SPARQL-Based Applications for RDF-Encoded Sensor Data

Mikko Rinne, Seppo Törmä, and Esko Nuutila

Department of Computer Science and Engineering,
Aalto University, School of Science, Finland
firstname.lastname@aalto.fi

Abstract. Complex event processing is currently more dominated by proprietary systems and vertical products than open technologies. In the future, however, internet-connected people and things moving between smart spaces in smart cities will create a huge volume of events in a multi-actor, multi-platform environment. End-user applications would benefit from the possibility for open access to all relevant sensors and data sources. The work on semantic sensor networks concerns such open technologies to discover and access sensors on the Web, to integrate heterogeneous sensor data, and to make it meaningful to applications. In this study we address the question of how a set of applications can efficiently access a shared set of sensors while avoiding redundant data acquisition that would lead to energy-efficiency problems. The INSTANS event processing platform, based on the Rete-algorithm, offers continuous execution of interconnected SPARQL queries and update rules. Rete enables sharing of sensor access and caching of intermediate results in a natural and high-performance manner. Our solution suggests that with incremental query evaluation, standard-based SPARQL and RDF can handle complex event processing tasks relevant to sensor networks, and reduce the redundant access from a set of applications to shared sensors.

Keywords: Complex event processing, SPARQL, RDF, Sensor systems

1 Introduction

Our future, with internet-of-things, is filled with sensors. Devices in our personal area network communicate with each other, with the things we own and with the services we are interested in. Connected devices form smart spaces, made to assist us in our daily tasks. Smart spaces interact with infrastructural sensor networks, forming smart cities. In the scale of cities the number of sensors can reach billions, and sensor observations be extremely heterogeneous due to differences in stimuli, vendors, software versions, operators, administrative domains etc.

At the same time there will be increasing numbers of applications that would need to access sensor observations. It would be wasteful for each application - or a closed group of applications - to deploy its own set of sensors. A lot of duplication could be avoided, and hardware and communication resources used

more efficiently, if sensors were openly exposed on the Web. Applications could be given broader access to sensors without locking application-sensor pairs to vertical silos. If access from applications to shared sensors is enabled, some new problems will arise [14]: How to discover, access and search sensor data? How to integrate the sensor data coming from heterogeneous sources? How to make sensor data meaningful to applications?

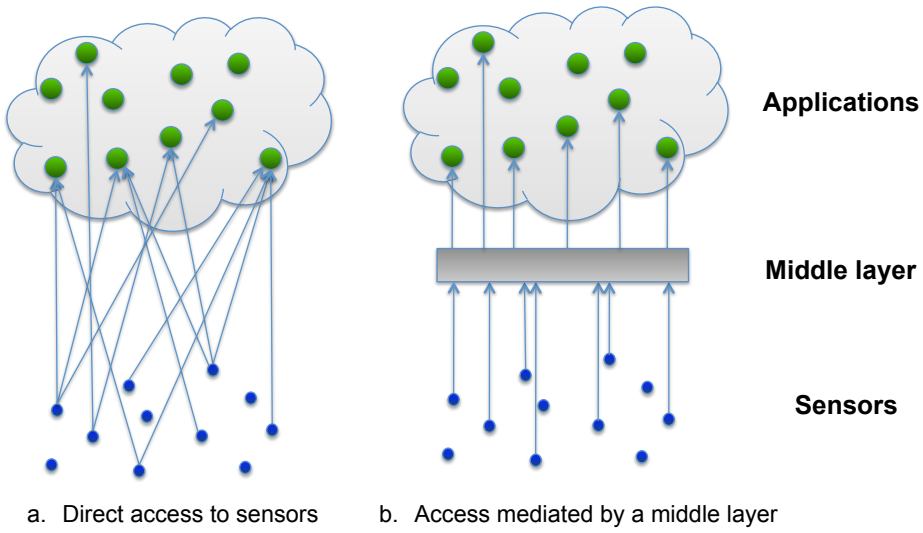

Fig. 1: Bridging between sensors and applications

A set of models to address these problems has been presented: Sensor Web Enablement (SWE) by OGC[1] [6], the model of Semantic Sensor Networks Incubator Group (SSN-XG) of W3C [14], the architecture of sensor Web applications in [7], and the Cloud, Edge, and Beneath model (CEB) by Xu et al [20]. They are all three-layer models where access from applications to sensors is mediated by a middle layer, as shown in Figure 1. In open sensor systems there are several needs for the middleware:

- *Abstraction*: The information from sensors needs to be layered to reasonable levels of abstraction, already for programmers but even more so for human end-users, who should only be notified or alerted with information significant to their personal contexts.
- *Interoperability*: These sensors, which can be mobile phones, thermometers, weather cameras or train positioning systems, are manufactured, owned and operated by various companies, public authorities and private persons. They

[1] Open Geospatial Consortium, http://www.opengeospatial.org/

will not operate under the same standard or service. There is a need for flexible representations for semantic relations of data from different origins.
- *Energy efficiency*: Many sensors will be depending on limited local power sources, and in the long run the applications, in total, can consume significant amounts of energy. Access to sensors should be managed in order to minimize unnecessary and wasteful work, in particular *redundant data acquisition* [20]. Redundancy can be minimized by sharing the work across applications to access the sensors, by caching intermediate results, and by suppression of irrelevant sensor input.

In this study we work on the basis of the *event* as an abstraction of a meaningful change in sensor readings. It is assumed that the primary operation of the system is based on sensors that *report events* in push-mode. This corresponds to the approach of Sensor Event Service (SES) [6] of the Sensor Web Enablement, although for the specification of complex events we rely on incremental evaluation of standard SPARQL queries and update rules [18] instead of the Event Pattern Markup Language (EML) [9] used in SES.

According to [15] a *complex event* is "an event that summarizes, represents, or denotes a set of other events". With this definition, "complex event processing" becomes defined by the *layering* of events, rather than the complexity of the underlying event processing problem, or the method used to solve it. This layering gives us the abstraction we need to hide the millions of events and come up with human-tangible conclusions like "the bus is late", "take this route to the office instead of your usual one" or "your house is probably on fire". So whereas 'simple event' is a production-oriented concept closer to sensor observations, 'complex event' is a consumption oriented concept: a specification of an event that an application or a user is interested in.

Interoperability is the central promise of semantic web technologies. They make it possible to establish relations both between ontologies and between instance data originating from different domains and sources. The expressive representation and query capabilities allow the flexible use of these technologies across domains. Event information can be enriched with linked data in the web, and the availability of inference tools enable the reasoning about event content.

If a set of applications were to access a shared set of sensors independently, there would be a lot of redundant acquisition, communication, and processing of sensor data. In this study we address the question of how a set of applications can efficiently access a shared set of sensors while avoiding unnecessary redundancy[2]. Our solution can be implemented using the Rete-algorithm that turns out to be a good fit for the task: it avoids the duplicate processing of events, it enables the sharing of sensor access and intermediate processing steps between applications, and it caches the intermediate results for efficient access later on. The big advantage of Rete is the high performance that manifests in short notification delays when the last piece of information that satisfies a query

[2] Some amount of controlled redundancy can be desirable for failure detection purposes [19]

is received. The natural place for Rete network is on the middle layer between sensors and applications but we discuss also its use on other layers.

The INSTANS event processing platform is based on continuous execution of interconnected SPARQL queries and update rules. We have presented how Rete-based incremental query evaluation enables efficient complex event processing with standard SPARQL and RDF [18]. This paper suggests that Rete is also suitable for reducing redundant access from applications to shared sensors.

The structure for the rest of this paper is the following: The principle of using collaborating SPARQL queries for event processing is introduced in Section 2. Our INSTANS platform is explained in Section 3. Section 4 explains the general approach of using Rete at different layers. Section 5 reviews some examples of potential application scenarios. Section 6 briefly reviews related work. Conclusions and future plans are presented in Section 7.

2 Event Processing Based on SPARQL

SPARQL is an expressive query language on RDF graphs. It can be used in a straightforward manner to filter events, construct new derived events, and specify complex patterns concerning the properties of multiple events. However, a single SPARQL query is not sufficient for many complex event processing applications [18]. SPARQL 1.1 Update[3] adds a critically important new feature: the capability to INSERT data into a triple store. When operated in an environment capable of continuous execution of multiple parallel SPARQL queries, the output of one query can be the input of other queries, as described in more detail in [18]. This way queries can store intermediate information for later use and pass information between each other, creating an entire event processing application out of a collaborative network of queries in standard SPARQL.

Compared to repeated execution of queries over time-based windows (as used in e.g. [3, 13]), continuous processing of SPARQL queries has clear benefits [18]:

1. *Instant availability of results.* For a memory-resident event processing application results are typically available in a few milliseconds. In most applications it would not be practical to re-run queries over event windows repeatedly with such rates.
2. *No duplicate detections due to overlapping windows.* With overlapping event windows same events can be processed more than once, resulting in duplicate notifications of exactly the same event instances.
3. *No missing detections on window borders.* If event windows do not overlap, event patterns crossing window borders will not be detected. To reduce the number of misses, the window length could be made longer but that would again increase the notification delays.
4. *No repeated processing over the same data.* The SPARQL queries and update rules can rely on the fact that each event is processed only once.

[3] http://www.w3.org/TR/sparql11-update/

The chaining and possibility to store intermediate results allows SPARQL queries to collaborate, forming an event processing application.

In window-based approaches to data stream processing such as [3, 13], the window lengths are typically based on either time duration or a number of triples. This approach is usually coupled with the assumption that each single RDF triple marks a standalone event. The input from sensors, however, could well contain any number of triples. There can be different sensor types providing partially overlapping information, different vendors and even different software versions of the same sensor. To be able to use all applicable sensors, we need to be able to support heterogeneous event formats. An example is illustrated in Figure 5, where the sensor sending location updates may or may not include altitude information. In open environments the existing event processing application should not be broken by additions of new event formats.

3 INSTANS Event Processing Platform

To address the requirement of near-real-time processing of complex, layered, heterogeneous events, we have created INSTANS[4] [1, 18]. Based on the Rete-algorithm [10, 11], INSTANS does continuous evaluation of incoming RDF[5] data against multiple SPARQL[6] queries. Intermediate results are stored into a β-node network. When all the conditions of a query are matched, the result is instantly available.

Again following the conventions of [15], an event processing network (EPN, Figure 2) consists of event processing agents (EPA, 1., 3., 4.) connected with event channels (2.). An output-only EPA is an event producer (1.), an input-only EPA is an event consumer (4.). In practice the same EPA can assume different roles in different contexts: The event consumer of one EPN can be the event producer of another one. In a sensor network context sensors are typical event producers and applications are typical event consumers. INSTANS can appear in any of the three EPA roles. Using RDF both for input and output means that different instances of INSTANS can be connected together. SPARQL queries can talk to each other both in micro-scale within one INSTANS-EPA and between different instances of INSTANS, offering very flexible possibilities to build a distributed system. For example:

- The computation of a single Rete can be distributed between parallel processors and / or processor cores
- When using multiple instances of Rete, the output of one EPA can be used to set the parameters of another EPA, e.g. the reporting trigger parameters of a sensor platform.

[4] Incremental eNgine for STANding Sparql, http://cse.aalto.fi/instans/
[5] http://www.w3.org/RDF/
[6] http://www.w3.org/standards/techs/sparql#w3c_all

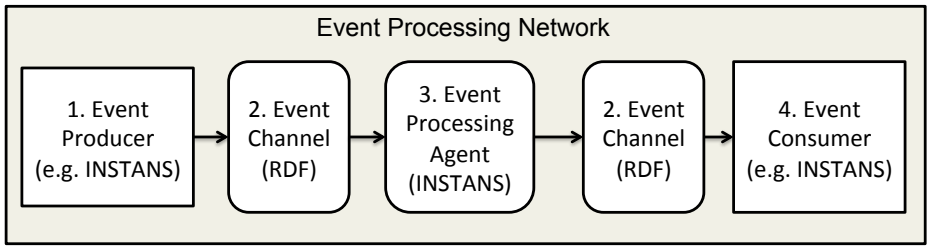

Fig. 2: Event Processing Network (EPN) architecture showing INSTANS

The first version of INSTANS was coded on the Scala language[7]. Even though Scala has a lot of flexibility, it isn't built to support runtime generation of code. To exploit more dynamic programming techniques, we are working on a Rete implementation in Common Lisp. In Lisp, the Rete-net is compiled in setup-phase through macro expansion to executable Lisp code. The first results are highly encouraging: The close friends example, introduced in Section 5, produces the same results but runs 100-200 times faster than our original Scala implementation.

INSTANS processes each incoming triple immediately for every matching condition and saves intermediate results into its beta-node structure, as illustrated in Figure 3 using a query example from the application discussed in Section 5.2 involving the approach for timed events presented in Section 3.1. This example starts a timer to track the committed pickup time of a delivery task, when the task is assigned to a bidding driver.

The main drawbacks of the Rete-algorithm are perceived to be memory consumption due to the saving of intermediate results within the structure and the slow processing of deletions [17, 16]. The memory consumption can be decreased by recognizing recurring patterns in queries and combining the corresponding nodes. Deletion can be made significantly faster by better indexing of the nodes.

3.1 Timed Events

The asynchronous nature of INSTANS means that all input is processed when it arrives. To support synthetic events at specific points in time like the detection of a missing event or the compilation of a report, the concept of timed events is needed. Timed events are built into INSTANS with the help of a special "timergraph" and a set of special predicates:

- **timer_sec / min / hour:** object (integer) specifies time-until-trigger in seconds / minutes / hours
- **timer_date:** triggers after the specified number of days at midnight
- **timer_month:** triggers after the specified number of months on the first day of the month at midnight

[7] http://www.scala-lang.org/

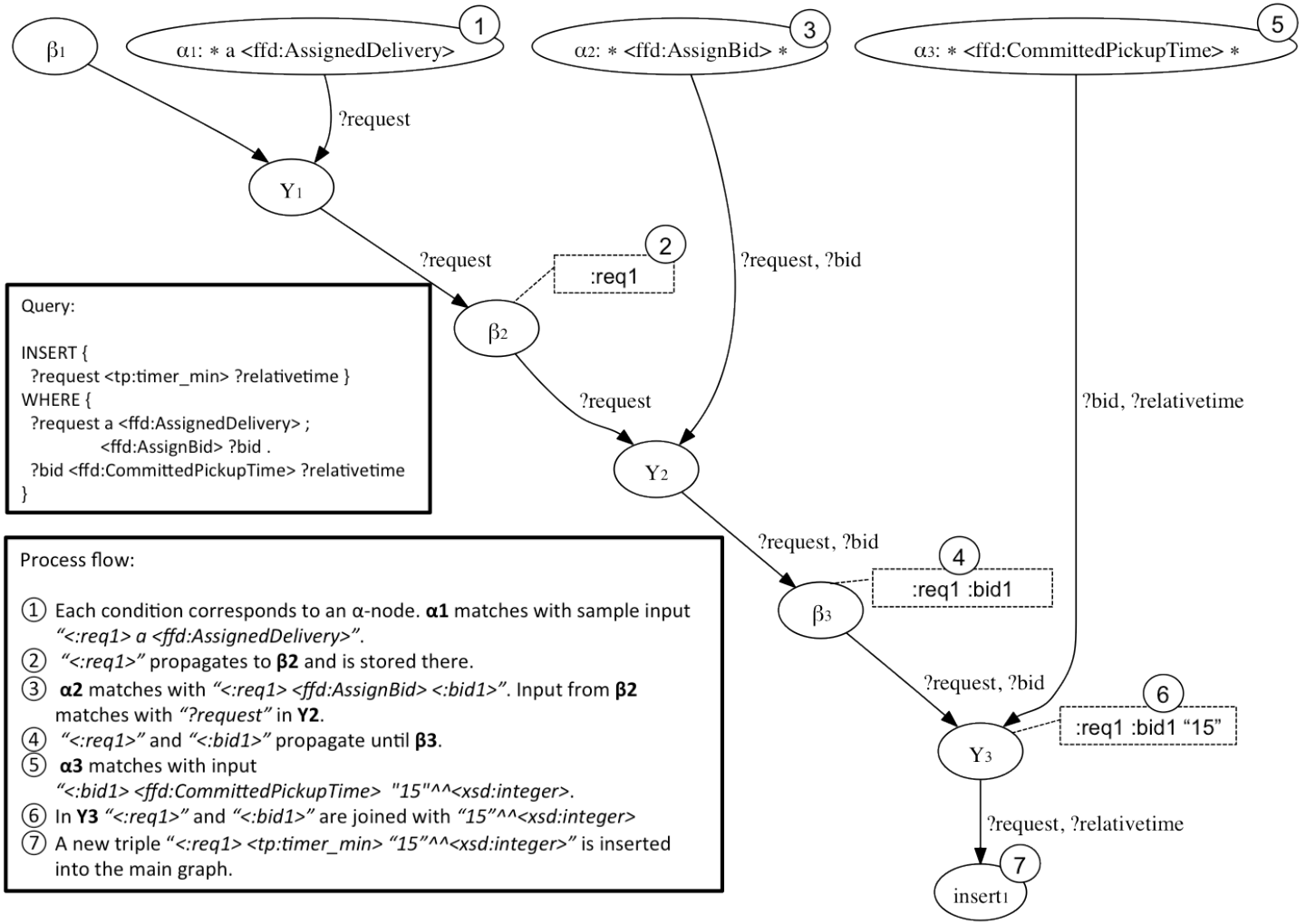

Fig. 3: Example of SPARQL query processing in a Rete-net

– **timer_year:** triggers after the specified number of years on the first day of the year at midnight
– **timer_abs:** object (dateTime) specifies the absolute time for trigger

When inserted into the special timer queue, the objects are converted to absolute date-time values according to current system time and all predicates are set to <tp:waiting> status. Actors are used to schedule wakeup, at which time the predicate of triggered timers is changed to <tp:triggered>. Using the following query:

```
INSERT { GRAPH <http://example.org/timergraph> {
   ?event <tp:timer_sec> ?timevalue } }
WHERE { ?event <:seconds> ?timevalue }
```

A five-second timer would start, when receiving:

```
<:5sec_pulse> <:seconds> "5"^^<xsd:integer>
```

A SPARQL query tuned to monitor a certain timer triple can react to the change in predicate and carry out the necessary actions. One possibility is to set a new timer, creating a pulse generator:

```
WITH <http://example.org/timergraph>
INSERT { <:5sec_pulse> <tp:timer_sec> 5 }
WHERE { <:5sec_pulse> <tp:triggered> ?triggertime }
```

4 Use of INSTANS in Sensor Applications

INSTANS can be utilized on all three layers of Figure 1. Naturally, each of the applications on the uppermost layer could use INSTANS separately to process any kind of incoming events – simple or complex. This is especially pertinent to those processing needs that are specific to that particular application. INSTANS will still yield performance advantages when compared to complex event processing solutions based on repeated queries on stream windows.

At the middle layer significant efficiency improvements can be achieved in those event processing tasks – filtering, aggregation, enrichment, and pattern detection – that can be shared among multiple applications. When deployed at the middle layer, INSTANS would accept subscriptions from applications in the form of complex event patterns presented as SPARQL queries and update rules. The subscriptions of all applications are compiled into a common Rete network in which similar query structures are shared among different applications. As INSTANS receives events from the sensor layer, they are immediately processed though the Rete network triple by triple. Each triple is propagated as far as it can proceed through the network. It may be filtered away, or the propagation may stop in a join node in which no matching data from the other branch can be found. The data content of the triple remains in a beta memory waiting for potential future matches.

The efficiency benefits at the middle layer are a direct result of the well-known properties of the Rete algorithm. Each event is evaluated only once against similar query structure, the state of partially completed propagation is memorized, and the notifications of matches are produced immediately when a pattern becomes satisfied.

At the lowest layer, INSTANS can be used in sensor platforms that have enough processing power and memory to run the Rete network. The main role of INSTANS would be to construct meaningful events out of the mass of observations. The overall goal is to reduce the amount of communication – which is usually the most significant component in the energy consumption of sensor platforms – by some additional computation in event processing. As a result, a large volume of sensory observations may turn out to produce only few meaningful events that need to be communicated upwards.

A meaningful event is one that is *relevant* in the sense that it can match some query structures, and *significant* in the sense that it can affect the result of a query. In Rete only *changes in observed values* are significant. This can even be generalized: only deviations from expectations are significant. It means, for instance, that only those observed values that deviate from an expected trend should be reported. The upper layers are responsible to communicate to lower layers what kinds of events are relevant and significant. The reconfiguration can be done by executing SPARQL Update commands to lower layers with appropriate configuration data. The basic flow is illustrated in Figure 4.

First (1.) the application has a new reporting requirement such as an updated temperature threshold. This triggers a query in an instance of INSTANS (2.), emitting an RDF-format update request (3.):

```
<:outdoor_sensor1> <:min_reporting_temperature> "22"^^<xsd:integer>
```

Another instance of INSTANS in the sensor platform receives the configuration information and replaces the earlier local parameter with the new configuration (4.):

```
INSERT { GRAPH <http://sensorplatform.org/local_configuration> {
   <:outdoor_sensor1> ?parameter ?newvalue } }
WHERE { <:outdoor_sensor1> ?parameter ?newvalue }
```

After the new configuration is set, only temperature readings higher than 22 get reported (5.):

```
INSERT { GRAPH <http://sensorplatform.org/sensor_output> {
   <:outdoor_sensor1> <:temp> ?reading } }
WHERE {
   GRAPH <http://sensorplatform.org/sensor_input> {
      <:outdoor_sensor1> <:temp> ?reading }
   GRAPH <http://sensorplatform.org/local_configuration> {
      <:outdoor_sensor1> <:min_reporting_temperature> ?reading }
   FILTER (?reading > ?min_temperature)
}
```

Beyond this bare bone example, a more elaborate SPARQL-query would take into account other parameters and trigger the report either periodically using

timed events, or only send the event once when the temperature rises above the threshold, depending on application requirements.

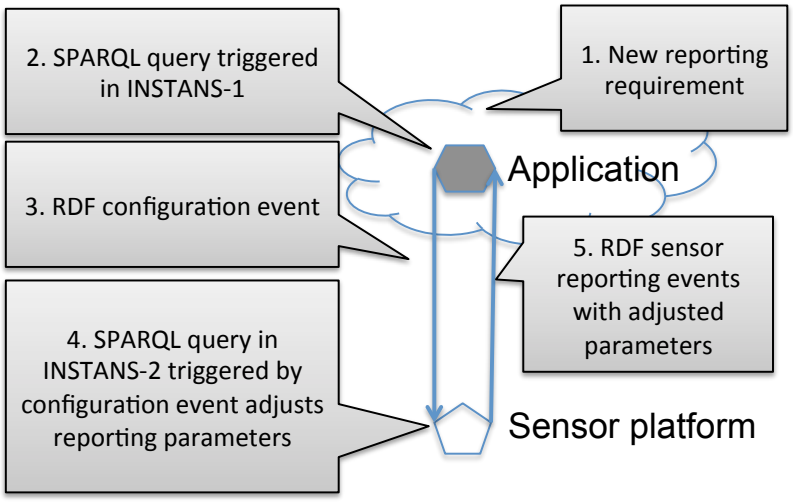

Fig. 4: Flow of operation using INSTANS both in the layer of applications and as the reporting configuration platform at the middle layer.

In the wider context of a sensor network this approach could be used to manage the reporting from different sensors based on application requirements. Instead of each application going to an middle layer or all the way to the sensor to request (overlapping) measurements, service management in the application layer should compile all requests towards a sensor into a reporting scheme, which would satisfy the requirements from all applications with minimum access to the sensor.

5 Examples Scenarios

Below are two example scenarios used as test cases for INSTANS.

5.1 Close Friends [18]

Subscribers in the "Close Friends" scenario form a social network defined by foaf:knows properties. They report their locations with events like the one shown in Figure 5. When two friends are nearby, the system recognizes the situation and reports a "nearby" status. It is able to avoid repeated detections as long as the condition persists and build hysteresis into detecting, when the same persons are far enough to reset the "nearby" status.

Each subscriber would be running a Close Friends application in his or her mobile device. The sensor layer is formed by the location sensors of the same devices. The middle layer receives runs INSTANS: it receives location updates from mobile devices and sends nearby events to applications.

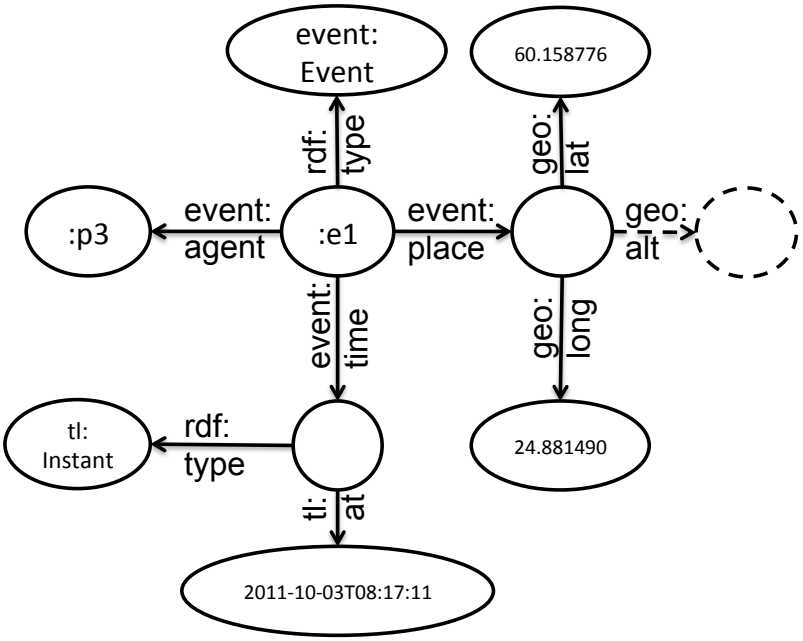

Fig. 5: Location Event Format

To generate input data, an event simulator has been created. The simulator can use map data from Open Street Map[8] and place a number of persons to move within the map area, generating location events. It is possible to output both to a file and to generate input data for INSTANS over a live TCP connection, slowing the simulator down to real-time operation. A screen capture of the event simulator is shown in Figure 6.

The detection of a "nearby" condition requires location update events from two separate persons. This setting works very well in the asynchronous environment of INSTANS: We only need to compare the two latest location events from two friends, checking that neither event is expired and we can get a match. In a system based on repeated execution of queries over time-based windows the requirements would contradict: To save power and network resources, location reporting frequency should preferably adapt to the movement of the person, but in a window-based system the achievable notification delay is limited by the

[8] http://www.openstreetmap.org/

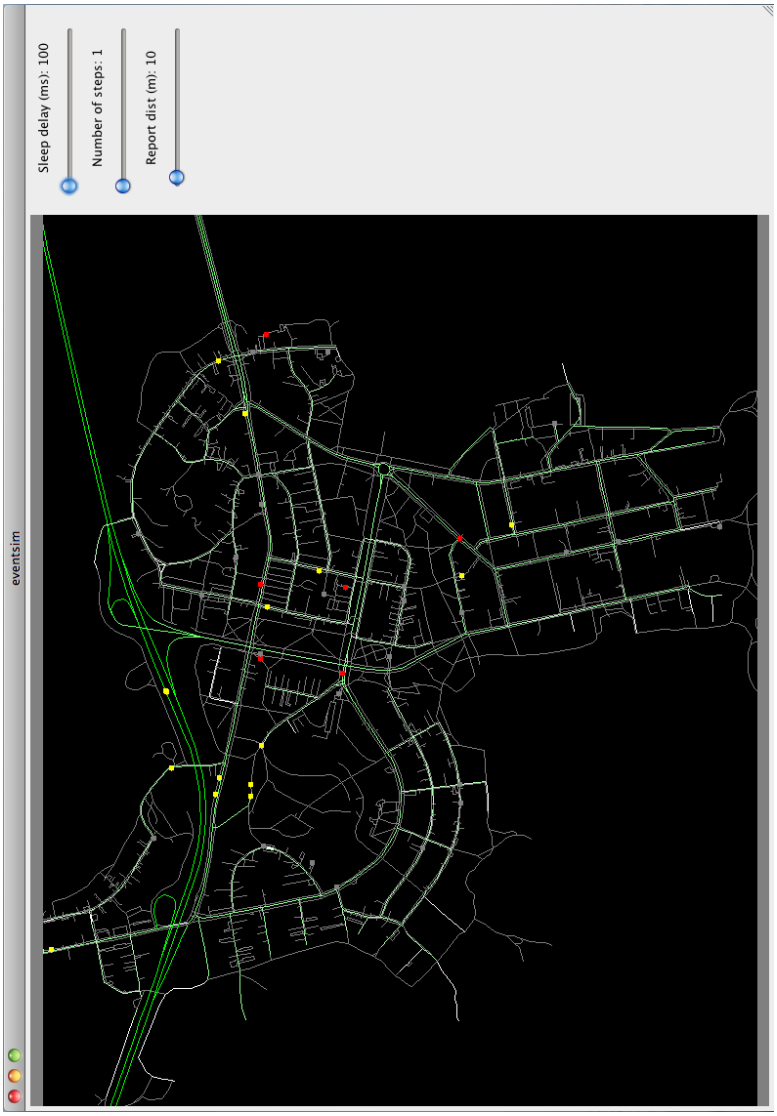

Fig. 6: Screen capture of the event simulator showing 50 persons moving around the island of Lauttasaari in Helsinki

repetition rate of the window. If the window is very long compared to repetition rate, the same nearby-condition is detected multiple times. If the repetition rate is slow, detection delay increases. And if the window is short, the reporting frequency from terminals needs to be artificially increased to make sure that all users report within the window. In INSTANS none of these compromises are needed; reporting rate can be adjusted based on detected location changes and notifications are available only a few ms after reporting, much faster than would be feasible as a window repetition rate.

Support of heterogeneous event formats is also easier, when there are no windows defined by time or number of triples, which could break events consisting of multiple triples. Whether altitude is included in the example of Figure 5 or not, "close friends" operates the same way. As long as there is a way to link all the triples of an event together (e.g. a common subject), each application can use the triples they need and copying or deleting of the entire event can be done by matching the linking information. In "close friends" event identifier :e1 can be used to copy or delete all triples of the event without explicit knowledge of what parameters are included.

With the first-generation INSTANS programmed on Scala the notification delay from the creation of the second qualifying event to the reporting of the "nearby" detection was about 12 ms on a 2.26 GHz Core 2 Duo Mac running OS X 10.6.

5.2 Logistics Management

The "Fast Flowers Delivery" (FFD) event processing application, a typical logistics management task including reporting, is detailed in [8]. Flower stores request delivery service for flower orders, independent drivers bidding for the assignment based on availability, location and driver ranking. Demo implementations[9] on six different platforms are available at the book website, but none of them are coded in SPARQL.

Timed events are in heavy use in FFD. Each phase of the flower delivery needs to be monitored for time: Driver response to bid request, assignment of driver, pickup and delivery of flowers. Additionally the reports and driver ranking need to be processed at designated times.

When using a single heterogeneous event to describe a delivery task, taken through multiple phases, new IRIs need to be generated to keep the timers unique. SPARQL offers good tools for this: the subject IRI of a request can be bound together with a status string or another IRI to generate a new IRI, which can be used as a unique identifier.

6 Related Work

So far we have not been able to find another system in the research community, which would enable continuous processing of SPARQL 1.1 queries, including the

[9] http://www.ep-ts.com/content/view/79/

SPARQL Update methods of communication between queries. Sparkweave[10] [12] applies SPARQL queries to RDF data using an extended Rete-algorithm, but focusing on inference and fast data stream processing of individual triples instead of heterogeneous events. Sparkweave v. 1.1 also doesn't support SPARQL 1.1 features such as SPARQL Update.

Data stream processing focusing on fast filtering of individual triples, not events consisting of multiple triples, and time- or triple-based repetition of queries on windows have been the most popular approaches used by SPARQL-based stream processing systems such as C-SPARQL[11] [3–5] and CQELS[12] [13].

Jena[13] and Sesame[14], both popular platforms in the research community, have support for SPARQL Update, but not for multiple simultaneous queries.

The field of complex event processing is even more diverse. Popular tools include but are not limited to TIBCO BusinessEvents[15], StreamBase[16], Esper[17], Aleri[18], Apama[19] and ETALIS[20], which has a SPARQL pre-compiler called EP-SPARQL [2] supporting a subset of ETALIS features. The summary of CEP market at 2011[21] compiled by Paul Vincent of TIBCO includes 23 different systems. We haven't looked at all of them, but so far we have not come across any system based on the use of collaboration of standard-compliant SPARQL v. 1.1 in the way we have described.

7 Conclusions and Future Plans

Semantic web standards RDF and SPARQL are well suited for complex event processing due to their inherent strengths in managing multi-vendor multi-platform environments. In addition to simple conversions between vocabularies, inference capabilities and access to all linked open data they are also likely to be useful in developing semantic reasoning and enriching of event data.

Based on initial testing of the event processing tasks described in Section 5, the central paradigm of INSTANS – continuous incremental matching of multiple standard-based SPARQL queries supporting inter-query communication – seems to be managing well. When complemented with support for timed events we have

[10] https://github.com/skomazec/Sparkweave
[11] http://streamreasoning.org/download
[12] http://code.google.com/p/cqels/
[13] http://incubator.apache.org/jena/
[14] http://www.openrdf.org/
[15] http://www.tibco.com/
[16] http://www.streambase.com/
[17] http://esper.codehaus.org/
[18] http://www.sybase.com/products/financialservicessolutions/complex-event-processing
[19] http://www.progress.com/en/Product-Capabilities/complex-event-processing.html
[20] http://code.google.com/p/etalis/
[21] http://www.thetibcoblog.com/wp-content/uploads/2011/12/cep-market-dec2011.png

found no show stopper problems, which would render the approach unusable for any complex event processing task.

Compared to systems based on repeated execution of windows the approach used in INSTANS has multiple benefits. Notification delays in the order of milliseconds cannot be practically competed with by re-running queries at similar rates. While other systems re-run queries at fixed time intervals or after a fixed number of triples, INSTANS combines similar parts of the queries, processes each input triple as soon as it becomes available and memorizes intermediate results, resulting in significantly smaller amount of duplicate computation.

In the context of a sensor network, INSTANS could be deployed as a programmable and reconfigurable platform both close to the sensors and as a central agent. It is distributable on various levels. Configuration and operation are fully based on semantic web standards without any mandatory extensions, with SPARQL used as the programming language to create event-based applications and RDF used for communication. Due to the compatible approach of using RDF both for input and output, even large event processing tasks can be split into manageable sizes and layered abstractions.

After finalizing the transition to the Lisp-based platform, we are planning to attempt a more complete solution of the Fast Flowers Delivery application to verify that everything can be properly supported. To prepare for true big data usage, longer runs of the platform should be combined with testing of different kinds of garbage handling approaches. Support for querying external SPARQL endpoints as well as support for a selected set of inference rules are being planned.

Acknowledgments

This work was partially supported by Tekes[22] through the funding to RYM-PRE[23] (Built Environment Process Re-Engineering) projects, and by European Commission through the SSRA (Smart Space Research and Applications) activity of EIT ICT Labs[24].

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. In: Proceedings of the 20th international conference on World wide web (WWW'11). 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)

[22] http://www.tekes.fi/en
[23] http://www.rym.fi/en
[24] http://eit.ictlabs.eu/ict-labs/thematic-action-lines/smart-spaces/

4. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-SPARQL: A Continuous query language for RDF data streams. International Journal of Semantic Computing 04, 3 (2010)
5. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying RDF streams with C-SPARQL. ACM SIGMOD Record 39, 20 (Sep 2010)
6. Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., Lemmens, R.: New generation sensor web enablement. Sensors 11(3), 2652–2699 (2011)
7. Corcho, O., García-Castro, R.: Five challenges for the semantic sensor web. Semantic Web 1(1), 121–125 (2010)
8. Etzion, O., Niblett, P., Luckham, D.: Event Processing in Action. Manning Publications (Jul 2010)
9. Everding, T., Echterhoff, J.: Event Pattern Markup Language (EML). Discussion paper, Open Geospatial Consortium (2008)
10. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1), 17–37 (Sep 1982)
11. Forgy, C.L.: On the efficient implementation of production systems. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1979), aAI7919143
12. Komazec, S., Cerri, D.: Towards Efficient Schema-Enhanced Pattern Matching over RDF Data Streams. In: 10th ISWC. Springer, Bonn, Germany (2011)
13. 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)
14. Lefort, L., Henson, C., Taylor, K.: Semantic Sensor Network XG Final Report (2011), http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/
15. Luckham, D., Schulte, R.: Event processing glossary – version 2.0 (Jul 2011), http://www.complexevents.com/
16. Miranker, D.P.: TREAT: a new and efficient match algorithm for AI production systems. Ph.D. thesis, University of Texas at Austin, New York, NY, USA (1987)
17. Miranker, D.P.: TREAT: A better match algorithm for AI production systems. Tech. rep., University of Texas at Austin, Austin, TX, USA (1987)
18. 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)
19. Silberstein, A., Puggioni, G., Gelfand, A., Munagala, K., Yang, J.: Suppression and failures in sensor networks: A bayesian approach. In: Proceedings of the 33rd international conference on Very large data bases. pp. 842–853. VLDB Endowment (2007)
20. Xu, Y., Helal, S., Thai, M.T., Schmalz, M.: Optimizing Push / Pull Envelopes for Energy-Efficient Cloud-Sensor Systems. In: MSWiM 2011 - 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems. Miami (2011)