Event Processing in RDF

Mikko Rinne[1], Eva Blomqvist[2], Robin Keskisärkkä[2], and Esko Nuutila[1]

[1] Department of Computer Science and Engineering,
Aalto University, School of Science, Finland
firstname.lastname@aalto.fi
[2] Linköping University, 581 83 Linköping, Sweden
firstname.lastname@liu.se

Abstract. In this study we look at new requirements for event models based on concepts defined for complex event processing. A corresponding model for representing heterogeneous event objects in RDF is defined, building on pre-existing work and focusing on structural aspects, which have not been addressed before, such as composite event objects encapsulating other event objects. SPARQL querying of event objects is also considered, to demonstrate how event objects based on the model can be recognized and processed in a straightforward way with SPARQL 1.1 Query-compliant tools.

Keywords: Complex event processing, ontologies, SPARQL, RDF

1 Introduction

Event models (e.g. [17, 19]) currently available for tools using Semantic Web technologies do not address all necessary aspects of event processing, for example, composite event objects where higher-level event objects encapsulate lower-level event objects. Work on stream processing using Semantic Web technologies has initially focused on processing streams of individual triples [3, 11, 10] rather than events, but the use of larger subgraphs with more heterogeneous structures using RDF[3] and SPARQL[4] has also been described [15]. We extend current event models to incorporate more aspects of (complex) event processing and demonstrate how streams of structured and heterogeneous event objects, represented based on our model, can be processed using SPARQL 1.1.

Complex event processing, as pioneered by Luckham, Etzion and Niblett [13, 6], is based on layered abstractions of events. An event is defined by [14] as "anything that happens, or is contemplated as happening". A complex event is "an event that summarizes, represents, or denotes a set of other events". Real-world events are observed by sensors, which translate them to simple *event objects*, i.e., records of the observations in the system environment, which constitute the representation of an event that the system processes. Interconnected rule processors, denoted "event processing agents" (EPA) [14], transform patterns of

[3] http://www.w3.org/RDF/
[4] http://www.w3.org/TR/sparql11-query/

simple event objects, potentially from very heterogeneous sources, to complex event objects of higher abstraction levels. As an example, we may have sensors measuring the water level and flow in different parts of a network of rivers and lakes. That information combined with a weather forecast for heavy rain could be used to derive a flood warning, which in this case would be an abstract complex event object. So far, none of the existing event ontologies address any of the challenges of treating complex and composite event objects.

The solution has been modelled in the form of a Content Ontology Design Pattern [7] (hereafter simply denoted ODP), which is a reusable ontology component that can be used independently of the event models it is built upon, but which is also aligned to several important models. The proposed ODP is available in the ODP Portal[5]. By defining a comprehensive and structured model for representing event objects, this work addresses challenges on the level of abstraction as well as integration [5], and constitutes a novel and necessary step for further research in event processing based on Semantic Web technologies. The proposed model can be used as a tool for event processing systems to structure, integrate and manage such streams (whatever their original vocabularies), and for interchange of event objects. To use the model, it is not necessary that an incoming stream is already structured according to the ODP, refactoring of the streamed data can also be a task performed by the event processing system itself.

The paper starts by an in-depth discussion of the state of the art, existing solutions and tools, which are reviewed in Section 2. Section 3 reviews the requirements for an event model for event processing. Our solution is described in Section 4, with a discussion on benefits and shortcomings, as well as future work, in Section 5. Conclusions are presented in Section 6.

2 Background and Related Work

When something happens in the material world, it may be detected by mechanical, electric, or human sensors. These sensors emit streams of observations. A particular pattern of observations, detected by an event processing system, triggers the creation of an *event object* mirroring and/or describing the real-world event, as illustrated in Figure 1. Traditionally, observation streams have been handled by data stream management systems, with their roots in databases, where the processing unit is a row of a table [2]. Parameters, such as time, are used to portion infinite streams into windows, which are processed by aggregate operators to derive numerical conclusions descriptive of the contents. This heritage has later been applied to the Semantic Web by constructing streams out of time-annotated RDF triples [8, 9, 3] and extending SPARQL with window operators [3, 11], which isolate portions of the streams based on the timestamps.

Processing based on individual triples is, however, very limited. Most data sources, including sensors, can attach various measurements and other attributes than time (e.g., location) to the units of data they provide [12]. Moreover, an

[5] http://ontologydesignpatterns.org/wiki/Submissions:EventProcessing

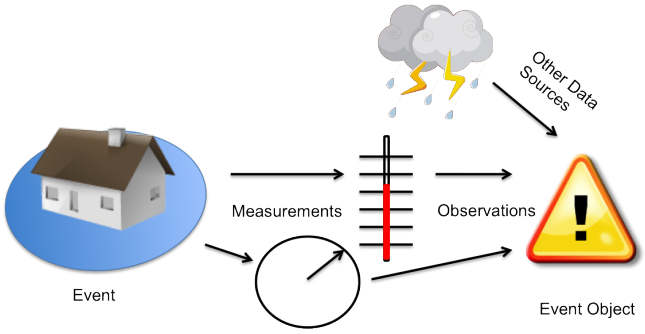

Fig. 1: An event object compiled from sensor observations and other data sources.

event object can be associated with multiple time-related parameters, e.g., time of sampling based on the clock of the sensor, time of entry to the data stream, time of arrival in the event processing system and time of event object validity [8], which need to be understood by the system to produce the desired result. Computing aggregate values such as minimum, maximum, sum and average from a single parameter is a practical way to summarize and clean noisy data, but it is not yet a means to derive layered conclusions for complex event processing.

The SPARQL query language has the capability to match and isolate subgraphs of data, with significant new functionality added in v. 1.1, e.g., property path handling. Processing heterogeneous event objects consisting of multiple RDF triples with a common timestamp has been demonstrated in [15, 16]. In this paper we review those aspects of (complex) event modelling, which in our view have not been fully addressed in the event (and semantic sensor) ontologies currently available [17, 19, 12].

The final report of the W3C Semantic Sensor Network XG [12] reviews numerous existing event and sensor ontologies, and subsequently describes a comprehensive ontology[6] (hereafter denoted the SSN ontology) for conveying the output of sensors. The concepts "Observation" and "SensorOutput" in the SSN ontology are, however, restricted to describing the output of exactly one sensor, so they do not extend to the concept of event objects nor abstractions into complex events. Our model extends the SSN ontology with such concepts, and follows a common baseline by using the DOLCE Ultra Light[7] (hereafter denoted DUL) top-level ontology as a formal basis, to be compatible with the SSN ontology.

When extending the SSN ontology with the concept of (complex) event objects, we have reviewed and considered to reuse existing event ontologies. The Event Ontology[8], rooted in describing music events, and the LODE ontology[9],

[6] http://www.w3.org/2005/Incubator/ssn/ssnx/ssn
[7] http://www.loa-cnr.it/ontologies/DUL.owl
[8] http://motools.sf.net/event
[9] Linking Open Descriptions of Events: http://linkedevents.org/ontology/

are both general enough to be applicable also for event processing, but lack some of the structures needed, for instance, the notion of complex events as abstractions over simple events. Taylor and Leidinger define an ontology [19] for complex event processing[10], but it is highly specific to the problem domain, containing references to particular observations, such as wind speed, which makes it unsuitable as a general pattern. The Event-F ontology[11] [17] on the other hand is a comprehensive framework, also derived from DUL, which is general enough to serve our purpose, but which still lacks the specifics of complex events. However, since Event-F is directly compatible with the SSN ontology, through DUL, it provides a good foundation for our extension, hence, we align our concepts also to Event-F. Another alignment between the two ontologies was made in the SPITFIRE project, producing an extended ontology[12] for describing sensor contexts as well as energy requirements, but this extension still lacks classes and properties for describing complex events. Also note that both SSN and Event-F are large ontologies (as is the SPITFIRE extension), and being based on DUL quite heavily axiomatized. In contrast to this, our model is published as an ODP, without importing either ontology, but rather simply aligning to them.

3 Requirements of an Event Model for Complex Event Processing

When developing the Event-F ontology, the WeKnowIt project collected a comprehensive set of requirements of general event models [18]. Such generic requirements include: participation of objects in events, temporal duration of events, spatial extension of objects, relationships between events (mereological, causal, and correlations), as well as documentation and interpretation of events. Additionally, a number of non-functional requirements, such as extensibility, formal precision (axiomatization), modularity, reusability, and separation of concerns. Even though the requirements are covered by Event-F, it does not cover all the needs of modelling complex events, hence, we here add the requirements that have not yet been addressed. The non-functional requirements have influenced the design of the model we are proposing, while taking some of the requirements even further, such as providing an ODP rather than a large core ontology of complex events, which takes the modularity requirement even one step beyond the design of the Event-F ontology.

Requirements not directly addressed by Event-F (nor any other current event model, or the SSN ontology):

1. *Events and event objects:* In the commonly agreed terminology of [14] there is a clear separation between *events*, as something occurring in the real world, and *event objects*, which are representations of the real-world events as described within some computer system that may be used to detect or process

[10] http://research.ict.csiro.au/conferences/ssn/EventOntology_no_imports.owl
[11] http://west.uni-koblenz.de/Research/ontologies/events
[12] http://spitfire-project.eu/ontology/ns/

the real-world events in some way. Although one could argue that an event model will never contain the real-world events themselves, i.e., whatever is modeled by an ontology will always be a representation of an event, we find it important to allow this distinction in a model for complex events because the breakdown of events into their parts and related events that a human user finds reasonable may considerably differ from the *event objects* that are actually present in the system for detecting or describing the event. Hence two parallel modelling structures should be used for this purpose. For instance, consider a music festival night as an event that occurs in the real world. Intuitively we may, for instance, describe this event as a set of concerts by different artists that are held in sequence on the same stage. However, a system representation of this event, i.e., the event objects, may very well display a completely different content and structural breakdown. For instance, we may use sound level sensors to detect that there is some activity on the stage, and make readings every minute, then the music festival night is actually represented by a "loud period" event object in the system, which consists of sub-events that are the individual sound level readings, together with some mechanism detecting that what is heard is actually music.

2. *Payload support:* Following [6], an event object is split into an event object header, which contains necessary information for processing the event object, and optional payload, which may not be fully understood or processed by the event processing network but needs to be kept associated with the event object. We incorporate optional "header" and "body" segments for event objects to demonstrate the capability of handling unknown components, e.g., unknown vocabularies used for the body of the event object.
3. *Encapsulated event objects:* The structurally most demanding type of event object in [14] is a "composite event object", which contains the event objects it is composed of, in a separable form. As the event objects constituting a composite event object may themselves be composite event objects, the recursion of composite event objects within composite event objects should be supported in any number of layers. It is worth noting that such a structural relation between *actual events* is already present in Event-F through the mereological relations borrowed from DUL. However, as we have already noted in the first requirement we need to clearly separate events from event objects, hence, when aligning event objects to the SSN ontology, i.e., modelling them as information objects in a system, we will need an additional such structure for the event objects (in addition to the one in Event-F).
4. *References to triggering events:* Complex event objects resemble composite event objects, since both are referencing other event objects, but while the parts of a composite event object are wholly dependent on the encapsulating event object (through partonomy) a complex event object can also simply be an event object that somehow is related to other event objects (referencing other event objects), e.g., by being an abstraction of a set of low-level event objects. The relation between complex event objects and their related event objects is therefore a kind of constituency (in the DUL terminology) rather than partonomy. An important use of such a relation is the ability to point

to the triggering event objects, so that it is possible to trace what triggered the abstraction, and hence the appearance of this complex event object.

5. *Multiple time-stamps:* An event object can be associated with multiple time-related parameters, e.g., time of sampling based on the clock of the sensor, time of entry to the data stream, time of arrival to the stream processing system and time of event object validity [8], which need to be understood by the system to arrive at the desired outcome. An event model needs to be able to distinguish between different kinds of timestamps.
6. *Querying ability:* An aspect that has been overlooked in, for instance, Event-F is the usage of the model for supporting queries over event objects. Although Event-F is logically sound and well-designed, it is not modelled particularly with querying in mind. Several structures in Event-F involve n-ary relations in several layers, modelled as OWL classes, which contributes to very long and complex query expressions. Although this may be an acceptable price to pay for increased reasoning capabilities, we also raise the importance of being able to easily query the represented event objects, and being able to formulate generic "query templates" for managing event objects.

In addition to these specific requirements, we have also tried to adhere to the general characteristics of Content ODPs, as described in [7], which can be seen as a list of desired features, including the provision of a reusable computational component representing the ODP, making the ODP small and autonomous, enabling inferencing on the ODP, and making it cognitively and linguistically relevant as well as a representation of best practices (including to adhere to a commonly agreed terminology, such as [14]).

4 Proposed Solution

4.1 Event Model in OWL

As a starting point for our proposed ODP model, we have taken the SSN ontology [12] and the Event-F ontology [18]. Event-F can be used together with our proposed ODP to connect to more detailed or user-friendly descriptions of the event itself, e.g., for reasoning purposes, when the rich axiomatization of events from Event-F is desirable, or for describing the event in a more user-oriented fashion. Both the SSN ontology and Event-F are based on DUL, and by extending and aligning to both these models, our ODP also relies on the DUL ontology. From our point of view, and in accordance with the terminology of [14], the *event object* (Req. 1) is a central concept (see Figure 2), which is the system representation, or record, of an event (real or system generated). An event object can then be related to a "real" event, i.e., a `dul:Event`, which through the alignment to the Event-F model, then has to be a documented event, i.e., a `dul:Event` involved in some `eventf:EventDocumentationSituation`.

An event object can then be either a *simple event object* or a *complex event object*, depending on if it abstracts (summarizes or represents) other, more low-level, event objects or not. A complex event object is something that has some

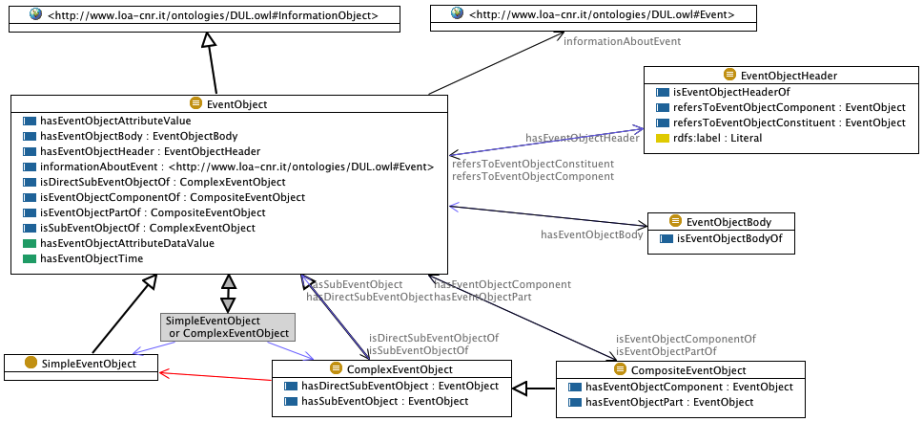

Fig. 2: The core classes of the Event Processing ODP and their relations to DUL concepts (using the UML-like notation of TopBraid Composer, where some details have been omitted due to readability reasons).

"sub-event objects" (Req. 4). A special case of a complex event object is a *composite event object* (Req. 3), which is a complex event object that is actually made up of a set of other event objects, i.e., acting as its parts. As noted by [14] a composite event object is always a complex event object, but every complex event object is not necessarily a composite event object, if it only represents or references other related event objects but does not include them as components. Encapsulated and referenced event objects can be modelled using two separate sets of properties, so that each type of relation can be treated independently. The structure does not require OWL reasoning per se, but gives the opportunity to reason over the structures, using transitivity and inverse properties.

Payload support (Req. 2) is provided through introducing classes for the header and body of an event object, making it possible to distinguish between the known parts of the information and the body, i.e., the payload that may not use any known vocabulary. Nevertheless, we feel that this legacy from earlier event processing systems may not be ideal to include in all RDF stream processing systems, whereby we have modeled the pattern in such a way that it is an optional feature. Event objects can be modelled directly, without header and body parts distinguished, which is more in line with the modelling "freedom" and simplicity of Linked Data and RDF graph data in general.

Multiple timestamps (Req. 5) are supported through a set of separate datatype properties: `hasEventObjectSamplingTime`, `hasEventObjectApplicationTime`, `hasEventObjectSystemTime`, and `hasEventObjectExpirationTime`, corresponding to the time points when the event object was sampled (e.g., recorded by a sensor), entered the data stream, arrived in the event processing system via the stream, and any known end time for the event objects validity, respectively. Although this does not solve the problem of different time references in general, at

least one can now explicitly say what time is actually recorded, and if needed, record several timestamps for each object. An additional desirable feature would be the ability to express which of the timestamps should be the default for time window operations, however, we feel that this lies outside the scope of our current pattern. Rather such capabilities should lie in a vocabulary for describing RDF streams, or event processing systems, not event objects themselves.

Finally, event objects can be effectively processed through SPARQL queries (Req. 6). In particular, we have made sure that some of the most common queries can be expressed in a generic manner, i.e., as "query patterns" (discussed in the following section), to facilitate reuse and to make event processing as uniform as possible between systems. More details on the modelling decisions and the detailed structure of the ODP can be found in the annotations of the ODP model itself, and in the pattern abstract that accompanies it [4].

4.2 Processing Events with SPARQL

Having an ODP for describing the event objects handled by an event processing system is a (practical) contribution in itself, since such a model has not existed before. However, for this contribution to be significant in the future it needs to be practically usable and beneficial to a large class of systems. As also stated in our list of requirements an important aspect of the work is to be able to effectively query the event data structured according to the model, which will make it useful in a system setting. Ideally, the ODP presented above would come with its set of generic query patterns that represent common operations on event objects, which can be reused within any domain. In this section, we show a step towards such generic query patterns, although we also point at some limitations with the current model and SPARQL standard that restrain us from providing a completely generalized solution.

The following example set of four event objects is used to demonstrate operations on composite complex event objects containing a header and a body and having a capability to reference other event objects without encapsulating them:

```
:floodWarning0001 a ep:EventObject ;
  ep:hasEventObjectHeader [
    rdfs:label "Flood Warning composite event" ;
    ep:hasEventObjectTime "2013-07-03T08:18:21"^^xsd:dateTime ;
    ep:refersToEventObjectConstituent :weather0001 ;
    ep:refersToEventObjectComponent :waterAlert0001 ;
    floodex:forecast floodex:ImminentDanger ;
  ] ; #end of Header
  ep:hasEventObjectBody [
    rdfs:comment "Exemplifies a composite event." ;
  ] . #end of Body

:waterAlert0001 a ep:EventObject ;
  ep:hasEventObjectHeader [
     rdfs:label "Water-related alert composite" ;
     ep:hasEventObjectTime "2013-07-03T08:17:21"^^xsd:dateTime ;
     ep:refersToEventObjectComponent:waterLevel2341 ;
     floodex:waterLevelChangeRate floodex:high ;
  ] ; #end of Header
  ep:hasEventObjectBody [
    rdfs:comment "Information external to our system." ;
```

```
    foaf:mbox <mailto:contactrelevanttoanothersystem@example.org> ;
  ] .

:waterLevel2341 a ep:EventObject ;
  ep:hasEventObjectHeader [
     ssn:isProducedBy  [
       a ssn:SensingDevice ;
       rdfs:label "Water Level Measurement" ;
     ] ;
     ep:hasEventObjectTime "2013-07-03T08:17:15"^^xsd:dateTime ;
     ssn:hasValue [
       dul:hasRegionDataValue 22 ;
     ] ;
  ] .

:weather0001 a ep:EventObject ;
  ep:hasEventObjectHeader [
     rdfs:label "Weather forecast for London" ;
     ep:hasEventObjectTime "2013-07-03T08:17:21"^^xsd:dateTime ;
     ep:hasEventObjectAttributeValue floodex:rain ;
  ] ; #end of Header
  ep:hasEventObjectBody [
     ...
  ] . #end of Body
```

The full example is available with the pattern and prefixes in the ODP portal. The `floodWarning` is a composite event object, encapsulating a `waterAlert`. The `waterAlert` is also composite, encapsulating a `waterLevel` measurement. The `floodWarning` also refers to a `weather` event object, but the `weather` event object is not encapsulated in the `floodWarning`. The challenge for querying is to match the `floodWarning` composite event object (for any move, copy, delete or other operation on the composite event object), with all levels of encapsulated event objects, excluding referenced information not integral to the event object.

After the addition of property paths in version 1.1, SPARQL has some new methods for supporting nested structures. Using the property path expression `(ep:hasEventObjectHeader / ep:refersToEventObjectComponent)*`, an arbitrary number of nested composite event objects can be supported. The referenced event object `weather0001` is not matched, as it is referred with `ep:refersToEventObjectConstituent`.

Matching more levels of depth for the header and body in a generic way is not as straightforward. Matching an arbitrary chain of unknown links would be a very powerful tool, which in the world of linked data could eventually end up matching that entire universe - not just the header and body triples we want. Using a property path of unspecified length with a known combination of predicates (`ep:hasEventObjectHeader` and `ep:refersToEventObjectComponent`) can be asserted safe in our controlled setting of an event object stream, where the pair of predicates can be guaranteed only to refer to direct sub-event objects in the same graph. Supporting an arbitrary number of levels of unknown predicates is, however, much more challenging and potentially dangerous. SPARQL doesn't currently offer means to follow such a property path. The same functionality would theoretically be available through negation, using a property path expression such as `(! :foobar)*`, where ":foobar" is a fabricated predicate, which should not appear in the data stream. In addition to the dangers explained above, tool support for this approach is uncertain.

In case of the header the support for more depth can always be achieved by making the SPARQL query more explicit, because the structure of the header is assumed to be known by our event processing application, but the structure of the body is assumed to be unknown. One way to support deeper structures is to restrict such structures to be linked only through blank nodes, since blank nodes cannot point to nodes outside the current graph. Linking through blank nodes allows more depth in the event object structure without setting an explicit requirement to know the contents.

Using these tools we can write a query, which correctly constructs a copy of the `floodWarning0001` composite event object including the encapsulated event objects `waterAlert0001` and `waterLevel2341` without prior knowledge of their existence or contents:

```
CONSTRUCT { # Create a copy of the matched event object, with encapsulated event objects
   ?event a ep:EventObject ;
          ep:hasEventObjectHeader ?header .
   ?header ?hp ?hv .                          # First level headers
   ?header2 ?hp2 ?hv2 .                       # Second level nested headers
   ?event ep:hasEventObjectBody ?body .
   ?body ?bp ?bv .                            # First level body
} WHERE { # Match an event object with all nested levels of encapsulated event objects
   :floodWarning0001 ( ep:hasEventObjectHeader /
      ep:refersToEventObjectComponent )* ?event .
   ?event a ep:EventObject .
   ?event ep:hasEventObjectHeader ?header .   # Mandatory header
   OPTIONAL { ?header ?hp ?hv                 # Optional first-level headers
     #Optional second-level nested headers, only through blank nodes
     OPTIONAL { BIND ( IF (isBlank(?hv), ?hv, 0) as ?header2)
              ?header2 ?hp2 ?hv2 } }
   OPTIONAL {
     ?event ep:hasEventObjectBody ?body .     # Optional body
     OPTIONAL { ?body ?bp ?bv } }             # Optional first-level body content
}
```

To keep the query compact the amount of nested levels in the header and the body has been set to match our example. More nested levels can be added by adding more nested OPTIONAL-clauses, checking that linking is taking place specifically through blank nodes. To the best of our knowledge matching an unspecified number of nested levels through blank nodes only is not possible with SPARQL 1.1 without explicit knowledge of the predicates, in which case the query would again need to be explicitly defined for each nesting level. Apart from the explicit subject (`:floodWarning0001`) the example query is generic and would work with a structurally compliant event object independent of the content. In a real-world application the explicit subject could be replaced by some other criteria to match the desired event object in an event object stream.

5 Discussion and Future Work

Our proposed model has been published as an ODP, which comes with several advantages. For instance, both the SSN and Event-F ontologies may be perceived as quite large and "heavy" to understand and use, while our small model only contains a handful of classes and properties that can be grasped quite easily. It can also be used independently of the DUL axiomatization, if this is not desirable

or compatible for a specific use case. On the other hand, the lack of upper level axiomatization can be easily amended through our careful alignments (included in the model as axioms), by simply adding the missing imports (SSN and Event-F), if the upper level is needed for a particular use case.

A generic way of matching event objects using SPARQL 1.1 was demonstrated, supporting:

- the distinction between events and *event objects* (Req. 1), if desired,
- inclusion of a header and an optional body (Req. 2), with unknown content, in the event object structure,
- composite event objects encapsulating other event objects (Req. 3), potentially over any number of nested layers (limited by the SPARQL implementation),
- referencing of other event objects without encapsulating them (Req. 4), and
- generation of query templates to process compliant event objects (Req. 6), observing related restrictions.

Support for multiple timestamps (Req. 5) is built into the model, but selection of the timestamp to use for a particular purpose was considered to be a stream-specific parameter and outside the scope of this paper.

As a restriction to the generality of queries, knowledge of the maximum number of nested levels of RDF triples supported within the header or body was observed to be required a priori, with every level adding some complexity to the query needed to match an event object. To avoid following links outside event objects, nesting of the body should only be done using blank nodes. Deviations of the format, such as allowing event objects both with and without explicit header, or objects of other classes (such as `ssn:SensorOutput`) add complexity to queries. The recommended approach would be to convert all event objects to a uniform format upon entry to the event processing network. The specific conversions are outside the scope of this document, but due to the flexibility of RDF most formats used for describing events can be converted to RDF representations compliant with the presented model in a straightforward way.

On a more general note, currently available commercial complex event processing tools[13] use different means of defining the event processing network and the agents within, apart from systems based on a common root. The introduction of an event processing model meeting the requirements of complex event processing enables tools based on Semantic Web technologies to address the same application space. Compared to proprietary approaches, RDF and SPARQL have the benefit of a specified definition language, paving the way for improved tool compatibility. Integrated processing of event streams with static, linked and open datasets in the cloud and the built-in reasoning capabilities of Semantic Web tools are also strong benefits.

A longer-term target is to make semantic stream processing systems configurable to understand and process heterogeneous, layered event objects both on

[13] e.g. http://www.thetibcoblog.com/wp-content/uploads/2011/12/cep-market-dec2011.png

live streams as well as recorded data. For recorded data, the systems will need to follow the same stream parameters (e.g. time) to be used for both algebra and relational operators. Descriptions of the operational semantics of flow processing systems should be developed so that it is possible to know a priori, where results of processing the same data using the same set of queries will be different.

To pave the way, harmonization of tools and specifications could be improved, e.g., on the following aspects:

- Common event processing model (vocabulary). This paper makes a contribution, but to be effective on a broad scale, further community consensus on the model is needed.
- Common stream description vocabulary, and publication mechanisms. In addition to describing the event objects inside the stream, and in the event processing system, streams themselves need to be described and published, e.g., similar to other web services, in an agreed upon format.
- Representations and handling of time. Understanding of a common time reference between systems using a recorded stream should be possible with a reasonable amount of configuration rather than requiring reprogramming of system components.
- Harmonized descriptions of the operational semantics of semantic flow processing systems [1]
- Benchmarking of semantic flow processing systems. There should be tests both for data stream management as well as layered event processing. Correct results, in light of operational semantics, should be defined so that performance and correctness of operation can be compared.

6 Conclusions

In this paper we propose a novel Event Processing ODP, i.e., a vocabulary for representing and reasoning over complex and composite event objects, which is needed for further progress in the area of RDF stream processing. In addition to the model itself, another contribution is the demonstration of generic query patterns for event object management using SPARQL 1.1, which facilitates event processing. The model is aligned to important standards, such as the SSN ontology, and compatible with other event models, such as Event-F, and it also meets all the requirements for representing and processing event objects that were discussed in Section 3.

Acknowledgments

This work was supported by European Commission through the SSRA (Smart Space Research and Applications) activity of EIT ICT Labs[14], and CENIIT at Linköping University through the grant 12.10.

[14] http://eit.ictlabs.eu/ict-labs/thematic-action-lines/smart-spaces/

References

1. Aglio, D.D., Balduini, M., Valle, E.D.: On the need to include functional testing in RDF stream engine benchmarks. In: 1st International Workshop On Benchmarking RDF Systems (BeRSys 2013) Co-loc. with ESWC 2013. Montpellier, FR (2013)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proc. of the 21st ACM SIGMOD-SIGACT-SIGART symp. on Principles of database systems - PODS '02. ACM Press, New York, USA (2002)
3. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for C-SPARQL queries. In: Proceedings of the 13th International Conference on Extending Database Technology. p. 441. Lausanne, Switzerland (2010)
4. Blomqvist, E., Rinne, M.: The Event Processing ODP. In: Proceedings of WOP2013 - Pattern track. CEUR Workshop Proceedings, CEUR-WS.org (2013)
5. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
6. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning Publications (Jul 2010)
7. Gangemi, A., Presutti, V.: Ontology Design Patterns. In: Handbook on Ontologies, 2nd Ed. International Handbooks on Information Systems, Springer (2009)
8. Gutierrez, C., Hurtado, C., Vaisman, R.: Temporal RDF. In European Conference on The Semantic Web (ECSW 2005) pp. 93—107 (2005)
9. Gutierrez, C., Hurtado, C.A., Vaisman, A.: Introducing time into RDF. IEEE Transactions on Knowledge and Data Engineering 19(2), 207–218 (Feb 2007)
10. Komazec, S., Cerri, D., Fensel, D.: Sparkwave : Continuous Schema-Enhanced Pattern Matching over RDF Data Streams. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 58–68. ACM (2012)
11. Le-Phuoc, D., Dao-Tran, M., Parreira, J.X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC'11. pp. 370–388. Springer-Verlag Berlin (Oct 2011)
12. Lefort, L., Henson, C., Taylor, K.: Semantic Sensor Network XG Final Report (2011), http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/
13. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, 1 edn. (2002)
14. Luckham, D., Schulte, R.: Event Processing Glossary Version 2.0 (2011), http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/
15. Rinne, M., Abdullah, H., Törmä, S., Nuutila, E.: Processing Heterogeneous RDF Events with Standing SPARQL Update Rules. In: Meersman, R., Dillon, T. (eds.) OTM 2012 Conferences, Part II. pp. 793–802. Springer-Verlag (2012)
16. Rinne, M., Törmä, S., Nuutila, E.: SPARQL-Based Applications for RDF-Encoded Sensor Data. In: 5th International Workshop on Semantic Sensor Networks (2012)
17. Scherp, A., Franz, T., Saathoff, C., Staab, S.: F – A Model of Events based on the Foundational Ontology DOLCE + DnS Ultralite. In: International Conference on Knowledge Capturing (K-CAP). Redondo Beach, CA, USA (2009)
18. Scherp, A., Papadopoulos, S., Kritikos, A., Schwagereit, F., Saathoff, C., Franz, T., Schmeiss, D., Staab, S., Schenk, S., Bonifacio, M.: D5.2.1 Prototypical Knowledge Management Methodology (2009), http://www.weknowit.eu/sites/default/files/D5.2.1.pdf
19. Taylor, K., Leidinger, L.: Ontology-Driven Complex Event Processing in Heterogeneous Sensor Networks. In: Proc. of the 8th Extended Semantic Web Conference (ESWC2011). pp. 285–299. Springer Berlin Heidelberg (2011)