# SPARQL Update for Complex Event Processing

Mikko Rinne

Distributed Systems Group,
Department of Computer Science and Engineering,
Aalto University, School of Science, Finland
mikko.rinne@aalto.fi

**Abstract.** Complex event processing is currently done primarily with proprietary definition languages. Future smart environments will require collaboration of multi-platform sensors operated by multiple parties. The goal of my research is to verify the applicability of standard-compliant SPARQL for complex event processing tasks. If successful, semantic web standards RDF, SPARQL and OWL with their established base of tools have many other benefits for event processing including support for interconnecting disjoint vocabularies, enriching event information with linked open data and reasoning over semantically annotated content. A software platform capable of continuous incremental evaluation of multiple parallel SPARQL queries is a key enabler of the approach.

**Keywords:** Complex event processing, SPARQL, RDF, Rete-algorithm

## 1 Smart Cities Need SPARQL

Smart environments of the future will need to interconnect billions of sensors based on platforms from multiple vendors operated by different companies, public authorities or individuals. To mitigate the need for overlapping sensors producing duplicate measurements, interoperation of different platforms should be maximized. Highly distributed, loosely coupled solutions based on common standards are needed in such open environments. Event processing systems based on proprietary definition languages have challenges to adapt to multi-vendor contexts.

The benefit of RDF in complex event processing is that it provides a flexible representation of heterogeneous events in an open distributed environment, where new sensors must be able to add new information fields without breaking compability with existing applications. SPARQL, tailor-made to query RDF, was augmented in SPARQL 1.1 Update by the powerful capability to insert selected data into named triple stores. When combined with a continuous query processing engine, INSERT gives SPARQL queries memory and capability to communicate and collaborate with each other. As a result, interconnected SPARQL queries can be used to create complex event processing applications, capable of handling layered and heterogeneous representations of event instances. When taking into account their other benefits, semantic web standards RDF, SPARQL and OWL form a very promising base for complex event processing.

In the Distributed Systems Group we are working on an incremental continuous SPARQL query processor based on the Rete-algorithm [5]. The INSTANS[1] platform supports selected parts of SPARQL 1.1 Query and Update specifications. The first generation of INSTANS was coded on Scala[2] [1, 8–10]. INSTANS is currently being ported to Lisp, where the Rete-net is compiled through macro expansion in the setup phase into executable Lisp code. The Scala-version reached notification delays of 5-14 ms for the cases tested, but first measurements indicate that the Lisp-version would be 100-200 times faster.

In event processing it is equally important to detect the events which didn't happen as the ones that did. Missing events are sometimes referred to as "no-events" or "absence patterns" [4]. A "timed events" mechanism is implemented with special predicate values used to mark input to a timer-queue. Events in the timer queue can be set to trigger either after a relative time or at absolute points in time. A triggered event can be used to set a new timed event, supporting periodic operations. The whole interface is SPARQL-compliant, with the triggering of a timer changing a corresponding triple predicate from "waiting" to "triggered", the change being detectable in a SPARQL query.

## 2 Related Activities

Other research teams have been looking into streaming SPARQL, e.g. C-SPARQL[3] [3] and CQELS[4] [7]. Some differences to our approach are:

- Individual triples: "Data stream processing" focuses on individual time-annotated triples. We are assuming heterogeneous event formats, where it may not be known at the time of writing an event processing application, what information future sensors are going to include into an event. Possibility to layer events is also of critical importance.
- Extensions: All other solutions extend SPARQL, typically with time-based windowing or processing a stream order of data. We have used no extensions.
- Repetition of queries: Defined on windows based on time or number of triples and a repetition rate, with which queries will be re-run. Our approach is based on continuous and incremental matching of queries, where a particular segment can be isolated by filtering.

Sparkweave[5] [6] applies SPARQL queries to RDF format data using an extended Rete-algorithm, but focuses on inference and fast data stream processing of individual triples instead of heterogeneous events. Sparkweave v. 1.1 also doesn't have support for SPARQL 1.1 features such as SPARQL Update.

[1] Incremental eNgine for STANding Sparql, http://cse.aalto.fi/instans/
[2] http://www.scala-lang.org/
[3] http://streamreasoning.org/download
[4] http://code.google.com/p/cqels/
[5] https://github.com/skomazec/Sparkweave

The Prolog-based ETALIS has a SPARQL compiler front-end called "EP-SPARQL" [2], but it is more limited than the Prolog notation and doesn't support (at the time of writing) SPARQL 1.1 features such as SPARQL Update, which is critical for our study. EP-SPARQL concentrates on operations on event sequences.

No other system based on collaborative SPARQL queries is known to us. Current systems in the research community are mainly concentrating on running one query at a time[6]. Even the ones allowing to register multiple simultaneous queries are not expecting the queries to communicate during runtime.

## 3 Measuring Success

Our event processing work focuses on two main components:

1. **Approach:** Multiple collaborating SPARQL queries and update rules processing heterogeneous events expressed in RDF.
2. **Implementation (INSTANS):** Incremental continuous query engine based on the Rete-algorithm

The overall target of the approach is that it would be easy to create and maintain efficient event processing applications for open and heterogeneous environments. Research questions are related to finding good principles and patterns for SPARQL queries used in event processing, creating a mapping to SPARQL for the main operations needed in event processing (e.g. filtering, splitting, enrichment, aggregation, pattern detection), developing efficient methods of linking event information with background knowledge, adopting ontology-based inference mechanisms in event processing and comparing to other event processing approaches.

An example application "Fast Flowers Delivery" is presented in [4]. It is a logistics management system, where flower stores send requests to an independent pool of drivers to send flowers to customers. Drivers are selected based on location and ranking. Ranking involves a periodic reporting system. Our next target is to verify that SPARQL has all the elements in place to support also this kind of event processing applications. Once the example cases have been confirmed to work, generalized solution patterns for the complex event processing elements found in literature using SPARQL building blocks will be defined.

Measuring the success of the implementation can be approached with:

- Implementation efficiency (compared to other Rete implementations)
- Algorithmic efficiency (Rete compared to other ways of processing SPARQL queries)
- Performance of the approach (compared to other event processing systems)

Targets for empirical studies are e.g. latency (notification time), throughput, memory consumption, system load, continuous operation over extended time

[6] e.g. Jena (http://incubator.apache.org/jena/), Sesame (http://www.openrdf.org/)

periods and energy efficiency (especially when operating over sensors). In addition to empirical comparisons, this work is expected to provide answers for understanding of garbage build-up in the system and for solutions to improve performance compared to basic Rete.

As a first step comparisons with C-SPARQL have been carried out and documented on the project homepage using an example "close friends" service[8], but since C-SPARQL is based on repeated execution of queries on windows, the results are very difficult to compare. A "notification delay" in C-SPARQL is dominated by the window repetition rate. Trying to minimize delay by increasing repetition rate leads to wasted computing resources and duplicate detections. Even doing so, the format of C-SPARQL only allows to execute queries once per second (far too often for most applications), whereas the notification delays for INSTANS have been clocking in at 5-14 ms (depending on hardware).

Both the approach and the implementation would be involved in testing the ease of deployment and management of the system in a distributed way in an open environment. Related questions are the processing and memory requirements of the implementation, arrangements for communication between distributed deployments and any security-related issues specific to the approach. Based on our verifications both the approach and INSTANS look very promising.

## References


1. Abdullah, H., Rinne, M., Törmä, S., Nuutila, E.: Efficient matching of SPARQL subscriptions using Rete. In: Proceedings of the 27th Symposium On Applied Computing. Riva del Garda, Italy (Mar 2012)
2. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. pp. 635–644. WWW '11, ACM, Hyderabad, India (2011)
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 - EDBT '10. p. 441. Lausanne, Switzerland (2010)
4. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning Publications (Jul 2010)
5. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17–37 (Sep 1982)
6. Komazec, S., Cerri, D.: Towards Efficient Schema-Enhanced Pattern Matching over RDF Data Streams. In: 10th ISWC. Springer, Bonn, Germany (2011)
7. 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)
8. 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)
9. Rinne, M., Nuutila, E., Törmä, S.: INSTANS: High-Performance Event Processing with Standard RDF and SPARQL. In: Poster in International Semantic Web Conference 2012. Boston, MA (2012)
10. Rinne, M., Törmä, S., Nuutila, E.: SPARQL-Based Applications for RDF-Encoded Sensor Data. In: 5th International Workshop on Semantic Sensor Networks (2012)