# Event Object Boundaries in RDF Streams – A Position Paper –

Robin Keskisärkkä and Eva Blomqvist

Department of Computer and Information Science
Linköping University, Sweden
{robin.keskisarkka|eva.blomqvist}@liu.se

**Abstract.** The amount of information available as online streams is increasing steadily. A number of RDF stream processing systems have been developed in an attempt to leverage existing Semantic Web technologies, and to support typical stream operations, but very little attention has been paid to the way in which event objects (i.e. data records representing events) are streamed. In this position paper, we present the issue of respecting event object boundaries in RDF streams, and discuss some pros and cons of the various solutions.

**Keywords:** RDF stream processing, RDF streams, event objects

## 1 Introduction

The amount of information available as online streams is increasing steadily, and a number of domains already rely heavily on streams of data, e.g., market feed processing and electronic trading [9]. The costs of deploying sensor networks has been dropping dramatically in recent years, and they are expected to become common in a variety of tasks, including natural disaster response, surveillance, monitoring of potential terrorist and criminal activity, military planning, and more [5, 11].

The Semantic Web community has attempted to lift streaming content to a semantic level [7], but in the context of streaming data traditional Semantic Web technologies are unsatisfactory. Streaming data is characterized by being received continuously in a possibly infinite stream [3], and in many contexts the streaming data becomes outdated quickly, and must therefore be consumed on the fly. Delays in query answering applies both to feeds generating thousands of messages per second, as well as feeds generating only a few messages every hour. In process control, and automation of industrial facilities, streams with large data volumes need to be processed with minimum latency, and in electronic trading systems even sub-second delays can affect profits [12].

While benchmarks have been developed to evaluate system performance [13, 10], and recommendations for publishing Linked Stream Data data have been produced [4, 11], the existing work on RDF stream processing has only discussed how complex data can be represented in a streams, to allow graph boundaries to be respected, to limited degree.

In this position paper, we present the issue of respecting event object boundaries in RDF streams, and present some possible solutions with respect to existing state of the art, and lists some of the pros and cons of the different solutions. Finally, we discuss the open challenges for future research.

## 2 Background - RDF Stream Processing

The authors in [12] differentiate between three different basic architectures that can be applied to solve high-volume low-latency streaming problems: Database Management Systems, rule engines, and stream processing engines. This paper makes no assumptions about underlying architectures, rather, all systems supporting continuous querying of RDF streams in combination with static RDF data is referred to as *RDF stream processing systems*.

Contrary to queries executed against static RDF data, the constant flow of information in a streaming data context means that no final answer can ever be returned. Queries must instead be executed continuously as new data becomes available. Typically, windows over streams are used to allow outdated triples to be ignored, which also makes it possible to support common aggregate functions in queries.

There are, as of yet, no standards for how RDF data should be represented as streams, but a number of state-of-the-art RDF stream processing systems have assumed that the data is delivered as individual RDF triples [1, 6, 3]. Generally, it is also assumed that the triples in a stream are implicitly ordered by arrival time, or explicitly ordered by timestamps. In [3] an RDF stream is defined as an ordered sequence of tuples, where each pair is made of an RDF triple and a timestamp $\tau$:

$$(\langle subj_i, pred_i, obj_i \rangle, \tau_i)$$

$$(\langle subj_{i+1}, pred_{i+1}, obj_{i+1} \rangle, \tau_{i+1})$$

$$...$$

Since the sequence of pairs is ordered the timestamps are strictly non-decreasing. Triples with the same timestamp, although sequenced in a specific order in the stream, should be regarded as having "occurred" at the same time. This view is similar to that which is often used in Data Stream Management Systems [2]. If timestamps are not supplied in the stream triples are typically associated with the time of arrival.

## 3 Problem

Event objects (i.e. data records denoting events) [8] do not only consist of individual RDF triples, but rather of RDF graphs. When event objects are decomposed into lists of RDF triples, intended to be streamed, it becomes essential that the boundaries of the event objects can be respected within queries, since a query that observes only a partial RDF graph may return false results.

In sensor streams, the issue of event object boundaries can often be handled without any great difficulties. For example, if synchronization issues are ignored, a sensor reporting a fixed number of values, or the intervals between streamed events, is known streams can be viewed through a window that slides over the stream. But when the input is not predictable, a window over a stream, by itself, cannot guarantee that all the triples of an RDG graph are within window boundaries at any given point in time, thus, we could risk querying a partially streamed graph.

The solutions presented in the following section are aimed both towards RDF stream processing systems (i.e. event consumers), and towards the RDF streams (i.e. event producers). For brevity, queries will be discussed only in general terms, rather than as full SPARQL queries.

## 4 Solutions

Depending on the complexity of the RDF graphs that are transferred in an RDF stream, different solutions are possible. As mentioned in the previous section, if the events in a stream have predictable event object boundaries, these can be respected simply by having a window that slides with a fixed amount between query evaluations. However, when event objects are less predictable, this is no a sufficient solution. To support dynamic, and heterogeneous, streams of event objects, a solution must be able to handle streams for which neither the intervals between events, nor the number of triples in an event, is known.

**Header** A common approach to making transferred data interpretable is to provide a message "header", containing meta data about the message. In the case of RDF streams each event object could be preceded by a header, providing information about the RDF graph being streamed. A header would have to be generated by the event producer for every new event object. The event consumer (i.e the RDF stream processing system) would subsequently have to interpret each header, ensuring that registered queries are not executed against a partially transferred RDF graph. The header element could be represented in various ways, e.g., as a triple, a set of triples, or as a dedicated type. A simple example of a header can be seen Figure 1.


**Fig. 1.** The header (square) states the number of triples (circles) in the succeeding RDF graph. The black line represents a point in time, where only parts of the graph currently being streamed is available.

**Commonality** A common attribute, or feature of "commonality", can be used as an indicator for the boundary of an RDF graph. A very basic feature of commonality could be that triples belonging to an RDF graph are streamed in sequence. A trailing *boundary triple* could then be used as a marker to denote that the entire RDF graph has been streamed (see Figure 2).

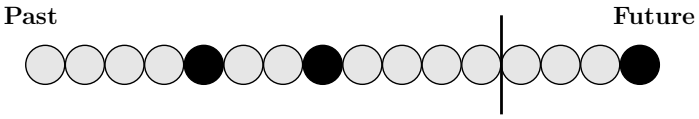


**Fig. 2.** Each black circle denotes the end of the preceding RDF graph. The black line represents a point in time, where only parts of the graph currently being streamed is available.

A more robust solution would be to attach the trailing triple to a node that is unique for the particular graph, i.e., some node that can be used to refer to the specific event object. This could provide support for graph boundaries in situations where graphs can overlap each other in a stream.

**RDF Graph Stream** RDF data streams are usually considered to be composed of individual RDF triples, but we can also consider streaming RDF graphs directly. Streaming sets of RDF triples together, rather than the individual triples separately, would greatly simplify the task of the event producer, since neither the order of decomposition of event object graphs, nor the addition of triples, needs to be considered. Figure 3 illustrates a stream of RDF triple sets, where each set represent a complete event object graph.

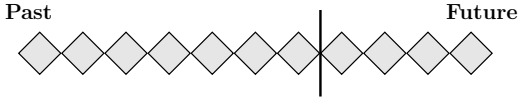


**Fig. 3.** Each diamond represents an object containing a complete RDF graph. The black line represents a point in time.

**Customized Decomposition** Controlling the order, by which the individual triples in an RDF graph are streamed, can sometimes provide a more efficient way of streaming RDF graphs. If an RDF graph can be transferred in an order that implicitly respects event object boundaries correctly, results can be returned even before the a complete RDF graph has been received. This can be quite valuable in contexts where latencies must be kept as low as possible. There is, however, no general way of employing this solution for arbitrary RDF graphs.

## 5 Discussion

Passing a header, as the first element of a decomposed RDF graph, can provide meta data about event objects. For example, before the arrival of an event object the header may convey the size of the arriving RDF graph, allowing the RDF stream processing system to adapt windows over the stream accordingly. A header could, on the other hand, require a different way of consuming data streams, if the header information is to be handled automatically by the RDF stream processing system. One of the benefits of the header solution is that, if the header was handled by the system internally, queries would not need use any special patterns to respect event object boundaries.

Providing a trailing triple, generated for the purpose of separating two RDF graphs, provides a simple way of separating event objects in a stream. The triple could be generated in any number of ways, but the most robust solutions would require it to be connected, in some way, to a unique event object identifier in the preceding graph. Even if we assume that event object triples may overlap each other in the stream, the trailing triple could still be used to determine that the entire graph has been streamed.

The streaming of RDF graphs, rather than individual triples, is a feasible solution for adding support for event object boundaries. Streaming graphs, rather than triples, would require a different way of consuming data streams than in the case of triples. RDF stream processing engines, that rely on evaluation of all queries for every new incoming triple, would either be forced to suppress results based on partial graphs, or would need to support adding multiple triples at a time. The benefit of streaming graphs directly is that, since the event object boundaries never become an issue, engines could provide a built-in support for event object boundaries.

The final solution, suggested in this paper, involves customizing the decomposition of a graph, so that the boundaries are implicitly respected by the order of the streamed triples. In domains where real-time execution is a priority this would enable queries to be executed as soon as a match can be found. However, since this solution is only guaranteed to work in some contexts, and requires a customized solution, it cannot be viewed as a general solution to the issue of event object boundaries.

## 6 Conclusions

While it is rarely discussed in research literature, the way in which RDF graphs are decomposed into streams of RDF triples has a direct impact on the ability to distinguish graph boundaries. In streams emitting predictable RDF graphs, it is usually sufficient to let the boundaries of event objects be defined by windows that slide across the stream. However, this solution is not sufficient if neither the time interval between events, nor the number triples in each event object, is known ad remains static.

In this paper, we have presented four possible solutions, ranging from adding a single trailing triple to the end of each graph, to solutions that require mod-

ifications to the way in which RDF data streams are represented, and the way in which they are typically consumed by RDF stream processing engines.

Future challenges involve practical evaluation, and standardization, of the various solutions. It would, e.g., be necessary to develop vocabularies to express various types of headers, and trailing triples. It is also necessary to study the efficiency of the various solutions empirically, to ensure that low latency, necessary in a number of domains [12], can be maintained.

The various solutions can also be combined in various ways. As an example, domains requiring near real-time execution could benefit from event object decomposition that optimizes the stream for the most common type of query, even if this solution alone does not provide support for event object boundaries.

## References


1. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Proceedings of the 20th international conference on World wide web. pp. 635–644
2. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The VLDB Journal (2), 121–142 (Jul.)
3. Barbieri, D.F., Braga, D.: An execution environment for C-SPARQL queries. In: Proceedings of the 13th International Conference on Extending Database Technology. pp. 441–452
4. Barbieri, D.F., Della Valle, E.: A proposal for publishing data streams as linked data - a position paper. In: Proceedings of the Linked Data on the Web LDOW2010 Workshop, co-located with WWW2010
5. Goodwin, J.C., Russomanno, D.J.: Ontology integration within a service-oriented architecture for expert system applications using sensor networks. Expert Systems (5), 409–432 (Nov.)
6. 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 Proceedings of the 10th international conference on the semantic web. pp. 370–388. No. 24761
7. Le-Phuoc, D., Parreira, J.X., Hausenblas, M., Hauswirth, M.: Unifying stream data and linked open data
8. Luckham, D., Schulte, R.: Event Processing Glossary Version 2.0 (Jul.)
9. Morris, G.W., Thomas, D.B., Luk, W.: FPGA Accelerated Low-Latency Market Data Feed Processing. 17th IEEE Symposium on High Performance Interconnects pp. 83–89 (Aug.)
10. Scharrenbach, T., Urbani, J., Margara, A., Della Valle, E., Bernstein, A.: Seven Commandments for Benchmarking Semantic Flow Processing Systems. The Semantic Web: ... pp. 305–319
11. Sequeda, J.F., Corcho, O.: Linked stream data: A position paper. In: The 2nd International Workshop on Semantic Sensor Networks
12. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. ACM SIGMOD Record (4), 42–47 (Dec.)
13. Zhang, Y., Duc, P., Corcho, O., Calbimonte, J.P.: SRBench: A Streaming RDF/SPARQL Benchmark. In: Proceedings of International Semantic Web Conference (ISWC 2012). pp. 641–657