The Event Processing ODP

Eva Blomqvist[1] and Mikko Rinne[2]

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

Abstract. In this abstract we present a model for representing heterogeneous event objects in RDF, 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. The model extends the SSN and Event-F ontologies, and is available for download in the ODP portal.

1 Introduction

Layered processing of heterogeneous events [2] with Semantic Web tools [4] sets new requirements to the event models to be used. For example, higher-level composite event objects may encapsulate the lower-level event objects, which triggered the composite abstraction. By defining a comprehensive and structured model for representing event objects, this work addresses current challenges, as discussed in [1], and constitutes a novel contribution in event processing based on Semantic Web technologies. The proposed Content ODP extends current event models to incorporate aspects of (complex) event processing, and additionally facilitates the use of generic SPARQL patterns for event object data management. For a detailed discussion of the requirements underlying the proposed ODP and related work that has been considered, see [5]. Our proposed Event Processing ODP is available in the ODP Portal[3]. The ODP is very general, and therefore applicable to any domain where event processing is to be performed. As with any Content ODP, if needed it can be specialized to include more domain-specific classes and properties suiting that domain, by importing the ODP and adding subclasses, subproperties, and additional domain-dependent axioms in the importing ontology (or ODP).

2 Structure of the Event Processing ODP

An important standardisation effort has been the Semantic Sensor Network (SSN) ontology[4], by the W3C Semantic Sensor Network Incubator group. It is

[3]http://ontologydesignpatterns.org/wiki/Submissions:EventProcessing
[4]http://purl.oclc.org/NET/ssnx/ssn

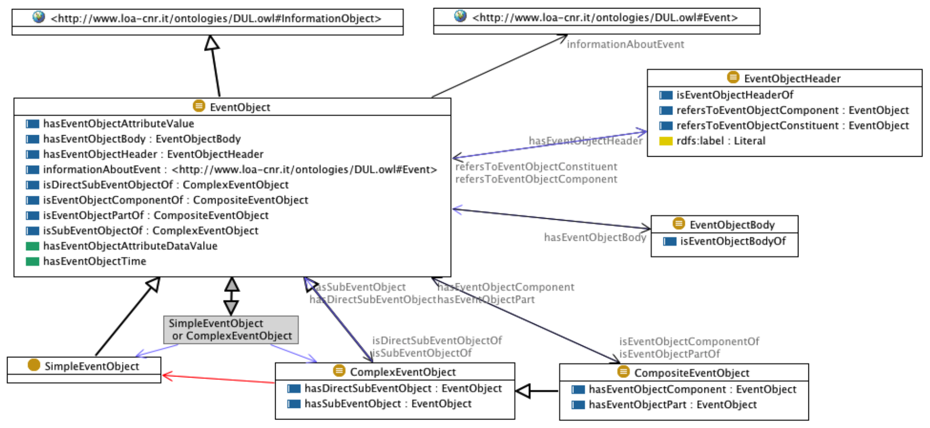
important that an event processing ODP is fully aligned with the SSN ontology, merely extending it with new concepts. Another important effort is the Event-F ontology[5], describing different aspects of events in the real world. However, what we describe are records of events (information about events in a system) rather than the event itself, c.f. `SensorOutput` of the SSN ontology. Nevertheless, Event-F can be used together with our ODP to connect to descriptions of the event itself, e.g., for exploiting the axiomatization of Event-F. Both the SSN ontology and Event-F are based on DOLCE Ultra Light [6] (DUL). By extending, and aligning to, both of these models, our ODP is also aligned to DUL. Another alignment between the two ontologies has already been made in the SPITFIRE project, producing an extended ontology[7] for describing sensor contexts as well as energy and network requirements, but this extension still lacks the necessary classes and properties for describing complex events, which we add through this ODP.

In the terminology of [3], the *event object* is the system representation, or record, of an event (real or system generated). An event object can 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. Additionally, a special case of a complex event object is a *composite event object*, which is a complex event object that is actually made up of a set of other event objects, i.e., acting as its parts. A composite event object is always a complex event object, but every complex event object is not necessarily a composite event object. A complex event object can be structurally equivalent with a simple event object, having only the semantic difference of representing other event objects in the network.

In the proposed ODP (core classes and properties illustrated in Fig. 1) we have modelled the `EventObject` class as a subclass of the `dul:InformationObject` (similarly as the `ssn:SensorOutput`). We have aligned our `EventObject` class to the SSN ontology, by expressing that `ssn:SensorOutput` is equivalent to `SensorOutput` in the local namespace, which in turn is a subclass of `EventObject`. The reason for introducing a new `SensorOutput` class is to make the ODP more self-contained, but still make the alignment explicit (even without importing the SSN ontology). A `SensorOutput` is normally a `SimpleEventObject`, however, when considering more complex sensors, such as human sensors entering information into a system, they can be directly entered into the system as complex event objects. `SimpleEventObject` may also consist of other kinds of event objects than a `SensorOutput`, e.g., simulated event objects produced inside a system.

To relate event objects to each other, we introduce a set of object properties (`hasSubEventObject` and `hasDirectSubEventObject`, and their inverses). The object properties are modelled in a way that allows us to keep both a hierarchical sub-event object structure through a non-transitive property (`hasDirectSubEvent-`

[5]http://west.uni-koblenz.de/koblenz/fb4/AGStaab/Research/ontologies/events/model.owl
[6]http://www.loa-cnr.it/ontologies/DUL.owl
[7]http://spitfire-project.eu/ontology/ns/



<http://www.loa-cnr.it/ontologies/DUL.owl#InformationObject>
<http://www.loa-cnr.it/ontologies/DUL.owl#Event>
informationAboutEvent
EventObjectHeader
isEventObjectHeaderOf
refersToEventObjectComponent : EventObject
refersToEventObjectConstituent : EventObject
rdfs:label : Literal
EventObject
hasEventObjectAttributeValue
hasEventObjectBody : EventObjectBody
hasEventObjectHeader : EventObjectHeader
informationAboutEvent : <http://www.loa-cnr.it/ontologies/DUL.owl#Event>
isDirectSubEventObjectOf : ComplexEventObject
isEventObjectComponentOf : CompositeEventObject
isEventObjectPartOf : CompositeEventObject
isSubEventObjectOf : ComplexEventObject
hasEventObjectAttributeDataValue
hasEventObjectTime
hasEventObjectHeader
refersToEventObjectConstituent
refersToEventObjectComponent
EventObjectBody
isEventObjectBodyOf
hasEventObjectBody
hasSubEventObject
hasDirectSubEventObject
hasEventObjectComponent
hasEventObjectPart
SimpleEventObject or ComplexEventObject
isDirectSubEventObjectOf
isSubEventObjectOf
isEventObjectComponentOf
isEventObjectPartOf
SimpleEventObject
ComplexEventObject
hasDirectSubEventObject : EventObject
hasSubEventObject : EventObject
CompositeEventObject
hasEventObjectComponent : EventObject
hasEventObjectPart : EventObject

Fig. 1: The core classes of the Event Processing ODP.

`Object`) but (using a reasoner) also allows for directly retrieving all sub-event objects through its transitive superproperty (`hasSubEventObject`). The properties are aligned to DUL, i.e., sub-properties of `dul:hasConstituent` (and `dul:isConstituentOf`). These properties allow us to express event abstraction on the side of the event objects, while a similar structure exists on the side of actual events in DUL, i.e., directly for the `dul:Event` class using `dul:hasConstituent` and `dul:isConstituentOf`. These two parallel structures do not necessarily need to mirror each other, which is actually the main motivation for separating them. 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, which could be modelled using Event-F, the `dul:Event` class and its associated properties for mereological breakdown. 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. The latter breakdown would then use the properties of the Event Processing ODP, to express the breakdown of the event objects from the system viewpoint. In addition, we also model event objects that are partitioned into components, i.e., whose parts are other event objects. This is modelled in a similar manner, i.e., again with a property structure parallel to Event-F, but this time exploiting the `dul:hasPart` and `dul:hasComponent` properties.

Since the notion of *event object header* and *event object body* is a prominent part of some systems (for a more detailed discussion on the aspect, see [5]), it is important to include in our model, but at the same time accommodate systems where this is not present. We include the concepts `EventObject-Body` and `EventObjectHeader` in the ODP (as `dul:InformationObject` subclasses), as well as object properties for relating an `EventObject` to its header and body, and further expressing the content of the header and body. For instance, if a header is used, the relation to other event objects (componency or constituency) can be expressed through a property of the header instead (`refersToEventObjectComponent` and `refersToEventObjectConstituent` respectively). By exploiting OWL property chains, this is then equivalent to directly stating the relation for an individual of `EventObject`.

The actual information content of an `EventObject`, whether organized into a header and body or not, can then be expressed through properties, such as `hasEventObjectAttributeValue` (subproperty of `dul:hasRegion`) and `hasEventObjectAttributeDataValue` (subproperty of `dul:hasDataValue`), or through arbitrary properties that the application case requires. To model different timestamps of the `EventObject`, there are a set of subproperties of `hasEventObjectAttributeDataValue` (in turn subproperty of `dul:hasDataValue`), grouped under a property called `hasEventObjectTime`, i.e., the following four OWL 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. The alignment to Event-F is constructed through a restriction on the `EventObject` class, expressing that any `EventObject` describes some "real" event (`dul:Event`), which then according to the Event-F model has to be a documented event, i.e., a `dul:Event` involved in some `eventf:EventDocumentation-Situation`.

3 Example of Usage

As an example usage of the ODP, 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 rain could be used to derive a flood warning, which would be a complex event object. An RDF file containing an instantiation of the ODP, realizing this small example scenario can be found in the ODP portal[8]. In this example, a flood warning is an instance of `EventObject`, which is a composite event object, encapsulating another `EventObject`, which is a water alert. The water alert, in turn is also composite, encapsulating a water level measurement. The flood warning also refers to a weather event object, but the weather event object is merely referenced (rather than encapsulated). If the flood warning did not encapsulate the triggering event objects, it would still be a complex event object but not a composite event object. Each of the `EventObject` instances have some data attached, and most of them use the header/body

[8] http://www.ontologydesignpatterns.org/cp/examples/eventprocessing/eventexample.owl

structure described previously. In addition, the example shows how external vocabularies can easily be used for expressing `EventObject` instances and data. For example, the weather data uses the meteo vocabulary[9], and to illustrate the alignment to the SSN ontology the water level measurement `EventObject` is modelled entirely using classes and properties from the SSN ontology.

4 Conclusions

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. The model is aligned to important standards, such as the SSN ontology, and compatible with other event models, such as Event-F, and it is particularly designed so as to be flexible enough to accommodate different event object structures, yet generic enough to allow for expressing generic queries for management of the event objects. For a complete discussion of benefits and relations to the underlying requirements, see the accompanying full paper [5].

Acknowledgments

This work was partially supported by European Commission through the SSRA (Smart Space Research and Applications) activity of EIT ICT Labs[10], and CENIIT at Linköping University through the grant 12.10.

References

1. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
2. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning Publications (Jul 2010)
3. Luckham, D., Schulte, R.: Event Processing Glossary Version 2.0 (2011), http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/
4. 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)
5. Rinne, M., Blomqvist, E., Keskisärkkä, R., Nuutila, E.: Event Processing in RDF. In: Proceedings of WOP2013 - Research paper track. CEUR Workshop Proceedings, CEUR-WS.org (2013)

[9] http://inamidst.com/sw/ont/meteo
[10] http://eit.ictlabs.eu/ict-labs/thematic-action-lines/smart-spaces/