Towards Ontology Based Event Processing

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

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

Abstract. The rapid change and heterogeneity of today's generated data calls for real-time decision making systems that can cope with the presented heterogeneity. In this paper, we present an Ontology Based Event Processing system that bridges the gap between ontology-based reasoning and event processing. We propose both a language and an architecture to perform event processing over abstract ontology concepts. This allows to perform efficient temporal reasoning, while the high-level ontological definitions reduce the need for knowledge of the underlying data structure in complex domains.

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

1 Introduction

In domains like Social Media, Financial Markets and Internet of Things (IoT), information is traditionally represented as data streams, i.e. unbounded sequences of data, or events , i.e. notifications about happened facts. Stream Reasoning (SR) [5] investigates how Semantic Web and Stream Processing technologies can be combined to make decision making systems work in real-time, across multiple data sources. SR investigates how to exploit the time ordering of data streams to perform deductive and temporal reasoning on the fly.

In order to clarify this domain, consider the following example: We are interested to identify the presence of fire in a room, but there is no way to detect it directly. Instead, the room contains sensors to detect the presence of smoke and measure the temperature. In this case, a data stream is a timestamped sequence of numbers representing the average temperature in the room; while an event is a notification about the detection of smoke. The data and events arise from different types of sensors. This heterogeneity impedes to perform queries across these data sources. Another obstacle comes from the domain complexity. In presence of fire the temperature will be higher, but how can we distinguish abnormal temperatures from normal ones? And, what if we had different rooms? This kind of information represent background knowledge that our decision making system has to combine with live data, in order to obtain an answer. Finally, assuming that we finally detected both smoke and abnormal temperature events, we had to relate them in time.

The presented example calls for an approach that solves data variety, that combines data with background knowledge, that deducts related information and operates temporal reasoning combining data streams from sensors and events. We name this approach Ontology-Based Event Processing (OBEP). For the best of our knowledge, there is no approach in the SR state of the art that tries to do so. Temporal extensions of deductive reasoning extends the ontological language with time relations and, thus, easily diverges into intractability. Semantic Complex Event Processing is limited to a semantic description of events and does not focus on the processing.

In this paper, we propose an approach for OBEP that operates the event processing a-posteriori above high level concepts deduced through deductive reasoning, but without including time relations at ontological level. The contribution of this work are: (i) a requirement analysis for an OBEP system to satisfy; (ii) a syntax named DELP, i.e. Description Logic Event Processing, to express information needs as the one presented in the example; (iii) an architecture that bridges the gap between event processing to capture temporal relations and event descriptions based on Semantic Web technologies and; (iv) a prototype that proves the feasibility of the approach.

The rest of the paper is structured as follows: Section 2 describes the related works. Section 3 describes the use case that is used throughout the paper. Section 4 introduces the Description Logic Event Processing (DELP) language we constructed, while Section 5 describes our OBEP system that implements the abstracted event processing. Section 6 concludes the paper and elaborates on the future research directions.

2 Background & Related Works

In this section we present the background knowledge required to understand the content of the paper and the relevant related works.

Stream Processing engines are systems capable to process potentially infinite sequences of data. Two main approaches exits to this extent:

- Data Stream Management Systems (DSMS) extend Data Base Management Systems by introducing stream-to-relation operators, e.g. Windows, that allow the transition between streaming and static data. Queries are continuously evaluated over finite portions of the data streams selected by the means of these operators.
- Complex Event Processing (CEP) engines exploit time-aware operators to detect patterns over infinite sequences of incoming events. The user specifies reaction rules that are concerned with the invocation of actions in response to events and actionable situations. These rules specify a pattern over the incoming data, e.g. A followed-by B, by using a declarative query language. Such a pattern is usually validated with a finite state machine. Therefore, the final complexity is at most polynomial in time and space.

Some stream processing engines offer declarative query language to operate with data streams. The event processing language (EPL)[1] is the most relevant one and it allows to

(i) write window-based continuous queries to process data streams; (ii) define simple events or compositions of them (i.e. complex events) (iii) treat events as first class citizens, i.e. the operators have direct influence on the events.

Semantic technologies such as RDF, OWL and SPARQL have been used for data integration in the IoT domain [2] and Semantic Complex Event Processing (SCEP) [9].

An example of the former is MASSIF [4], i.e. an event-based semantic-enabled IoT platform consisting of multiple semantic reasoning services each fulfilling a distinct reasoning task. These services can collaborate on a high level by subscribing to the Semantic Communication Bus (SCB) and indicating the high level concepts they are interested in. The platform follows the notion of high level events, however, it does not support any temporal reasoning between these events.

An example of The SCEP is the work of Taylor et al [9], i.e. an ontology and a system for complex event specification that, in combination with reasoning techniques, simplify the rule definitions of a target complex event processing language (e.g. EPL), eliminating the need of address manually the domain complexity. To this extent, the ontology contains language constructs and operators, e.g., *seq*, as properties and classes. This approach generalizes the query definition task enabling interoperability between different event processing engines, but it does not extend the semantics of the target query language nor does it propose a unified syntax for it.

In the SR state-of-the-art, RDF Stream Processing (RSP) engines combines semantic technologies and stream processing to perform continuous querying or complex event processing [1] over streams encoded into RDF. EP-SPARQL [1] is the most relevant work w.r.t ours, because it extends SPARQL 1.0 with event processing operators, i.e., *seq, equals, optionalseq, and equalsoptional*[2]. Event processing and SR is enabled over RDF Basic Graph Pattern (BGP). Complex events are defined as BGPs combined with event processing operators. As this is similar to the UNION or OPTIONAL operators in SPARQL, events are not first class citizens. Since the events are defined through BGPs, it can be devious to construct advanced event processing patterns.

Finally, temporal extension of deductive reasoning approaches such as Description Logics are worth to mention. They include time relations at ontological level, but this easily diverges into intractability and limiting the possible entailments [6].

In summary, state-of-the-art solutions in the stream processing context successfully model time relations but lack to address the data variety and the domain complexity. Semantic technologies can be used to describe these extents, but existing approaches either lack to provide an unified syntax to model the full

[1] `https://docs.oracle.com/cd/E13157_01/wlevs/docs30/epl_guide/overview.html`

[2] The semantics of these operators is similar to a left, right or full -join but their selectivity depends on how the constituents are temporally related.

processing [9] or have limited expressiveness and do not treat events as first class citizens [1]. Finally, temporal logics are limited due to the hurdle of including time within the reasoning algorithms.

3 Use Case

In this section we introduce a simple use case that we will use in the reminder of the paper to explain our contributions.

A company wants to deploy an intelligent system to detect dangerous situations. Internally, they distinguish between three classes of conditions:

- Hazardous, i.e., situations that are dangerous for the company assets, e.g., fire or floods,
- Risky, i.e., situations that are dangerous for the complete business, e.g., information leaks or unauthorized access to restricted areas, and
- Unsafe, i.e., situations that are directly dangerous for people, e.g., fire or gas leaks.

For each dangerous situation class, different alarms are defined (e.g., sound and lights), alternative escape plans are organized and different authorities are responsible for handling the situation.

The company is interested in monitoring Unsafe situations within their buildings and the surrounding areas. To this extent, sensors for smoke detection, temperature, humidity and air quality monitoring are deployed within the building into a wireless sensor network. To monitor the surrounding areas a public infrastructure provided by the local government is available through web APIs.

For the remainder of the paper, we will provide examples of the Unsafe situation Fire Detection. As explained in the Section 1, there is no direct way to sense fire, but we can assume its presence through the detection of smoke and abnormal temperature measurements within the same time interval. Many challenges arise to define such a simple rule:

(i) *Data Integration*: How can the proprietary data and those coming from external APIs be combined?
(ii) *Domain complexity*: How can we decide if the detected temperature is abnormal?
(iii) *Temporal Relation*: How do we model the temporal relation between smoke events and abnormal temperature so we can infer the presence of fire?

4 Ontology-Based Event Processing Language

In this section, we introduce our first contribution DELP, a syntax for Description Logic Event Processing. DELP design is based on the definition of the following requirements elicited on the challenges presented in Section 3.

(R1) Semantic Event Representation [9]: this allows the integration of multiple heterogeneous sources (a) and derivation of implicit data in combination with background knowledge (b).

(R2) Event Processing [1]: this allows to combine high level ontological concepts capturing the temporal dependencies and build complex events.
(R3) First Class Citizens Events, i.e., creation and direct manipulation with language operators (e.g. pattern matching) should be possible.
(R4) Filtering and Joining: The former allows to remove irrelevant events, while the latter allows to combine events over multiple event streams to achieve intelligent decision making.

In Section 4.2, we show each challenge should be tackled for an OBEP system, finally in Section 4.3 we present the grammar of DELP and how it fulfills the requirements above.

4.1 Semantic Event Representations

In our running example, we want to derive abnormal temperature and measurements and combine them with smoke detection events. These need are captured by challenges (i) and (ii), that call for a semantic representation of events. This need becomes clear when we analyze the domain complexity, e.g. temperature normality is different in different building areas, e.g., elevator are colder than server rooms.

Static Information Integration systems as Ontology Based Data Access systems solve this circumstances by the means of an integrated conceptual model (ICM). The ICM enables query answering across heterogeneous data sources by the means of a common vocabulary formally specified with an ontological language, e.g. DL or OWL. The ICM of our example currently contains axioms from (1) to (5).

$$SmokeDetectionEvent \equiv \exists hasContext.(\exists observedProperty.Smoke) \quad (1)$$

$$TemperatureEvent \equiv Observation \sqcap (\exists observedProperty.Temperature) \quad (2)$$

$$AbnormalTemperatureEvent \sqsubseteq TemperatureEvent \quad (3)$$

$$ElevatorAbnormalTemperatureEvent \sqsubseteq AbnormalTemperatureEvent \sqcap (\exists observationResult.[hasValue_{>40}]) \sqcap (\exists hasLocation.Elevator) \quad (4)$$

$$ServerRoomTemperatureEvent \sqsubseteq AbnormalTemperatureEvent \sqcap (\exists observationResult.[hasValue_{>20}) \sqcap (\exists hasLocation.ServerRoom) \quad (5)$$

Data integration requires a generic data model. RDF commonly used by the Semantic Web community to overcome heterogeneity of static data. In our case, RDF is enough to represent the background knowledge but not to represent streams, which require RDF Streams (see Section 2).

Last but not least, the ICM, if combined with a reasoners, allows to exploit background knowledge to derive information that is only implicit described in the data, as the axioms (4) and (5) show.

Deciding the entailment to use for representing the ICM is a domain specific problem and a trade-off with the final system complexity. One may argue the need of a very expressive ontological language such as OWL 2 DL, that allows us to define events in a generic and concise manner and it enables to create a truly abstracted view over the events by the means of DL reasoning. Fragments like OWL RL, DL-lite, or EL++ have been shown to be interesting for Stream Reasoning use cases. At this stage, we do not discuss which restriction DELP should include. In order to express meaningful examples w.r.t. our use case we opted for OWL 2 DL[3] , postponing a deep complexity study for the future works.

4.2 Capturing Time Relations

In our running example, the central part represent the time relation between abnormal temperature and smoke. This need is captured by challenge (iii), that calls for event processing operators. In practice, we need to explain simple temporal pattern such as *seq*, combined with modifiers that provide enough expressiveness to capture the entire domain complexity, e.g. *not*.

Regarding time, we assume a point-based time semantics [3] for events; an event e as a pair (G,t), where G is an RDF graph containing the event statements and t is the associated timestamps. A partial ordering is established among events, i.e. events can occur at the same timestamps. Regarding the event processing, we consider the following time-aware operators:

- *seq*: (G_1, t_1) and (G_2, t_2), returns true iff the events occur and $t_1 > t_2$;
- *and*: (G_1, t_1) and (G_2, t_2), returns true when both the events occur regardless their ordering;
- *or*: (G_1, t_1) or (G_2, t_2), returns true iff at least one of the events occur;

and the following modifiers:

- *every*, forces the re-evaluation of the qualified according to its positive evaluation of the sub
- *within*, which limits the validity of the pattern by constraining its evaluation into time boundaries; and
- *not*, which negates the truth value of a pattern[4].

Notably, in the state of the art, none of the existing solution implements all these operators.

4.3 Description Logic Event Processing

In this section, we finally explain how the event processing operators (see Section 4.2) are used in combination with ontological concepts.

[3] https://www.w3.org/TR/owl2-direct-semantics/

[4] Not can be used only as a combination of other patterns

In our example, we are interested in abnormal temperature and smoke sensor readings to detect fire. We saw in Section 4.2 that semantic event representation (R1) is possible at in the ICM. Alternatively, high level events can be specified within a DELP query, by the means of the EventDecl clause (see Listing 1.4). Listing 1.1 is an example of event declaration in DELP. The Manchester syntax[5] is chosen for two reasons: it is conciser than RDF and highlights the idea of specifying events using high level abstractions. Moreover, it was combined already with SPARQL in the past [8].

```
EVENT :OfficeAbnormalTemperaturEvent subClassOf
  AbnormalTemperaturEvent
        and (observation_result some (hasValue (hasDataValue >= 40)))
        and (hasLocation some Office))
```

Listing 1.1: Event Declaration for office abnormal temperature in DELP

Event defined through this clause are added to the TBox of ontology the reasoner uses for the inference process. Each of the defined events in DELP is translated to OWL class expressions. The translation is straight forward, since the event definition is based on the DL Manchester syntax. For example, the Office Abnormal Temperature definition in Listing 1.1 is translated to:

$$OfficeAbnormalTemperaturEvent \sqsubseteq AbnormalTemperatureEvent \sqcap (\exists observationResult.[hasValue_{>40}]) \sqcap (\exists hasLocation.Office) \quad (6)$$

DELP exploits the time-aware operators included in Section 4.2 and Listing 1.2 shows how Fire detection can be defined exploiting the temporal relation between a SmokeDetectionEvent and AbnormalTemperaturEvent.

Event processing over high level concepts (R2), an example of which is available in Listing 1.2, is enabled by the sub-clause PatternExpr of the PatternDecl clause. The definition of event patterns relies on user-defined ontological concepts or those already existing in an ontology.

```
NAMED EVENT :FireEvent {
      MATCH :AbnormalTemperaturEvent SEQ :SmokeDetectionEvent WITHIN (5m)
}
```

Listing 1.2: Event Declaration for fire, based on temperature and smoke, in DELP.

```
NAMED EVENT :FireEvent {
      MATCH :AbnormalTemperaturEvent SEQ :SmokeDetectionEvent[6] WITHIN (5m)
      IF {
          EVENT :AbnormalTemperaturEvent { ?tmpSnsLoc :hasValue ?v}
          EVENT :SmokeDetectionEvent { ?smkSnsLoc :hasValue ?v;
                ?smokeObs ssn:observationResult ; :hasValue ?smokeLevel
                FILTER (?smokeLevel == "3"^^xsd:integer)
                }
      }
}
```

Listing 1.3: Example of event pattern with filters (R4)

[5] https://www.w3.org/TR/owl2-manchester-syntax/

[6] We assume this event is already defined in the ontology.

Last but not least, the IFDECL clause enables to express filters and joins over RDF Streams. Using a SPARQL-like syntax, the user can specify a basic graph pattern to match for each event, e.g., EVENT :AbnormalTemperaturEvent in Listing 1.3, and joins that exploit a name-based notation, i.e., variables with the same name obtain the same binding (e.g., variable ?v in Listing 1.3). Filters are specified using the SPARQL 1.1 Filter clause e.g., variable ?smokeLevel in Listing 1.3.

Finally, Listing 1.4 describes sub-portion the DELP grammar, the full one is available at `http://bit.ly/2bURXUt`. Due to the lack of space, we omitted those parts that relies on other grammars, in particular: The EVENTDECL clause allows definition of events as first class citizens; it relies on the classes formulation typical of Manchester Syntax. An example of this is available in Listing 1.1. The CONSTRAINT clause allows the specification of filters; it relies on the SPARQL 1.1 grammar; an example of this is available in Listing 1.3. The user can specify time relations over semantic event declarations using the MATCH clause. Notably, the optional keyword **NAMED** works differently from SPARQL 1.1. It indicates which events the user is interested to select for the retrieval of the underlying RDF graph.

```
EventClause -> [NAMED] 'EVENT' EventIRI (EventDecl | PatternDecl)
EventDecl  ->  Follows Manchester Syntax Grammar[7]
PatternDecl -> 'WHEN' PatternExpr [IFDecl]
PatternExpr -> 'MATCH' FollowedByExpr [WITHIN TimePeriod ]
TimePeriod -> 'INTEGER' (ms | s | m | h | d | w)
FollowedByExpr -> OrExpr (([ 'NOT'] 'SEQ') OrExpr)*
OrExpr -> AndExpr ('OR' AndExpr)*
AndExpr -> EveryOrNotExpr ( 'AND' EveryOrNotExpr)*
EveryOrNotExpr -> [ 'EVERY' | 'NOT' ] ( EventIRI ['AS' EventAltIri]
                | ( PatternExpr ) )*
IFDecl -> IF '{' 'EVENT' (EventIRI | Var) FilterExpr '}'
FilterExpr -> '{' ( BGP | 'FILTER' Constraint )*'}'
Constraint -> Follows the SPARQL 1.1 Grammar[8]
```

Listing 1.4: Ontology-Based Event Processing Language Grammar

5 Ontology-Based Event Processing Architecture

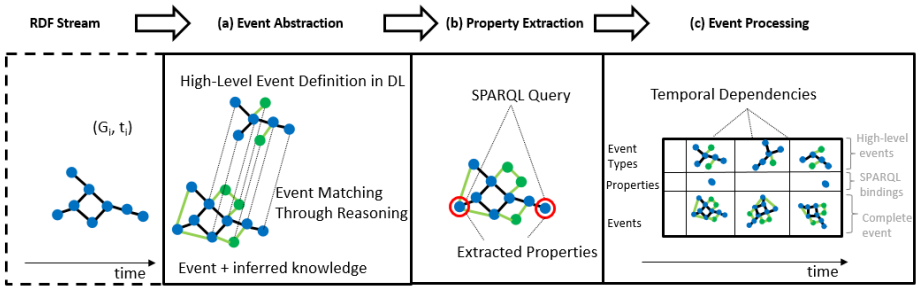

Fig. 1: Overview of the ontology-based event processingarchitecture

[7] `https://www.w3.org/TR/owl2-manchester-syntax/#description`
[8] `https://www.w3.org/TR/rdf-sparql-query/#rConstraint`

In this section, we describe a system architecture for an OBEP system that supports DELP syntax.

Figure 1 shows three different layers, each of which addresses a specific part of the processing to go from RDF Streams to results of a DELP query. As anticipated in Section 4.3, we assume incoming events as a pair (G,t) where G is an RDF Graph and t is a timestamp (RDF Stream in Figure 1).

Building on this assumption, Layer (a) is responsible for inferring high level concepts by applying reasoning over the incoming events; Layer (b) is responsible for identifying and extracting, from the underlying RDF graph, those properties that are relevant for filtering and joining, as specified in the query; last, but not least, Layer (c) applies event processing over the abstracted events as well as filtering and joining using the extracted properties. In the following paragraphs, each layer is described in detail.

To better understand how each layer behaves, we continue on our running example. We want to capture the temporal relation between abnormal temperature and smoke in order to detect fire, but we need to ensure that the smoke detection and the abnormal temperature measure belong to the same room. In Listing 1.5, this requirements are translated into a time relation and a join condition: the variable ?v is used for the AbnormalTemperaturEvent and the SmokeDetectionEvent.

```
NAMED EVENT :FireEvent {
  MATCH :AbnormalTemperaturEvent -> :SmokeDetectionEvent WITHIN (5m)
  IF {
        EVENT :AbnormalTemperaturEvent { ?tmpSnsLoc a :Location.
                                         ?tmpSnsLoc :hasValue ?v}
        EVENT :SmokeDetectionEvent { ?smkSnsLoc a :Location.
                                     ?smkSnsLoc :hasValue ?v}
    }
}
```

Listing 1.5: Event Declaration for fire, if the smoke and temperature are sensed in the same location.

The incoming RDF graphs are added to the ABox, processed by the reasoner, and then removed. This process is show in Figure 1.a. DL reasoning is utilized, together with ontological definition of events, to materialize the incoming RDF graphs. When the reasoner, after a realization step, infers one of the defined high level events, these are forwarded to the next layer that can perform event processing over high level abstractions.

DELP allows the specification of filters and joins over the defined events. However, performing joins or filters requires to compare the values of those variables expressed in the DELP query. Which means access to the underlying RDF graph of high level ontological concepts that DELP targets. An additional SPARQL-querying layer, shown in Figure 1.b, is added in order to reach the underlying RDF graph that the high level event definition implies and extract the variables required for joining or filtering.

The translation from DELP filters to SPARQL queries is again straight forward. Listing 1.6 shows one of the required queries for the property extraction of the SmokeDectectionEvent in our example. For joins, the variable value must be the same for all the events sharing a variable; filters should positively validate a given conditional expression (e.g. lower than a specified threshold). Once the

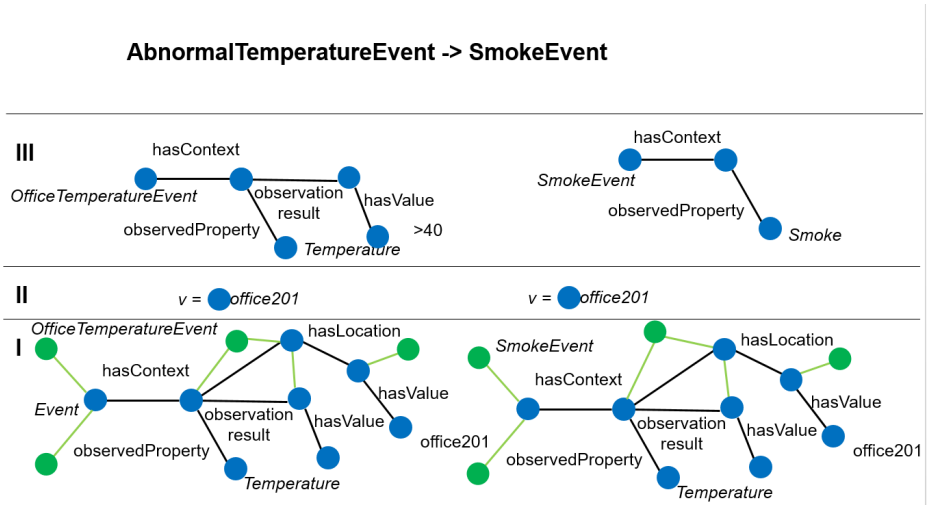

Fig. 2: Event processing over high level events.

query is executed, the variable bindings are added to the event as properties, maintaining the naming convention. If no properties need to be extracted and no additional filtering is required, this step can be omitted.

```
SELECT ?tmpSnsLoc ?v
WHERE { ?tmpSnsLoc a :Location; :hasValue ?v }
```

Listing 1.6: Translated SPARQL query for the property extraction based on the defintion in Listing 1.5 for the SmokeDetectionEvent

The last layer in our proposed architecture is the one responsible for the actual event processing; it corresponds to Figure 1.c.

In our example, SmokeDetectionEvent and AbnormalTemperatureEvent are matched. Figure 2 zooms in Figure 1.c and shows the structure of the events once they reach the event processing layer for our running example: (Figure 2.I) the materialized events that therefore contain both explicit data (Blue) and those which have been inferred (Green); (Figure 2.II) the previously extracted values for variables involved in filters or, in this very case, joins. (Figure 2.III) the high level event definition, represented as an RDF graph to maintain a coherent notation.

Assuming such a layered data structure, the pattern matching can be translated into a target CEP language that provides filtering and joining using a name-based notation such as EPL. Listing 1.7 shows an example of this translation related to the fire detection example.

```
select * from pattern
[every a=AbnormalTemperaturEvent -> b=SmokeDetectionEvent(v=a.v)
 where timer:within(5 min)]
```

Listing 1.7: Event Declaration for fire, translated to EPL

Building complex event structure is the goal of both CEP and SCEP systems. Therefore, it is worth discussing how complex events are provided to the user in case of positive pattern matching. At current stage, DELP does not include the specification of composed events explicitly. This is because it is hard to combine high level event description with their low level construction and we leave this as future work. Since event composition is crucial in event processing, we opt for a conservative solution and we define the complex event as the union of the underlying RDF graphs. The union is used since the event processor will only return values when the operator turned true. For example if E_1 has been detected and E_2 not, then E_1 OR E_2 will return true with E_2 as an empty collection.

Last but not least, we implemented an OBEP proof-of-concept system[9] containing the following technologies: the HermiT reasoner [7] for event abstraction in the first layer; Jena ARQ[10] for the property extraction of the underlying RDF graph in the second layer and the Esper engine[11] to perform the event processing on the high level events in the third layer.

6 Discussion and Conclusion

In this paper, we presented a first step towards ontology-based event processing. We designed an approach that contributes to the state-of-the-art of stream reasoning with a requirement analysis; a syntax for Description Logic Event Processing, i.e. DELP; a three-layered architecture for an OBEP system that supports the proposed DELP syntax and fulfills our requirements; and a proof-of-concept implementation of a system.

Table 1 summarizes the differences and similarities between the related works mentioned in Section 2 and our approach for OBEP. This table highlights the novelty of the proposed system through the requirements that we presented in Section 4. Our approach combines semantic event declaration (R1.a) and event processing (R2). It also allows to compute temporal inference over the high-level concepts outputted by a deductive reasoning process. This is different from approaches that extend the ontological language to perform temporal inference, because they have to choose between either small entailments or intractability. DELP implements all the typical event processing operator (R3), while the other approaches focus on a subset. In particular, we include the *not*, which allows the definition of more expressive patterns. The final system complexity is composed by two layers, i.e. deductive reasoning and event processing. The second one is known to be polynomial in time, therefore the final complexity is bounded by the complexity of the ontological language used to describe the events.

In our future work, we will focus on the full language specification, i.e. full complexity description and the analysis under different DL fragments. We will investigate how to add the underlying definition of the events defined as RDF

[9] The code is part of the new version of MASSIF platform which is not yet available as open source. A stand alone version will be published at `https://github.com/IBCNServices/OBEP`

[10] https://jena.apache.org/documentation/query/

[11] http://www.espertech.com/

graphs. Integrating this in the language facilitates the creation of a more complete system that allows the processing of data on different levels. We aim at introducing explicit complex event construction semantics and also important time-aware operators, such as the ones of Allen's algebra. Finally, we plan to combine our approach with static knowledge for advanced inference and to thoroughly compare the performance of a prototype with state-of-the-art solutions, such as EP-SPARQL.

	R1.a	R1.b	R2	R3	R4 (Filters)	R4 (Joins)
EPL	Relational	/	$\star$	✓	✓	✓
EP-SPARQL [1]	RDF BGP	RDFS	seq, opt_seq eq_opt_seq	/	✓$^{\otimes}$	✓$^{\otimes}$
Taylor et al [9]	OWL	Boh[12]	seq, or, and	✓	/	/
MASSIF [4]	DL Axioms	OWL 2 DL	/	/	✓	/
OBEP	DL Axioms	OWL 2 DL	$\star$	✓	✓$^{\otimes}$	✓$^{\otimes}$

Table 1: Differences and similarities between (S)CEP and OBEP approaches against Section 4 requirements. $\otimes$, i.e. SPARQL-like; $\star$, i.e. seq, and, or, not, every, within.

References

1. Anicic, D., et al.: EP-SPARQL: a unified language for event processing and stream reasoning. pp. 635–644 (2011)
2. Barnaghi, P., et al.: Semantics for the Internet of Things: Early Progress and Back to the Future. Int. J. Semant. Web Inf. Syst. 8, 1–21 (Jan 2012)
3. Böhlen, M.H., et al.: Point-versus interval-based temporal data models. In: Proceedings of the Fourteenth International Conference on Data Engineering, 1998. pp. 192–200 (1998)
4. Bonte, P., et al.: The massif platform: a modular and semantic platform for the development of flexible iot services. KAIS pp. 1–38 (2016)
5. Della Valle, E., et al.: It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems 24(6), 83–89 (2009)
6. Lutz, C., et al.: Temporal description logics: A survey. In: 15th International Symposium on Temporal Representation and Reasoning, TIME 2008, Université du Québec à Montréal, Canada, 16-18 June 2008. pp. 3–14 (2008)
7. Shearer, R., et al.: Hermit: A highly-efficient owl reasoner. In: OWLED. vol. 432, p. 91 (2008)
8. Sirin, E., et al.: Terp: Syntax for owl-friendly SPARQL queries. In: Proceedings of the 7th International Workshop on OWL: Experiences and Directions (OWLED 2010), San Francisco, California, USA, June 21-22, 2010 (2010)
9. Taylor, K., et al.: Ontology-driven complex event processing in heterogeneous sensor networks. In: The Semanic Web: Research and Applications - ESWC 2011, Proceedings, Part II. pp. 285–299 (2011)