/7/srs#>
3  CONSTRUCT { ?roomID srs:contains [ srs:userCount ?userTotalCount
       ; srs:topic ?topic] }
4  FROM STREAM <http://sr.org/streams/S6> [RANGE 10m STEP 1m]
5  FROM STREAM <http://sr.org/streams/S7> [RANGE 10m STEP 1m]
6  FROM <http://sr.org/staticData/museum.rdf>
7  WHERE {
8     { ?roomId srs:contains [ srs:userCount ?userTotalCount ; srs:
          topic ?topic] . }
9     DURING
10    { ?guideID srs:status <http://sr.org/data/statuses/free> ;
11      srs:expertIn ?topic .
12      { ?guideID srs:detectedAt ?roomID . }
13        UNION
14      { ?guideID srs:detectedAt ?roomID2 . ?roomID2 srs:
            connectedTo ?roomID . }
15    }
16 }
```

Query Q4 pushes the recommendations for the guides on the stream S8. An example of recommendation is presented in Listing 11. The recommendations will

appear on the mobile app of the guide who can decide to try to engage the visitors with a blitz guided tour about Renaissance Art.

Listing 11 An event in the Stream S8 produced by the query Q4 containing the Blitz Guided Thematic Tour suggestions for the Guides

```
guides:54 a srs:Guide ;
          srs:suggestedTour [ srs:userCount "54"^^xsd:integer ;
          srs:topic  topics:Renaissance_art ] .
```

5 Conclusions

The integration of heterogeneous data from different sources is gaining more and more attention, and when the data is characterised by a high degree of dynamism, as happens with data stream, new problems raise. This chapter discusses the semantic interoperability of data streams through RDF streams and RDF Stream Processing engines. It analyses the data models, the query models, and the existing implementations. It highlights the features of such implementations and their points of strength. Finally, it presents a case study to show how to cope with data heterogeneity in a realistic scenario.

Research in RSP is still open, and new trends are rising. Given the number of existing systems, a problem is their evaluation: there are several initiatives to compare the operators they support [41], their performances [28], and the ability to produce correct results [21]. The former highlights the differences in the query language syntaxes and semantics, and it was the starting point to an initiative to define shared standards in RSP, through a W3C Community Group[20].

Reasoning on heterogenous, incomplete and noisy data streams is one of the most active topics in RDF Stream Processing. It investigates methods and techniques to define conceptual schemata for streaming data and to use them as driver to access the data streams, to query them and to integrate the continuously flow of answers. We saw that EP-SPARQL supports some Stream Reasoning features, while SPARQL$_{stream}$ is a first attempt of ontology-based data stream access. They are initial works and their support to Stream Reasoning techniques is still immature. IMaRS [22] and DynamiTE [40] are more advanced reasoning techniques to process RDF streams they use ontologies to materialise the implicit data in data streams, coping with the dynamic and incomplete nature of the streams. Some works on Inductive Stream Reasoning also address the noisy nature of data streams [10], but more research is required.

[20] Cf. http://www.w3.org/community/rsp/.

References

1. Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors. *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, volume 8219 of *Lecture Notes in Computer Science*. Springer, 2013.
2. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
3. Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in etalis. *Semantic Web*, 3(4):397–407, 2012.
4. Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2):121–142, 2006.
5. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In Popa [34], pages 1–16.
6. Marco Balduini, Alessandro Bozzon, Emanuele Della Valle, Yi Huang, and Geert-Jan Houben. Recommending venues using continuous predictive social media analytics. *IEEE Internet Computing*, 18(5), 2014.
7. Marco Balduini, Emanuele Della Valle, Daniele Dell'Aglio, Mikalai Tsytsarau, Themis Palpanas, and Cristian Confalonieri. Social listening of city scale events using the streaming linked data framework. In Alani et al. [1], pages 1–16.
8. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Incremental reasoning on streams and rich background knowledge. In *Proc. of ESWC2010*, 2010.
9. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Querying rdf streams with c-sparql. *SIGMOD Record*, 39(1):20–26, 2010.
10. Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Yi Huang, Volker Tresp, Achim Rettinger, and Hendrik Wermser. Deductive and inductive stream reasoning for semantic social media analytics. *IEEE Intelligent Systems*, 25(6):32–41, 2010.
11. Davide Francesco Barbieri and Emanuele Della Valle. A proposal for publishing data streams as linked data - a position paper. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas, editors, *LDOW*, volume 628 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
12. Irina Botan, Roozbeh Derakhshan, Nihal Dindar, Laura M. Haas, Renée J. Miller, and Nesime Tatbul. Secret: A model for analysis of the execution semantics of stream processing systems. *PVLDB*, 3(1):232–243, 2010.
13. John G. Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. Sioc: an approach to connect web-based communities. *IJWBC*, 2(2):133–142, 2006.
14. Jean-Paul Calbimonte, Óscar Corcho, and Alasdair J. G. Gray. Enabling ontology-based access to streaming data sources. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2010.
15. Jean-Paul Calbimonte, Hoyoung Jeung, Óscar Corcho, and Karl Aberer. Enabling query technologies for the semantic sensor web. *Int. J. Semantic Web Inf. Syst.*, 8(1):43–63, 2012.
16. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
17. Michael Compton, Payam M. Barnaghi, Luis Bermudez, Raul Garcia-Castro, Óscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory A. Henson, Arthur Herzog, Vincent A. Huang, Krzysztof Janowicz, W. David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin R. Page, Alexandre Passant, Amit P. Sheth, and Kerry Taylor. The ssn ontology of the w3c semantic sensor network incubator group. *J. Web Sem.*, 17:25–32, 2012.

18. Gianpaolo Cugola and Alessandro Margara. *Introduction to the Complex Event Processing (CEP) approach - Chapter 6 in this book*, pages xx–yy. Springer Verlag, Berlin, 2015.
19. Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. *W3C recommendation*, 2012.
20. Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.
21. Daniele Dell'Aglio, Jean-Paul Calbimonte, Marco Balduini, Óscar Corcho, and Emanuele Della Valle. On correctness in rdf stream processor benchmarking. In Alani et al. [1], pages 326–342.
22. Daniele Dell'Aglio and Emanuele Della Valle. Incremental Reasoning on RDF Streams. In Andreas Harth, Katja Hose, and Ralf Schenkel, editors, *Linked Data Management*. Chapman and Hall/CRC, 2014.
23. Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological queries: Rewriting and optimization. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *ICDE*, pages 2–13. IEEE Computer Society, 2011.
24. Ashish Gupta and Iderpal Singh Mumick, editors. *Materialized views: techniques, implementations, and applications*. MIT Press, Cambridge, MA, USA, 1999.
25. Yevgeny Kazakov. Consequence-Driven Reasoning for Horn SHIQ Ontologies. In Craig Boutilier, editor, *Proc 21st Int Conf on Artificial Intelligence (IJCAI'09)*, pages 2040–2045. IJCAI, 2009.
26. Graham Klyne and Jeremy J Carroll. Resource description framework (RDF): Concepts and abstract syntax. *W3C recommendation*, 2006.
27. Danh Le Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, pages 370–388. Springer, 2011.
28. Danh Le Phuoc, Minh Dao-Tran, Minh-Duc Pham, Peter A. Boncz, Thomas Eiter, and Michael Fink. Linked stream data processing engines: Facts and figures. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *International Semantic Web Conference (2)*, volume 7650 of *Lecture Notes in Computer Science*, pages 300–312. Springer, 2012.
29. Maurizio Lenzerini. Data integration: A theoretical perspective. In Popa [34], pages 233–246.
30. Deborah L McGuinness, Frank Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.
31. Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. OWL 2 web ontology language: Structural specification and functional-style syntax. *W3C recommendation*, 27(65):159, 2009.
32. C Enrique Ortiz. An introduction to near-field communication and the contactless communication API. *Oracle Sun Developer Network*, 2008.
33. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3), 2009.
34. Lucian Popa, editor. *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*. ACM, 2002.
35. Eric PrudHommeaux, Andy Seaborne, et al. SPARQL query language for RDF. *W3C recommendation*, 2008.
36. Mikko Rinne, Esko Nuutila, and Seppo Törmä. Instans: High-performance event processing with standard rdf and sparql. In Birte Glimm and David Huynh, editors, *International Semantic Web Conference (Posters & Demos)*, volume 914 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2012.

37. Fabio Schreiber. Is time a real time? an overview of time ontology in informatics. In WolfgangA. Halang and AlexanderD. Stoyenko, editors, *Real Time Computing*, volume 127 of *NATO ASI Series*, pages 283–307. Springer Berlin Heidelberg, 1994.
38. Fabio A. Schreiber, Emanuele Panigati, and Carlo Zaniolo. *Data Streams and Data Stream Management - Chapter 5 in this book*, pages xx–yy. Springer Verlag, Berlin, 2015.
39. Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. Consequence-based reasoning beyond horn ontologies. In Toby Walsh, editor, *IJCAI*, pages 1093–1098. IJCAI/AAAI, 2011.
40. Jacopo Urbani, Alessandro Margara, Ceriel J. H. Jacobs, Frank van Harmelen, and Henri E. Bal. Dynamite: Parallel materialization of dynamic rdf data. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *International Semantic Web Conference (1)*, volume 8218 of *Lecture Notes in Computer Science*, pages 657–672. Springer, 2013.
41. Ying Zhang, Minh-Duc Pham, Óscar Corcho, and Jean-Paul Calbimonte. Srbench: A streaming rdf/sparql benchmark. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 641–657. Springer, 2012.