A Query Model for Ontology-Based Event Processing over RDF Streams

Riccardo Tommasini, Pieter Bonte, Emanuele Della Valle, Femke Ongenae, Filip De Turck

Politecnico di Milano, DEIB, Milan, Italy
`{riccardo.tommasini,emanuele.dellavalle}@polimi.it`
Ghent University - imec
`{pieters.bonte,filip.deturck,femke.ongenae}@ugent.be`

Abstract. Stream Reasoning (SR) envisioned, investigated and proved the possibility to make sense of streaming data in real-time. Now, the community is investigating more powerful solutions, realizing the vision of expressive stream reasoning. Ontology-Based Event Processing (OBEP) is our contribution to this field. OBEP combines Description Logics and Event Recognition Languages. It allows describing events either as logical statements or as complex event patterns, and it captures their occurrences over ontology streams. In this paper, we define OBEP's query model, we present a language to define OBEP queries, and we explain the language semantics.

Keywords: Stream Processing, Semantic Web, Stream Reasoning, Complex Event Processing

1 Introduction

Stream Reasoning (SR) is evolving fast. Answering information-needs in real-time has a business-critical role for many applications. SR research is pushing the boundaries of the state-of-the-art, addressing three crucial challenges towards real-time decision-making: (1) enabling continuous analytics over heterogeneous data streams, (2) enabling event detection considering both domain knowledge and streams of events, and (3) solving (1) and (2) simultaneously [6, 10, 17].

The SR solution for continuous data integration is RDF Stream Processing (RSP). RSP engines adopt the Data Stream Management Systems (DSMS) processing model to tame velocity and combine it with semantic technologies, which can tame data variety (e.g., RDF and SPARQL) [10]. Therefore, RSP engines can execute continuous analytical tasks over heterogeneous data streams.

For event detection, there is not a unified SR solution [10]. On the one hand, there is the ETALIS logic-programming framework that offers a time-aware rule-based language for SR and complex event processing (CEP). ETALIS treats event detection as a reasoning task exploiting the relation between event-recognition languages and temporal reasoning [7]. On the other hand, RSP proposals for CEP aim at providing a unified language for event detection and analytics [1, 8].

EP-SPARQL [1] and RSEP-QL [8] extend respectively SPARQL 1.0 and an RSP-QL [9] with time-aware operators. Both started from a language for continuous analytics and added operators for event detection, as showed in [19]. To preserve the operational semantics of the base languages, existing constructs are reused to define events and, thus, events are not first-class objects in the resulting language. However, the notion of event is critical for CEP languages as CEP users expect two clear ways to define events, i.e., providing schema or types, or using temporal operators [4, 6, 7].

In this paper, we investigate the foundations of Ontology-Based Event Processing (OBEP) [18]. OBEP enables event detection over RDF Streams without neglecting events first-class nature. It seamlessly combines temporal operators with a family of knowledge representation languages around the notion of event. OBEP's users can specify and compose events working with high-level abstractions. Moreover, OBEP works with RDF Streams, i.e., it can be combined with RSP analytics solutions.

The remainder of the paper is organized as follow: Section 2 introduces the notions relevant to understand the content of the paper. Section 3 explains OBEP data model and query model. Section 4 formalizes the evaluation semantics of the language. Section 5 compares OBEP to existing stream reasoning languages and shows how OBEP simplifies the encoding of SR tasks compared to the state-of-the-art. Finally, Section 6 concludes the paper and presents the future work.

2 Preliminaries

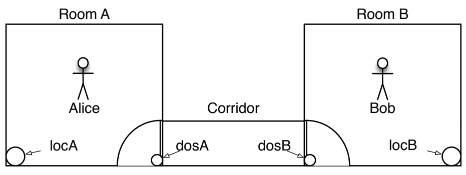

Fig. 1: Running Example.

In this section, we detail the required background and preliminary knowledge to introduce our contributions[1]. To this extent, we present a running example also depicted in Figure 1.

Two rooms, *Room A* and *Room B* are connected through a corridor. Room A is armed with a door sensor *sA* that monitors whether the door is *open* or *closed.* A people sensor *sB*, deployed in *Room B*, tracks the people inside. We also assume that either *Alice* or *Bob* can be in the room at a given time, that nobody else can be in the room and that the people sensor is capable of detecting when *nobody* is in the room.

[1] Due to the lack of space, we focus on the essential definition, and we provide references for the interested reader.

Description Logics (DLs) are a family of knowledge representation languages with reasoning capabilities [3]. We introduce the syntax of a simplified DL[2], explaining the basic notions required to understand the remainder of the paper.

DL defines *concepts* to represent the classes of individuals and *roles* to represent binary relations between the individuals. Basic concepts C can be defined as follows: $C ::= A_i|\top|\bot|\neg C|C_1 \sqcap C_2|C_1 \sqcup C_2|\exists P_1.C_1|\forall P_1.C_1$ where A denotes an atomic concept, P denotes an atomic role or its inverse P^-, $\top$ resembles the top concept, $\bot$ denotes the bottom concept, $\sqcap$ the conjunction of concepts, $\sqcup$ the disjunction of concepts, $\exists P.C$ states that there should exist a role P to an individual of the type C and $\forall P.C$ denotes that all roles P should be linked to an individual of the type C. To model the domain, we can define inclusion axioms of the form

$$C_1 \sqsubseteq C_2 \text{ and } P_1 \sqsubseteq P_2$$

where the equivalence between two concepts (e.g. $C_1 \equiv C_2$) can be interpreted as both C_1 and C_2 include each other (i.e. $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$).

A DL knowledge base consists of a terminological box (TBox) that collects intentional knowledge and an assertion box (ABox) that collects extensional knowledge. Listings 1.1 and 1.2 show a DL TBox and ABox respectively, that model the situations illustrated in Figure 1.

```
atomic concepts: Room, Person, Sensor
deployedIn ≡ armedWith⁻
observedBy ≡ observes⁻
observesPerson ⊑ observes
observesNobody ⊑ observes
observesClosedDoor ⊑ observes
observesOpenDoor ⊑ observes
```

Listing 1.1: Intensional knowledge describing Figure 1.

```
Room(RoomA)
Room(RoomB)
Person(Bob)
Person(Alice)
Sensor(sA)
Sensor(sB)
armedWith(RoomA, sA)
armedWith(RoomB, sB)
```

Listing 1.2: Extensional knowledge describing Figure 1.

RDF Statement, RDF Graph & SPARQL Dataset. An *RDF statement* is a triple (subject,predicate,object) $\in$ (I $\cup$ B) $\times$ (I) $\times$ (I $\cup$ B $\cup$ L), where I, B and L are respectively the sets of IRIs, blank nodes, and literals. A finite set of RDF statements is called a *RDF graph* A SPARQL Dataset (DS) is a set of pairs (u,G) where G is an RDF Graph and u is an IRI or a special symbol *def* denoting the default graph. For example, DS = { (def, G_0),(u_1,G_1)..}. A comprehensive discussion on SPARQL semantics can be found in [14].

Continuous Reasoning (CR) identifies those logic frameworks that are able to perform reasoning tasks over time. CR is based on the notion of ontology stream [16, 5].

Definition 1. *An Ontology Stream S^T is an unbounded sequence of pairs (A_i,t_i) where A_i is a set of ABox axioms compliant with a static TBox T, and t_i is a non-decreasing timestamp. S^T (i) returns the pair (A_i,t_i).*

Continuous reasoning can be reduced to traditional static DL reasoning if we consider the union of all the ABox axioms in a windowed ontology stream.

[2] We refer the reader to Horrocks et. al. [11] for a thorough discussion of a more expressive DL.

1	2	3	4	5	6	7	8
	sA :ocd :doorA sB :obsP :Bob	sA :ood :doorA sB :obsP :Bob	sA :ood :doorA sB :obsN :empty		sA :ood :doorA sB :obsN :empty	sA :ocd :doorA sB :obsP :Alice	

Table 1: An example of an RDF Stream. We use the following abbreviations: :ocd=observesClosedDoor, :ood=observesOpenDoor, :obsP=observesPerson, :obsN=observesNobody. Column names are the timestamps, while each entry is an RDF graph.

```
CONSTRUCT  {?person :isIn ?room }
FROM NAMED WINDOW <win> ON <stream> [RANGE 5s STEP 1s]
FROM <static_data>
WHERE { ?room a :Room ; :armedWith ?locSensor .
        WINDOW <win> { ?locSensor :observes ?p .
                       ?p a :Person }}
```

Listing 1.3: An example of RSP-QL query.

Definition 2. *A windowed ontology stream $S^T_{[o,c)}$ is a finite portion of an ontology stream, i.e., all the pairs $(A_i, t_i) \in S^T(i)$ such that $o \leq t_i < c$.*

RDF Stream Processing (RSP) identifies a family of SR approaches that aim to answer continuous queries over heterogeneous data streams [10]. To this extent, the fundamental notion is the one of RDF Stream [9]:

Definition 3. *An RDF Stream is a set of pairs $(G_i,\ t_i)$, where G_i is an RDF Graph, and t_i is a timestamp, e.g., $S = \{(G_1, t_1), (G_2, t_2), (G_3, t_3), (G_4, t_4), ...\}$.*

An example of an RDF Stream is depicted in Table 1. Column names are the timestamps, while each entry is an RDF graph.

A second important concept is the one of Continuous Queries over RDF Streams. RSP-QL [9] is a reference model that unifies all the existing RSP dialects [10]. Listing 1.3 shows an example of an RSP-QL query that continuously reports who is in the rooms in the example (see Figure 1). RSP-QL queries can be executed under an entailment regime combining RSP with OWL reasoning [10].

Operator	Cardinality	Matches
A AND B	Binary	When A and B are detected at the same time.
A OR B	Binary	When A or B or both are detected at a given time.
A SEQ B	Binary	When B is detected after A.
FIRST A	Unary	When the first occurrence of A is detected
LAST A	Unary	When the last occurrence of A is detected
ALLEN's Algebra	Binary	Denoted 12 relations between intervals.

Table 2: Most common event operators ($\mathcal{L}$) and their (informal) semantics.

Complex Event Processing (CEP) aims at recognizing and combining events over streams of data [1, 4, 13]. Event recognition relies on the notion of *Event Type* that characterizes events conceptual specification, and allows the assertions of Event Expressions w.r.t. a given CEP language [12]. Although many approaches populate CEP state-of-the-art – i.e., from Event-Condition-Action (ECA) [4] to Event Calculus [12] – a standard CEP language is still missing.

Since one of our requirements is considering events as first-class objects, we opted for Chackvrathy et al. [4] event algebra as a solid foundational formalism. Chackvrathy et al. defines events as data occurrences that happen completely or not at all (atomic). They distinguish between physical events, which are known and manipulated by the system, and conceptual events that are abstract specifications functional to the recognition task. Conceptual and physical events are linked via functional mappings that depends on the approach [4].

We consider the most language operators common to existing approaches in the Stream Reasoning state-of-the-art. Table 2 summarizes these operators ($\mathcal{L}$) and reports their semantics informally. Later on in the paper, we will formalize these operators w.r.t. our approach[3].

3 Ontology Based Event Processing

In this section, we present the foundational aspects of OBEP: we discuss its data model, the building block of OBEP queries as well as the semantics of the most relevant operators.

3.1 Data Model

Similar to existing approaches in Stream Reasoning state-of-the-art [1, 8], OBEP data model is based on RDF Streams: any RDF Graph in an RDF Stream represents an OBEP physical event. On the other hand, OBEP data model relies on the notion of Ontology Stream to represent conceptual events as logically defined event types. Although these notions are intuitively related, to the best of our knowledge, it is still missing a formal explanation of their relations. Therefore, in the following we provide a few intuitive yet necessary definitions that reconcile the two presented above. Moreover, we introduce the concept of Event Stream, that is required for the definition of OBEP query model. To support our formalization, we exploit the following helper functions: (i) *axioms* :: G → L, and (ii) *triples* :: L → G; where L is a set of logical axioms and G is the corresponding RDF Graph.

Definition 4. *A Well-Grounded RDF Stream S^D is an RDF Stream such that for each pair $(G_k,t_k) \in S^D$, the RDF Graph G_k is expressed according to a static TBox D, and G_i contains only triples of the form <:s rdf:type :C>, where <:C rdf:type owl:Class>, and <:s :p :o>, where <:p rdf:type owl:ObjectProperty>, and :s, :o are individuals. Literals are not considered.*

Example 1. The RDF Stream of Table 1 is a Well-Grounded RDF Stream w.r.t. the ontology of Listing 1.1.

Proposition 1. *For each Well-Grounded RDF Stream an Ontology Stream exists. Given a Well-Grounded RDF Stream S^D=(G_1,t_1),...(G_k,t_k) where G_i are defined according to a static TBox D. We denote with $\mathcal{D}$ = axioms(D) the*

[3] Due to the lack of space, we only present SEQ, FIRST, and DURING operators. The remaining ones are available in our extended version at `https://github.com/riccardotommasini/obep`

set of TBox axioms obtained converting triples from D[4], and we denote with A_k=axioms(G_k) the set of ABox axioms obtained converting triples from the RDF Graph G_k. The sequence of pairs (A_1, t_1),...,(A_k, t_k) about $\mathcal{D}$ is an Ontology Stream (denoted with $S^{\mathcal{D}}$).

Definition 5. *An Event Stream $S^{\mathcal{E}}$ is an Ontology Stream, where (i) the static TBox $\mathcal{E}$ contains some axioms of the form $E \sqsubseteq B$ where B is a basic concept, and E, which is distinguishable from other atomic classes*[5]*, is a logical event. (ii) for some (A_i, t_i) $\in S^{\mathcal{E}}$ it is true that $A_i \models^{\mathcal{E}} E_i(e)$ with E_i a logical event and e an individual. We call (A_i,t_i) a Physical Event.*

Proposition 2. *For each Well-Grounded RDF Stream an Event Stream exists. Given a Well-Grounded RDF Stream S^D from Proposition 1, we know that there is an Ontology Stream $S^{\mathcal{D}}$. Let's consider a static sub-set of $\mathcal{D}$, e.g., $\{E_1,...,E_n\}$, denoted as $\mathcal{E}$; (A_1, t_1)...(A_k,t_k) is an Event Stream (denoted as $S^{\mathcal{E}}$), if $A_k \models^{\mathcal{E}} E(e)$ for some k. We call E a logical event and we call (A_k,t_k) a Physical Event.*

3.2 Building Blocks of OBEP Queries

The OBEP query model is based on an abstract event specification, generically called complex events. We distinguish between Logical and Composite Events.

Definition 6. ***Logical Events*** *are logic assertions $H \leftarrow B$, where H is an atomic DL concept and B is a DL basic concept as specified in Section 2.*
The abstract syntax[6] *of Logical Event is*

$$\mathsf{EVENT}\ H\ \mathsf{AS}\ B$$

The helper function named(B) returns all the concepts and roles used in B.

Example 2. (**cont'd**) In Listing 1.4, we define the Logical Event $BusyRoom \sqsubseteq$B, where B is $Room \sqcap \exists\ armedWith.(Sensor \sqcap \exists observesPerson)$ where named(B) = {Room, Sensor,Person, armedWith, observesPerson}.

```
EVENT BusyRoom AS
  (Room and armedWith some (Sensor and observesPerson some Thing))

EVENT FreeRoom AS
  (Room and armedWith some (Sensor and observesOpenDoor some Thing))
```

Listing 1.4: Examples of Logical Events.

Definition 7. *A* ***Composite Event*** *is an assertion $H \leftarrow E$ where:*

- *H is an atomic DL concept denoting a logical event.*
- *E is an event expression.*

[4] We consider only the rules (i) <:s rdf:type :C> → C(s); (ii) <:s :p :o> → P(s,o)

[5] We implemented this mechanism using OWL Annotation Properties since they do not impact the reasoning, but allows distinguishing TBox axioms.

[6] we will use Manchester Syntax to express B `https://www.w3.org/TR/owl2-manchester-syntax/`

The abstract syntax for composite events is

$$\mathsf{EVENT}\ H\ \mathsf{MATCH}\ E$$

The helper function named(E) returns all the events used in E.

Example 3. (**cont'd**) In Listing 1.5, we define the complex event *Exiting* using OBEP syntax where named(Exiting) = {BusyRoom, FreeRoom}, and *Entering* using OBEP syntax where named(Exiting) = {BusyRoom, FreeRoom},

```
EVENT Entering MATCH FreeRoom SEQ BusyRoom
EVENT Exiting MATCH BusyRoom SEQ FreeRoom
```

Listing 1.5: A Complex Event without guards.

Logical and Composite Events are **Complex Event**s, i.e, assertions H ← L where (i) H is an atomic DL concept, and (ii) L is either (a) a basic DL concept or (b) an event expression.

Last but not least, we have to discuss which results a query generates. OBEP consumes streaming data and it produces streams as output. Inspired by SPARQL query forms[7], we introduce a return clause that allows alternative output stream formats.

Definition 8. *A* ***Return Clause*** *is a function that determines the type of stream to output from an OBEP query, i.e.,*

$$\mathcal{R} :: \mathcal{LE} \rightarrow S$$

where $\mathcal{LE}$ is a list of logical events and S is either an Event Stream, i.e., $(At_1,t_1)...(A,t_n)$ or a Well Grounded RDF Stream, i.e., (triples(At_1),t_1)... (triples(At_n),t_n).
the abstract syntax of the return clause is

$$\mathsf{RETURN\ LE\ AS\ [EVENT|RDF]\ STREAM}$$

where L is a list logical events [E_1,...,E_n] defined in a OBEP query.

3.3 Query Definition

Building on the previous definitions, we can finally define an OBEP query.

Definition 9. *An OBEP query is a tuple $< S^D, K, \mathcal{E}, \mathcal{CE}, \mathcal{R}, ET >$ where:*

- *S^D is a Well Grounded RDF Stream as defined in Definition 4.*
- *K is a static ABox.*
- *$\mathcal{E}$ is a set of Logical Events axioms defined as in Definition 6.*
- *$\mathcal{CE}$ is a set of Composite Events defined as in Definition 7.*
- *$\mathcal{R}$ is a return clause.*
- *ET is the set of evaluation time instants.*

We can now define an evaluation function considering the query definition.

[7] https://www.w3.org/TR/rdf-sparql-query/#QueryForms

Definition 10. *Given an OBEP query $Q = < S^D, K, \mathcal{E}, \mathcal{CE}, \mathcal{R}, ET >$ we define the evaluation of $\mathcal{E}$ and $\mathcal{CE}$ over S^D and K, at time $t \in ET$, with the return clause $\mathcal{R}$ as:*

$$eval(S^D, \mathcal{E}, \mathcal{CE}, \mathcal{R}, t)$$

Applying the evaluation semantics of the Return Clause we obtain

$$eval(S^D, \mathcal{E}, \mathcal{CE}, \mathcal{R}, t) = \mathcal{R}(eval(S^D, \mathcal{E}, \mathcal{CE}, t))$$

According with Proposition 2, given that $\mathcal{E}$ is a static set of TBox axioms based on D, we can assert the following equivalence:

$$\text{eval}(\text{S}^D, \mathcal{E}, \mathcal{CE}, \text{t}) = \text{eval}(\text{S}^{\mathcal{E}}, \mathcal{CE}, \text{t})$$

Last but not least, we pose two conditions that allow to evaluate independently each composite event definition [7], i.e.,

$$\text{eval}(\text{S}^{\mathcal{E}}, \mathcal{CE}, \text{t}) = \bigcup_{\pi \in \mathcal{CE}} \text{eval}(\text{S}^{\mathcal{E}}, \pi, \text{t})$$

1. Composite Events must not have circular-dependencies, i.e., given $\text{H}_1 \leftarrow \text{L}_1 \in \mathcal{CE}$, $\nexists$ $\text{H}_2 \leftarrow \text{L}_2 \in \mathcal{CE}$, such that $\text{H}_1 \in \text{named}(L_2)$ and $H_2 \in \text{named}(L_1)$;
2. the Composite Event assertions set $\mathcal{CE}$ can be stratified to ensure a loop-free evaluation across partitions.

4 OBEP Semantics

In this section, we present the evaluation semantics of logical and composite event expressions. To this extent, we will exploit the following helper function:

- *explain* :: $\mathcal{T} \times \text{A} \times \text{C} \rightarrow \mathcal{G}$, where $\mathcal{T}$ is a set of TBox axioms, A is a set of ABox axioms w.r.t. $\mathcal{T}$ and C is a class concept. *explain* returns the set of RDF Graphs $\mathcal{G}$ with $G_i \in \mathcal{G}$ and axioms(G_i) a minimal subset of A such that axioms(G_i) $\models^{\mathcal{T}}$ C(c), where c is a an individual.

4.1 Complex Event Evaluation Semantics

Definition 11. ***Evaluation Semantics of Complex Event Expressions.*** *Given an event stream $S^{\mathcal{E}}$, a time instant t, and a Logical Event E we define*

$$[\![E]\!]^t_{S^{\mathcal{E}}}$$

as the evaluation of E at t over $S^{\mathcal{E}}$.

Although Event streams are defined with one timestamp for each pair (A,t), we adopted a two-point time semantics for the evaluation. This approach, introduced in [1, 2], allows formulating interval-based temporal operators.

Logical Events are the first building block. Their evaluation aims at asserting the logical equivalence between an atomic concept H denoting a logical event and a basic concept B defined in the TBox $\mathcal{E}$. We define K as the static, time independent, ABox describing background information.

$$[\![E]\!]^t_{S^\mathcal{E}} = \{ \text{(A,}t_k\text{,}t_k) \mid (\text{A}_k\text{,}t_k) \in \text{S}^\mathcal{E} \wedge t_k < t \wedge \exists c \in A \wedge \text{A}_k \cup \text{K} \models^\mathcal{E} \text{E(c)} \wedge \text{A} \in \text{explain}(\mathcal{E}, \text{A}_k \cup \text{K, E}) \}$$

Example 4. Let's consider the event stream $S^\mathcal{E}$ obtained from the Well-Grounded RDF Stream from Figure 1 w.r.t. the TBox of Listing 1.1, and the logical event expression of Listing 1.4. Let's also consider the static knowledge base K of Listing 1.2. We want to evaluate if the logical event BusyRoom occurs in $S^\mathcal{E}$ at t=8 considering the static knowledge base K.

$[\![BusyRoom]\!]^8_{S^\mathcal{E}} = \{$
({Room(rb) Person(Bob) Sensor(sb) armedWith(rb,sb) obsP[8](sb,Bob) },2,2),
({Room(ra) Sensor(sa) armedWith(ra,sa) ocd[9](sa,doorA) },2,2),
({Room(rb) Person(Bob) Sensor(sb) armedWith(rb,sb) obsP(sb,Bob) },3,3),
({Room(rb) Person(Alice) Sensor(sb) armedWith(rb,sb) obsP(sb,Alice) },7,7),
({Room(ra) Sensor(sa) armedWith(ra,sa) ocd(sa,doorA) },7,7)}

Composite Events are the second building block of the OBEP semantics. Their evaluation aims at asserting the logical equivalence between an atomic concept H denoting a logical event and a complex event expression E.

$$[\![H \textbf{ MATCH } E]\!]^t_{S^\mathcal{E}} = \{ (\text{A} \cup H(c), t_1, t_2) \mid (\text{A}, t_1, t_2) \in [\![E]\!]^t_{S^\mathcal{E}}, c \text{ is a named individual.} \}$$

Composite Event Expressions combine logical events according to the $\mathcal{L}$ operators. Their evaluation aims to check when a specific temporal relation occurs between two or more logical events.

$[\![\textbf{FIRST } E]\!]^t_{S^\mathcal{E}} = \{ (\text{A}, t_1, t_2) \mid (\text{A}, t_1, t_2) \in [\![E]\!]^t_{S^\mathcal{E}} \wedge \nexists (\text{A}', t_3, t_4) \in [\![E]\!]^t_{S^\mathcal{E}}$ such that $t_3 \leq t_4 < t_1 \leq t_2 \}$.

Example 5. (**cont'd**) $[\![$ FIRST $BusyRoom]\!]^t_{S^\mathcal{E}} =$
{(Room(rb) Person(Bob) Sensor(sb) armedWith(rb,sb) obsP(sb,Bob) },2,2)}

$[\![E_1 \textbf{ SEQ } E_2]\!]^t_{S^\mathcal{E}} = \{ (\text{A}_2, t_1, t_4) \mid (\text{A}_1, t_1, t_2) \in [\![E_1]\!]^t_{S^\mathcal{E}} \wedge (\text{A}_2, t_3, t_4) \in [\![E_2]\!]^t_{S^\mathcal{E}} \wedge t_1 \leq t_2 < t_3 \leq t_4 \}$.

Example 6. (**cont'd**) $[\![$ BusyRoom SEQ FreeRoom $]\!]^t_{S^\mathcal{E}} = \{$
(Room(rb) Person(Bob) Sensor(sb) armedWith(rb,sb) obsN(sb,nobody) },3,4),
({Room(rb) Person(Alice) Sensor(sb) armedWith(rb,sb) obsP(sb,Alice) },6,7),
({Room(ra) Sensor(sa) armedWith(ra,sa) ocd(sa,doorA) },6,7) }

$[\![\textbf{LAST } E]\!]^t_{S^\mathcal{E}} = \{ (\text{A}, t_1, t_2) \mid (\text{A}, t_1, t_2) \in [\![E]\!]^t_{S^\mathcal{E}} \wedge \nexists (\text{A}', t_3, t_4) \in [\![E]\!]^t_{S^\mathcal{E}}$ such that $t_1 \leq t_2 < t_3 \leq t_4 \}$.

Example 7. (**cont'd**) $[\![$ LAST BusyRoom $]\!]^t_{S^\mathcal{E}} = \{$
({Room(ra) Sensor(sa) armedWith(ra,sa) ocd(sa,doorA) },7,7) }

$[\![E_1 \textbf{ AND } E_2]\!]^t_{S^\mathcal{E}} = \{ (\text{A}, t_1, t_2) \mid \text{A} = \text{A}_1 \cup A_2 \wedge (\text{A}_1, t_1, t_2) \in [\![E_1]\!]^t_{S^\mathcal{E}} \wedge (\text{A}_2, t_1, t_2) \in [\![E_2]\!]^t_{S^\mathcal{E}} \}$

[8] observesPerson
[9] observesClosedDoor

Example 8. (**cont'd**) $\llbracket$ BusyRoom AND FreeRoom $\rrbracket^t_{S^{\mathcal{E}}}$ = {
({ Room(ra) Sensor(sa) armedWith(ra,sa) observesClosedDoor(sa,doorA) Room(rb) Sensor(sb) Person(Bob) armedWith(rb,sb) observesPerson(sb,Bob) },3,3) }

$\llbracket E_1$ **DURING** $E_2\rrbracket^t_{S^{\mathcal{E}}}$ = { (A_1,t_1,t_2) | (A_1,t_1,t_2) $\in \llbracket E_1\rrbracket^t_{S^{\mathcal{E}}} \wedge$ (A_2,t_3,t_4) $\in \llbracket E_2\rrbracket^t_{S^{\mathcal{E}}} \wedge t_3 < t_1 \wedge t_2 < t_4$ }.

Example 9. (**cont'd**) $\llbracket$ OpenDoor DURING Exiting $\rrbracket^t_{S^{\mathcal{E}}}$ = {
({observesOpenDoor(sa,doorA) },3,3)}

4.2 Matching Negative Patterns

A common feature in Event Processing Languages [1, 4] is the detection of negative patterns, i.e., the negation of the detection at a given time instant. We define the **Negative Event Evaluation Semantics** as

$$\llbracket NOT\ E\rrbracket^t_{S^{\mathcal{E}}} = \{\ (\mathrm{A},\mathrm{t}_k,\mathrm{t}_k) \mid \mathrm{A} = \bigcup \mathrm{A}_k \setminus \mathrm{A}^U_k \wedge \mathrm{A} \neq \emptyset \text{ where}$$
$$(\mathrm{A}_k,\mathrm{t}_k,\mathrm{t}_k) \in \mathrm{S}^{\mathcal{E}} \wedge \mathrm{t}_k < \mathrm{t} \wedge \mathrm{A}^U_k = \bigcup \mathrm{A'}_k \text{ for each } (\mathrm{A'}_k,\mathrm{t}_k,\mathrm{t}_h) \in \llbracket E\rrbracket^t_{S^{\mathcal{E}}}\ \}$$

4.3 Guarded Complex Event Patterns

The assertion of a Complex Event H depends on the truth value of the logical expression L that characterizes it. In the following, we explain how to the truth value of L can be modified using Guards.

Definition 12. *A **Guard** is a boolean function that poses a condition to the evaluation of a complex event expression [4]. OBEP provides two types of guards: Data Guards and Time Guards.*

Data Guards or Filters are conditions to a composite event, within an event expression, evaluated w.r.t. the related physical event.

Since (Well-Grounded) RDF Streams are OBEP's underlying data model, we opted for SPARQL syntax and semantics to express data guards. Listings 1.6 shows an example of an OBEP query with filters.

```
EVENT Entering MATCH FreeRoom SEQ BusyRoom WITHIN (5 min)
    IF { EVENT BusyRoom { ?room :armedWith ?sensor . }
         EVENT FreeRoom { ?room :armedWith ?sensor . } }
```

Listing 1.6: OBEP query With Frame and Filters.

Given a Complex Event H $\leftarrow^G$ CExp, a filter is a SPARQL ASK query q, where restriction w.r.t. a logical event E $\in$ named(CExp), denotes (i) a *TriplesBlock* if E is used in any unary sub-expression; (ii) an *OptionalGraphPattern* if E is used in a Disjunction sub-expression; (iii) a *GraphGraphPattern* if E is used in any remaining sub-expression.

```
ASK
FROM NAMED :BusyRoom
FROM NAMED :FreeRoom
WHERE {  GRAPH :BusyRoom { ?room :armedWith ?sensor }
         GRAPH :FreeRoom { ?room :armedWith ?sensor } }
```

Listing 1.7: SPARQL ASK query equivalent to Listing 1.6 conditions.

Listing 1.7 shows the SPARQL query equivalent to the filter at Lines 2-3 in Listing 1.6. Having chosen SPARQL to specify filters, the evaluation scope of a filter is a SPARQL dataset that we name the Event Dataset (EDS).

Considering a H $\leftarrow^{G}$ CExp, the EDS, to evaluate the filter against, is built as follow: EDS = {(def,∅), (E_i,triples(A_k))...} where E_i ∈ named(CExp) and (A_k,t_k) ∈ $[\![E_i]\!]^t_{S^{\mathcal{E}}}$.

The EDS is populated with RDF graphs subsuming a logical event E_i ∈ named(CExp). It is worth to note that, since the evaluation semantics is defined using sets, a combinatorial blow-up w.r.t. the cardinality of named(CExp) is possible[10]. Nevertheless, we are using this representation only to explain the semantics of OBEP data guards. The most common implementation [4, 7] push the evaluation of data guards close to the related patterns, reducing drastically the number of combinations to evaluate.

Example 10. (**cont'd**) The data guards for the logical events detecting, respectively, *BusyRoom* at 8, i.e., (G_1={ :ra a :Room ; :ew :sa . :sa a :DoorSensor ; :obs [:closed a :Status] . },2,2), and *FreeRoom* at 8, i.e., (G_2={ :ra a :Room ; :ew :sa . :sa a :DoorSensor ; :obs [:open a :Status] . },3,3). are evaluated over the EDS that contains {(def,∅), (BusyRoom,G_1), (FreeRoom,G_2)}.

We define the complex event patterns clause evaluation in presence of data guards as follow:

$$[\![\text{ H MATCH E } \textbf{IF} \text{ F }]\!]^t_{ST} = \{ \; (\text{A}\cup H(c),\text{t}_1,\text{t}_2) \mid (\text{A},\text{t}_1,\text{t}_2) \in [\![\text{ H } \textbf{MATCH} \text{ E}]\!]^t_{S^{\mathcal{T}}} \wedge [\![F]\!]_{EDS} \neq \emptyset \; \}$$

To avoid meaningless filters, we must define the vocabulary that can be used in their formulation. We chose to consider all those classes and properties used in Logical Events, which is determined by the following function.

Given a complex event H $\leftarrow^{G}$ CExp, ∀ E_i ∈ named(CExp),

$$allowed(E_i) = \begin{cases} named(E_i) & if\ logical\ E_i \\ \bigcup_{E_j \in named(E_i)} named(E_j) & if\ composite\ E \end{cases}$$

Time Guards or Frames are functions that determine a portion of an ontology stream, restricting the scope of an evaluation, e.g., consider the windowed stream $S^{\mathcal{E}}_{[5,15)}$.

Complex event expressions are evaluated over a whole event stream $S^{\mathcal{E}}$. However, most of the use-cases pose strict requirements on responsiveness. Therefore, we define the following function, which restricts the evaluation to a finite portion of $S^{\mathcal{E}}$.

[10] Virtually, the EDS is populated by all the combination of events instances.

$$\text{WITHIN} :: \text{D} \times \text{S}^{\mathcal{E}} \rightarrow S^{\mathcal{E}}_{D}$$

where D indicates a Duration, $\text{S}^{\mathcal{E}}$ is an event stream and $S^{\mathcal{E}}_{D}$ is a windowed event stream. We define the complex event patterns clause evaluation in presence of guards as follow:

$$[\![\text{H } \textbf{MATCH} \text{ E } \textbf{WITHIN} \; \imath]\!]^{t}_{S^{\mathcal{E}}} = [\![\text{ H } \textbf{MATCH} \text{ E}]\!]^{t}_{S^{\mathcal{E}}_{[t-\imath,t)}}$$

Definition 13. *A* ***Guarded Complex Event*** *is an assertion $H \leftarrow^{G} E$ with $H \leftarrow E$ a Complex Event as in Definition 7 and G an optional guard as defined above.*

Notably, the semantics of the negation is not compatible with the one of data guards due to the impossibility of determining a vocabulary that is consistent with the incoming data.

5 Comparison with Existing Languages

In this section, we compare EP-SPARQL and RSEP-QL with OBEP. Table 3 summarizes the comparison, listing each language operators, reasoning support and first-class objects. Listings 1.8, 1.9, and 1.10 show the same query written with each language respectively.

EP-SPARQL [1] is a SPARQL 1.0 extension for complex event processing over RDF Streams. It supports the temporal operators listed in Table 3 and Allen's algebra relations. EP-SPARQL can answer queries over RDF streams in combination with a RDFS background knowledge. Its syntax extends SPARQL 1.0 (see Listings 1.8), while the execution model is based on ETALIS [2]. EP-SPARQL queries are translated into ETALIS rules, flattening events and Basic Graph Pattern (BGP) patterns to the same structure, i.e., predicates of the form *triple(s,p,o)*. Consequently, pattern-maching is forced to happen at attribute level, and EP-SPARQL can handle seamlessly BGP evaluation and event detection. However, this approach drastically reduces ETALIS expressiveness since EP-SPARQL events are not first-class objects but triple predicates [15].

RSEP-QL [8] extends RSP-QL [9] with event detection operators (see Table 3). RSEP-QL pattern matching is based on basic event patterns (BEP), which are defined extending BGP contextually to a time-preserving window operator,

	Operators	Allen's Algebra	Reasoning	Policies	FCO
EP-SPARQL	SEQ, O-SEQ, EQUALS, O-EQUALS, AND	Yes	RDFS	r,c,u	triples
RSEP-QL	FIRST, LAST, SEQ	No		n,r,c,u	BGP
OBEP	SEQ,FIRST, LAST,OR,AND, NOT	Yes	DL	u	Events

Table 3: OBEP vs EP-SPARQL vs RSEP-QL. **Legend**: O-*=Optional, r=recent, n=naive,c=chronological,u=unrestricted; FCO: first-class objects.

named window function. BEP are labeled to allow their reuse, but labels do not have a well-defined semantics. Event-pattern operators in RSEP-QL work according to SPARQL operators and, thus, we can conclude that events are not first-class object, but an abstraction build on top existing RSP-QL features (BGP and Windows Operators).

Operators. OBEP, EP-SPARQL and RSEP-QL employ the same temporal model based on two timestamps. Although EP-SPARQL operational semantics is based on ETALIS, RSEP-QL showed we could redesign it using RSP-QL primitives. Therefore, the languages are substantially similar in terms of operators.

Selection or Consumption policies. RSEP-QL is the most expressive and fully captures EP-SPARQL behaviors. OBEP does not specify any policy, and it adopts an unrestricted selection which is the default for EP-SPARQL.

Syntax. As shown in the following examples, the OBEP approach results in better organized queries than existing ones. Let's consider the following query, a simple extension of an example from [2]: *Provide all the ranking augments that are followed by a stock price increase, and all the ranking decrements that are followed by a stock price decrease.*

EP-SPARQL query, in Listing 1.8, requires the use of a UNION pattern, to represent the alternatives cases. The query is not unmanageable, but it is easy to show that it becomes too complicated when event definitions become more complicated than one triple. Listing 1.9 reports only a sub-example of the same query translated in RSEP-QL. Indeed, RSP-QL does not support UNION patterns, and we are forced to create a query-network that produces the results. Finally, Listings 1.10 shows how OBEP handles the use-case. Although OBEP is as succinct as EP-SPARQL, the query shows a more organized structure, providing a clear separation between event definitions and processing. Moreover, it allows sharing, and even extending event definitions across queries, which was not possible in EP-SPARQL and only partially in RSEP-QL.

6 Conclusion & Future Work

In this paper, we studied the foundation of Ontology-Based Event Processing (OBEP) [18]: an approach for event definition and detection on RDF Streams.

In this paper, (i) we defined OBEP's data and query models, and (ii) we explained the evaluation semantics of OBEP's operators[11]. Moreover, (iii) we showed why OBEP is alternative to existing RSP approaches for event detection, i.e., EP-SPARQL, and RSEP-QL.

Future work for OBEP comprises the introduction of synthesized events, enabling simple analytical queries. Moreover, we aim at studying OBEP performance systematically, investigating how different DL impact the performance. Last but not least, we aim at combining OBEP and existing RSP approaches towards a unified language that tames variety and velocity as well as reconciles analytics and event detection rules [6].

[11] An extended version of this paper, with more examples and all the operators semantics is at `https://github.com/riccardotommasini/obep`

```
SELECT ?comp ?r2
WHERE {{{
      {?comp :rank ?r1 } SEQ {?comp :rank ?r2 }
      FILTER (?r1 < r2) }
      SEQ
      {{ ?comp :price ?p1 . } SEQ { ?comp :price ?p2 . }
      FILTER (?p1 < ?p2*0.5) } }
 UNION
    {{ {?comp :rank ?r1 } SEQ {?c :rank ?r2 }
         FILTER (?r1 > r2) }
     SEQ { { ?comp :price ?p1 . } SEQ { ?comp :price ?p2 . }
        FILTER (?p1 > ?p2 *0.5) }}
FILTER ( getDURATION() < "P1H"^^xsd:duration)
```

Listing 1.8: EP-SPARQL.

```
REGISTER <StockChange> CONSTRUCT { ?company :price1 ?p1 ; price2 ?p2. }
FROM NAMED :S WIN [LND 1H] AS :w1
EVENT ON :w1 { ?company :price ?p1 . } AS Price1
EVENT ON :w1 { ?company :price ?p2 . } AS Price2
WHERE  {  MATCH { Price1 SEQ Price2} }

CONSTRUCT { ?company :price ?p1 . }
FROM NAMED :S WIN [LND 1H] AS :w1
FROM NAMED :StockRaise WIN [LND 1H] AS :w2
EVENT ON :w1 { ?company :rank ?r . } AS RankChange
EVENT ON :w2 { ?company :price1 ?p1 ; :price2 ?p2 . } AS PriceFall
WHERE  {  MATCH { RankChange SEQ PriceFall}
          FILTER (?p1 < ?p2 * 0.5)}
```

Listing 1.9: RSEP-QL (Partial)

```
EVENT RankChange AS rank some .
EVENT StockPrice AS price some .

EVENT UpRank MATCH RankChange AS R1 SEQ RankChange AS R2 WITHIN 1H
 IF { EVENT R1 { ?company :rank ?r1 . }
      EVENT R2 { ?company :rank ?r2 . }
      FILTER ( ?r1 < ?r2 ) }

EVENT DownRank MATCH RankChange AS R1 SEQ RankChange AS R2 WITHIN 1H
 IF { EVENT R1 { ?company :rank ?r1 . }
      EVENT R2 { ?company :rank ?r2 . }
      FILTER ( ?r1 > ?r2 ) }

EVENT UpPrice MATCH StockPrice AS S1 SEQ StockPrice AS S2 WITHIN 1H
 IF { EVENT S1 { ?company :price ?p1 . }
      EVENT S2 { ?company :price ?p2 . }
      FILTER ( ?p2 < ?p1 * 0.5) }

EVENT DownPrice MATCH StockPrice AS S1 SEQ StockPrice AS S2 WITHIN 1H
 IF { EVENT S1 { ?company :price ?p1 . }
      EVENT S2 { ?company :price ?p2 . }
      FILTER ( ?p2 > ?p1 * 0.5) }

EVENT Alert MATCH
  (UpRank SEQ UpPrice) OR (DownRank SEQ DownPrice) WITHIN 1H
RETURN Alert AS RDF STREAM
```

Listing 1.10: OBEP.

References

1. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. pp. 635–644 (2011)
2. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in ETALIS. Semantic Web 3(4), 397–407 (2012)
3. Baader, F.: The description logic handbook: Theory, implementation and applications. Cambridge university press (2003)
4. Chakravarthy, S., Mishra, D.: Snoop: An expressive event specification language for active databases. Data Knowl. Eng. 14(1), 1–26 (1994)
5. Chen, J., Lécué, F., Pan, J.Z., Chen, H.: Learning from ontology streams with semantic concept drift. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 957–963 (2017), `https://doi.org/10.24963/ijcai.2017/133`
6. Cugola, G., et al.: Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. 44(3), 15 (2012)
7. Cugola, G., Margara, A.: TESLA: a formally defined event specification language. In: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS, Cambridge, United Kingdom (2010)
8. Dell'Aglio, D., Dao-Tran, M., Calbimonte, J., Phuoc, D.L., Della Valle, E.: A query model to capture event pattern matching in RDF stream processing query languages. In: Knowledge Engineering and Knowledge Management - 20th International Conference, EKAW, Bologna, Italy (2016)
9. Dell'Aglio, D., Della Valle, E., Calbimonte, J., Corcho, Ó.: RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. Int. J. Semantic Web Inf. Syst. 10(4), 17–44 (2014)
10. Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook 1, 59–83 (2017), 1-2
11. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible sroiq. Kr 6, 57–67 (2006)
12. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
13. Paschke, A.: Eca-ruleml: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. CoRR abs/cs/0610167 (2006)
14. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34(3), 16:1–16:45 (2009)
15. Phuoc, D.L., Dao-Tran, M., Pham, M., Boncz, P.A., Eiter, T., Fink, M.: Linked stream data processing engines: Facts and figures. In: The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part II. pp. 300–312 (2012)
16. Ren, Y., Pan, J.Z.: Optimising ontology stream reasoning with truth maintenance system. In: Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011. pp. 831–836 (2011), `http://doi.acm.org/10.1145/2063576.2063696`
17. Stuckenschmidt, H., et al.: Towards expressive stream reasoning. In: Semantic Challenges in Sensor Networks, 24.01. - 29.01.2010 (2010)
18. Tommasini, R., Bonte, P., Della Valle, E., Mannens, E., De Turck, F., Ongenae, F.: Towards ontology based event processing. In: OWLED2016, the International Experiences and Directions Workshop on OWL (2016)
19. Zemke, F., Witkowski, A., Cherniack, M., Colby, L.: Pattern matching in sequences of rows. Tech. rep., Technical Report ANSI Standard Proposal (2007)