Processing Heterogeneous RDF Events with Standing SPARQL Update Rules

Mikko Rinne, Haris Abdullah, Seppo Törmä, and Esko Nuutila

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

Abstract. SPARQL query language is targeted to search datasets encoded in RDF. SPARQL Update adds support of insert and delete operations between graph stores, enabling queries to process data in steps, have persistent memory and communicate with each other. When used in a system supporting incremental evaluation of multiple simultaneously active and collaborating queries SPARQL can define entire event processing networks. The method is demonstrated by an example service, which triggers notifications about the proximity of friends, comparing alternative SPARQL-based approaches. Observed performance in terms of both notification delay and correctness of results far exceed systems based on window repetition without extending standard SPARQL or RDF.

Keywords: Complex Event Processing, RDF, SPARQL Update

1 Introduction

Widening adoption of semantic web technologies has fueled interest to extend the applicability of RDF and SPARQL from static datasets towards processing of dynamic data [1, 2, 4, 9], with potential applications in smart spaces, mobile computing, internet-of-things, and the real-time web.

We have implemented a system for processing incrementally changing RDF data with standing SPARQL queries and Update[1] rules denoted INSTANS[2] [1]. Multiple queries and rules can be active simultaneously, enabling them to work together as a "program". The queries and rules are parsed into a Rete network [5]. INSTANS does not therefore *execute queries* on demand but rather *propagates data* through a query matching network. Each new RDF triple is processed only once through any part of the network, and the output is produced immediately when all the conditions of a registered SPARQL query become satisfied.

In this paper we study the use of RDF/SPARQL in complex event processing – the detection of patterns and derivation of new events from incoming events. Previous attempts of stream processing with RDF/SPARQL have adopted the approach of extending RDF triples with time stamps [7, 11, 12], and SPARQL queries with streams [2, 4], windows [4, 9], or sequences [2].

[1] http://www.w3.org/TR/sparql11-update/
[2] Incremental eNgine for STANding Sparql, http://cse.aalto.fi/instans/

We have tried to find an alternative to such extensions. Our hypothesis is that *incremental query evaluation (as provided by* INSTANS*) using standard RDF/SPARQL can be as good a platform for event processing as non-incrementally evaluated RDF/SPARQL with stream processing extensions.*

In this paper we test the hypothesis with an example application *Close Friends* that produces a "nearby" notification if two friends in a social network come geographically close to each other. For reasons of space and presentation, we have chosen a simple application, although one which can still demonstrate essential aspects of event processing: combination of independently arriving events, filtering based on non-trivial conditions, maintenance of context to avoid repeated notifications, and selection of the recent incoming events.

We present three approaches to Close Friends: (1) *Single SPARQL query* as a reference implementation based on non-incrementally evaluated RDF/SPARQL without any extensions, (2) *Window-based streaming SPARQL* which is repeatedly re-evaluated RDF/SPARQL with stream processing extensions (based on C-SPARQL [4], as described below in more detail), and (3) *Incrementally evaluated RDF/SPARQL* without any extensions, executable with INSTANS.

The solutions are evaluated with event data generated with a mobile user simulator created by INSTANS group. The following evaluation criteria are used:

- *Correctness*: Produce all correct notifications and no incorrect ones.
- *No duplication*: Avoid duplicate notifications of a same event.
- *Timeliness*: How fast are notifications given?
- *Scalability*: How does the system scale with increasing number of events?

The results show that INSTANS outperforms the other solutions. The problem cannot be solved with the single SPARQL query at all and streaming SPARQL is inferior with respect to the criteria above. INSTANS gives corrent notifications without duplicates in a fraction of the notification time of Streaming SPARQL.

While the results do not prove our hypothesis, they nonetheless corroborate it. In the example, which could well be a part of a larger event processing application, incremental query matching turned out to be a more powerful solution enabler than stream processing extensions. The conclusion is that if one wants to do complex event processing with RDF/SPARQL, incremental query matching should be considered before language-level extensions.

Below in Section 2 we first review the previous approaches to event processing with RDF/SPARQL and select the comparison systems. Section 3 presents different solution approaches. In Section 4 we review how successfully each approach can meet the criteria. Finally, Section 5 draws conclusions on the general applicability of the presented methods and discusses topics for future research.

2 Event Processing with RDF and SPARQL

EP-SPARQL described by Anicic et al. [2] is a streaming environment focusing on the detection of RDF triples in a specific temporal order. The examples given on EP-SPARQL also support heterogeneous event formats, create aggregation over sliding windows using subqueries and expressions, and layer events by constructing new streams from the results of queries. EP-SPARQL translation to ETALIS[3] does not currently handle our example queries. Due to the other benefits of using standard-based semantic web technologies we did not investigate the use of the native prolog-based ETALIS further.

C-SPARQL [4], where RDF streams are built based on time-annotated triples, focuses on relatively homogeneous event streams, where each event is represented by a single triple, annotated by time. In this study we consider the suitability of C-SPARQL to process heterogeneous events, and what would be the benefits and challenges of the approach in such an environment.

CQELS also takes the approach of extending SPARQL with window operators on RDF Streams [9]. Like C-SPARQL, it also supports time-based and triple-based windowing. The version of CQELS available at the time of preparing this manuscript does not yet support SPARQL 1.1 Update. Therefore it shares the C-SPARQL limitation of not supporting communication between queries. The distributed version of CQELS is also restricted to use the input files provided by the authors, permitting no comparisons on other example cases to be carried out at this time.

The only stream processing oriented RDF/SPARQL system that is available for comparisons is C-SPARQL. One version of Close Friends application is implemented with it in the next chapter.

A pure performance comparison of INSTANS would be best made against another RDF/SPARQL system using incremental query evaluation. Sparkweave[4] version 1.1 presented by Komazec and Cerri [8] – which is based on Rete-algorithm – is the only one we have discovered so far. Unfortunately, it does not support FILTERs or SPARQL 1.1 features, so it was not possible to use it for comparison of our example.

3 Solution Approaches

The *Close Friends* application can be approached from multiple angles using SPARQL. We concentrate on three evolutionary methods: 1) Single SPARQL 1.1 Query, 2) Window-based streaming SPARQL using C-SPARQL and 3) Incremental query evaluation with SPARQL 1.1 Query + Update using INSTANS. In order to compress our program code, some common formats and fragments of code are collected here. The following prefixes are used:

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX tl:<http://purl.org/NET/c4dm/timeline.owl#>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX event:<http://purl.org/NET/c4dm/event.owl#>
```

The social network is defined by foaf:knows properties:

[3] http://code.google.com/p/etalis/
[4] https://github.com/skomazec/Sparkweave

```
:p2 foaf:knows :p1 . :p1 foaf:knows :p2 .
```

The mobile clients provide location updates in RDF format using Turtle:

```
:e1 a event:Event ;
    event:agent :p3 ;
    event:place [ geo:lat 60.158776 ; geo:long 24.881490 ; ] ;
    event:time  [ a tl:Instant ; tl:at "2012-09-10T08:17:11"^^xsd:dateTime ; ] .
```

The format is flexible and allows clients to add further triples. Some clients could e.g. add altitude reporting.

The following two macros appear repeatedly in the queries:

`<bind events for p1+p2>` means:

```
?event1 event:agent ?person1 ;
        event:place [ geo:lat ?lat1 ;
                geo:long ?long1 ; ] ;
        event:time [ tl:at ?dttm1] .
?event2 event:agent ?person2 ;
        event:place [ geo:lat ?lat2 ;
                geo:long ?long2 ; ] ;
        event:time [ tl:at ?dttm2 ]
```

`<bind event to variables>` means:

```
?event a event:Event ;
       event:agent ?person ;
       event:place [ geo:lat ?lat ;
               geo:long ?long ; ] ;
       event:time [ a tl:Instant ;
                    tl:at ?dttm ; ]
```

The first fragment is used for comparing locations and times (only necessary triples matched), whereas the second fragment is used for cleanup.

3.1 Approach 1: Single SPARQL Query

This example illustrates how the Close Friends use case could be approached using a single SPARQL 1.1 Query without proprietary extensions. It constructs an RDF graph consisting of triples showing which subscribers were found close to each other in the data available in the main graph:

```
CONSTRUCT { ?person1 :nearby ?person2 }
WHERE { # Part 1: Bind event data for pairs of persons who know each other
   GRAPH <http://externalgraphstore.org/socialnetwork> {
      ?person1 foaf:knows ?person2 }
   <bind events for p1+p2>
   # Part 2: Remove events, if a newer event can be found
   FILTER NOT EXISTS {
     ?event3 rdf:type event:Event ;
             event:agent ?person1 ;
             event:time [tl:at ?dttm3] .
     ?event4 rdf:type event:Event ;
             event:agent ?person2 ;
             event:time [tl:at ?dttm4] .
     FILTER ((?dttm1 < ?dttm3) || (?dttm2 < ?dttm4)) }
   # Part 3: Check if the latest registrations were close in space and time
   FILTER ( (abs(?lat2-?lat1)<0.01) && (abs(?long2-?long1)<0.01)  &&
      (abs(hours(?dttm2)*60+minutes(?dttm2)-hours(?dttm1)*60-minutes(?dttm1))<10))}
```

The query produces correct detections within the limits of the simplified filters. Duplicate matches are removed by filtering out older location events of participating persons. We can also confirm that both registrations were within a set interval of each other.

In order to produce results from a stream the query needs to be executed repeatedly. If events keep accumulating into the main graph used for input, the query will quickly slow down. Clearly all the requirements have not been met. Most of the questions, however, can be answered by C-SPARQL in approach 2.

3.2 Approach 2: Window-Based Streaming SPARQL

Out of the tools for window-based streaming SPARQL C-SPARQL [4] was chosen for closer inspection due to the availability of a package for testing. Our Close Friends use case is approached with the following query:

```
REGISTER QUERY CloseFriends COMPUTED EVERY 2m AS
SELECT ?person1 ?person2
FROM STREAM <http://myexample.org/personlocationupdates> [RANGE 10m STEP 2m]
FROM <http://streams.org/socialnetwork.rdf>
WHERE { # Part 1: Bind event data for all friends
   ?person1 foaf:knows ?person2
   <bind events for p1+p2>
   FILTER ( ( ((?lat2-?lat1)*(?lat2-?lat1)) < 0.01*0.01) )
   FILTER ( ( ((?long2-?long1)*(?long2-?long1)) < 0.01*0.01) ) }
ORDER BY ?dttm1 ?dttm2
```

[Remarks: Property paths are currently not supported in C-SPARQL, so the paths in p1+p2 need to be expanded. With no support for an Abs()-function latitudes and longitudes are raised to the power of two.]

The C-SPARQL environment takes care of executing the query at set intervals - in this case every two minutes to keep detection interval reasonably short. Windowing is also provided, set here to ten minutes with a two-minute step size. Removal of old location events is handled by the C-SPARQL windowing system, mitigating any performance penalty due to accumulation of obsolete events in the main graph. If old location events are requested to be archived, a separate method is needed for saving those events.

Some challenges remain. The shorter-than-window-length step size will create frequent duplicates from the same events, forcing duplicate detection and removal to be handled outside this query and the C-SPARQL environment. Ordering the results by the dateTime fields helps to detect the first matching events of a new pair of persons matching the nearby-criteria. Only one location event from every subscriber within one window is needed, but since our event format and reporting interval are flexible, we have to define long enough time-based windows so that at least one report per subscriber fits in with reasonable probability. Other subscribers may submit multiple locations per window, slowing down processing and creating additional duplicates, which need to be filtered. Notifications can only be emitted at execution intervals, in this case every two minutes, leading to increased average and maximum notification delay. Even if no subscribers are active, the query is still executed at the regulated intervals, causing unnecessary processing overhead.

3.3 Approach 3: Incremental Query Evaluation Using INSTANS

Our final approach is based on having a set of collaborating queries processed in parallel. Instead of periodic execution of queries all incoming event triples propagate in a Rete network [5] immediately upon arrival. Two properties of Rete make it especially suitable for incremental query evaluation:

State Saving: Computing of partial results for later use. Eliminates the need to repeatedly process a query from scratch due to new events or time triggers.

Sharing: Network nodes are shared whenever possible. Removes the overhead of computing results which are already available in the network.

Figure 1 shows an example SPARQL query translated into a Rete-network and processed. The query extracts some characteristics of Query 2 below to list events occurring between 10 and 11 am.

The original Rete algorithm performs poorly in case an event has to be deleted from the network. The effects can be minimized by keeping track of an event and its partial matches in the Rete network. This does not add significant processing overhead since the information can be added simultaneously along with the natural flow of events in the network.

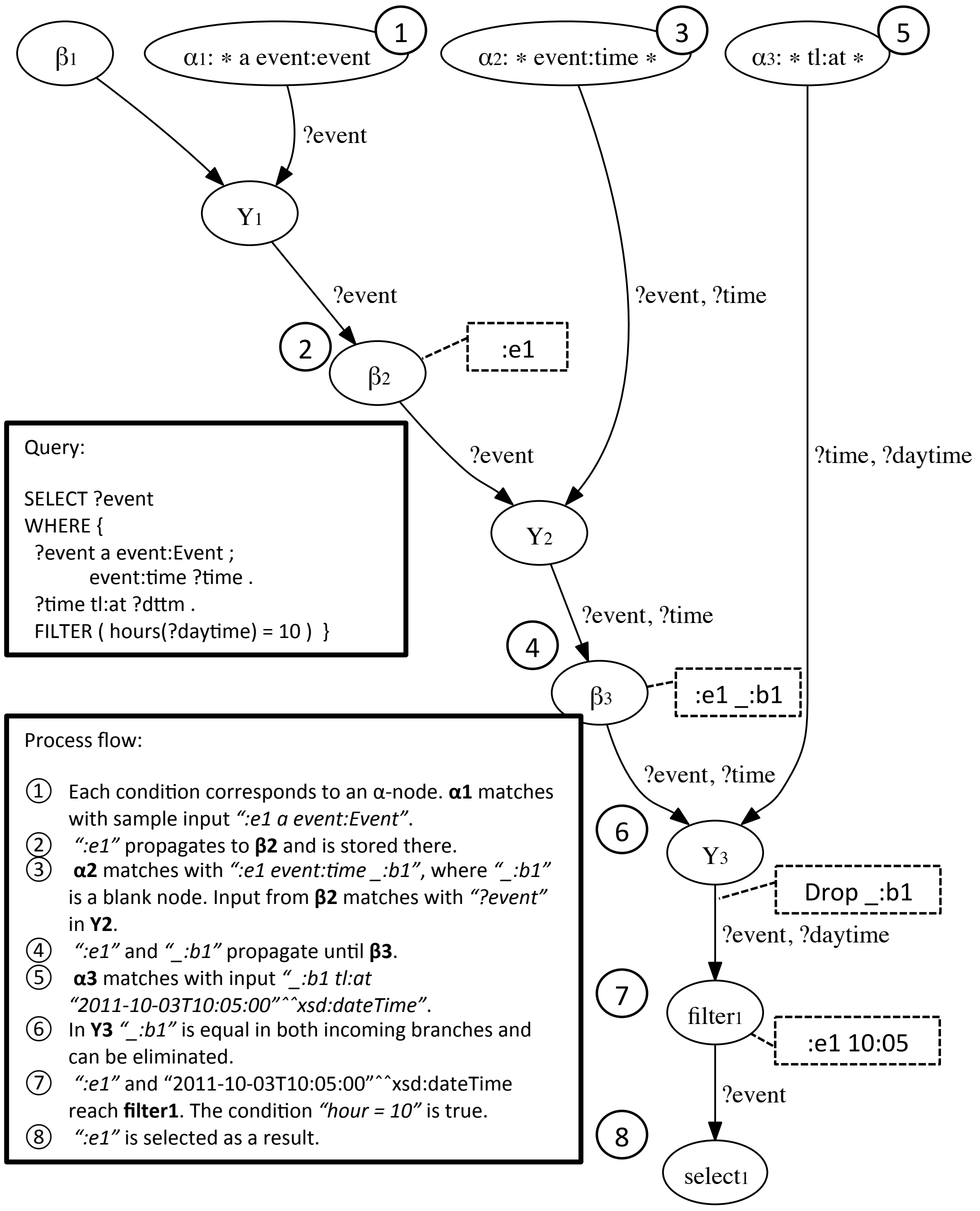
Query 1) Window-query: Creates a window by maintaining the latest location registration from each participant in the workspace, deleting all older ones.

```
DELETE { <bind event to variables> }
WHERE { <bind event to variables>
    FILTER EXISTS {
         ?event2 event:agent ?person ;
                event:time [tl:at ?dttm2] .
         FILTER (?dttm < ?dttm2) } }
```

Query 2) Nearby detection: Inserts a "?person1 :nearby ?person2" detection marker into the workspace when conditions match and only if such a marker did not exist already.

```
INSERT { ?person1 :nearby ?person2 }
WHERE { ?person1 foaf:knows ?person2 .
    <bind events for p1+p2>
    # Check proximity in space and time
    FILTER ((abs(?lat2-?lat1)<0.01) && (abs(?long2-?long1)<0.01) &&
        (abs(hours(?dttm2)*60+minutes(?dttm2)-hours(?dttm1)*60-minutes(?dttm1))<10))
    # Don't do anything, if the relation already exists
    FILTER NOT EXISTS { ?person1 :nearby ?person2 } }
```

Unlike in C-SPARQL, the time filter for expired location events is necessary, because there is no other mechanism to delete obsolete location events from the graph. Another way to do this would be to add one more query to delete old events, when a subscriber has stopped updating.



Query:

```
SELECT ?event
WHERE {
  ?event a event:Event ;
          event:time ?time .
  ?time tl:at ?dttm .
  FILTER ( hours(?daytime) = 10 )  }
```

Process flow:

① Each condition corresponds to an α-node. **α1** matches with sample input *":e1 a event:Event"*.
② *":e1"* propagates to **β2** and is stored there.
③ **α2** matches with *":e1 event:time _:b1"*, where *"_:b1"* is a blank node. Input from **β2** matches with *"?event"* in **Y2**.
④ *":e1"* and *"_:b1"* propagate until **β3**.
⑤ **α3** matches with input *"_:b1 tl:at "2011-10-03T10:05:00"^^xsd:dateTime"*.
⑥ In **Y3** *"_:b1"* is equal in both incoming branches and can be eliminated.
⑦ *":e1"* and "2011-10-03T10:05:00"^^xsd:dateTime reach **filter1**. The condition *"hour = 10"* is true.
⑧ *":e1"* is selected as a result.

Fig. 1: The Rete-network generated from an example query in Instans

Query 3) Notification: Looks for "?person1 :nearby ?person2" triples in the workspace and emits notifications.

```
SELECT ?person1 ?person2
WHERE  { ?person1 :nearby ?person2 }
```

Query 4) Removal of "nearby" status: Deletes a corresponding "?person1 :nearby ?person2" triple from the workspace, if one existed and if ?person1 and ?person2 register sufficiently far from each other.

```
DELETE { ?person1 :nearby ?person2 }
WHERE { ?person1 foaf:knows ?person2 .
        <bind events for p1+p2>
        FILTER ( (abs(?lat2-?lat1)>0.02) || (abs(?long2-?long1)>0.02))
        FILTER EXISTS { ?person1 :nearby ?person2 } }
```

4 Comparison of Approaches

Differences of the three approaches presented in Section 3 based on the criteria defined in Section 1 are summarized in Table 1.

	One query	**C-SPARQL**	**INSTANS**
Correctness of notifications	yes	yes if windows overlap	yes
Duplication elimination	only within one query	only inside window	yes
Timeliness of notifications	query triggered	periodically triggered	event triggered
Scalability wrt #events	no	yes	yes

Table 1: Comparison of One SPARQL Query, C-SPARQL and INSTANS

Performance of INSTANS in terms of notification delay was compared to C-SPARQL[5]. Notification delay was defined as the "time from the availability of events triggering a nearby condition to the time the condition is detected by the system."

In INSTANS queries are processed immediately when the data becomes available, yielding average processing delays of 12 ms on a 2.26 GHz Intel Core 2 Duo Mac. In C-SPARQL average query processing delay varied between 12 - 253 ms for window sizes of 5-60 events, respectively, causing the window repetition rate to be a dominant component of the notification delay for any window repetition

[5] Version 0.7.3, http://streamreasoning.org/download

rate longer than a second. Using window repetition rates of 5-60 seconds with 1 event per second inter-arrival time C-SPARQL notification delay was measured at 1.34-25.90 seconds. Further details on the simulations are available on the INSTANS project website.

5 Discussion and Conclusions

Our goal was to study the applicability of RDF and SPARQL for complex event processing. We were looking for alternatives to language-level extensions to RDF or SPARQL, such as time-stamps, streams, windows, or sequences used in stream processing oriented RDF/SPARQL systems.

Our hypothesis was that incremental query evaluation – as provided by the INSTANS system – using standard RDF/SPARQL can be as good a platform for event processing as non-incrementally evaluated RDF/SPARQL with stream processing extensions. The hypothesis was tested with an example application Close Friends from mobile/social computing domain.

The comparison against C-SPARQL showed that Close Friends application could be better implemented in standard RDF/SPARQL using INSTANS. INSTANS gives corrent notifications without duplicates in a fraction of the notification time of C-SPARQL. All functionality of INSTANS has been achieved using SPARQL 1.1 Query and SPARQL 1.1 Update without any non-standard extensions, maintaining compatibility of all building blocks with existing tools.

While the Close Friends example was designed to be simple to make the analysis self-contained, it exhibits many aspects of complex event processing: combination of independently arriving events, filtering based on non-trivial conditions, maintenance of context to avoid repeated notifications, and selection of the most recent incoming events. Even if we cannot say that the results prove our hypothesis, they nonetheless corroborate it. No stream processing extensions are necessary to solve that problem if incremental query evaluation is used. The conclusion is that if one wants to do complex event processing with RDF/SPARQL, incremental query matching should be considered before extensions to the already voluminous specifications of RDF and SPARQL.

We are currently working on defining the principles for creating event processing applications with INSTANS. One important additional mechanism is the capability to create *timed events*, that is, events that occur in the future. They can be used to capture the *absence of expected events* with a timeout mechanism.

The implementation of applications more complex than Close Friends will create challenges in the areas of engineering and performance. Engineering concerns create a need for modularization of the event processing application, which can benefit from the ability of SPARQL 1.1 Update to process data between different RDF graphs. SPARQL Update rules can use local named RDF graphs as temporary storages. By using such named graphs as workspaces, our queries can process data into windows or buffers, collecting and maintaining only relevant pieces of information from an event stream. We can also share information between queries: all queries can be given access to the same RDF graphs. Finally, we can cascade queries: like CONSTRUCT creates an RDF graph, an INSERT can write query output into a graph, which can modify the input of another query. This gives the possibility to layer the detection of groups of events, which is a fundamental capability for complex event processing.

The performance issues faced in larger applications can be tackled with concurrent execution or distribution of Rete networks. The concurrent execution has been studied already when Rete was applied in rule-based systems [6, 3]. Rete network itself is distributable [10], paving the way for a scalable system. The possibility to create an application out of multiple queries simplifies distribution further, by allowing to execute different queries in different network nodes. Different levels of distributability make the approach highly scalable.

References

1. Abdullah, H., Rinne, M., Törmä, S., Nuutila, E.: Efficient matching of SPARQL subscriptions using Rete. In: 27th Annual ACM Symposium on Applied Computing. pp. 372–377. 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. Aref, M., Tayyib, M.: Lana–match algorithm: a parallel version of the rete–match algorithm. Parallel Computing 24(5–6), 763–775 (1998)
4. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for C-SPARQL queries. In: 13th International Conference on Extending Database Technology. p. 441. Lausanne, Switzerland (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. Gupta, A., Forgy, C., Newell, A.: High-speed implementations of rule-based systems. ACM Transactions on Computer Systems (TOCS) 7(2), 119–146 (1989)
7. Gutierrez, C., Hurtado, C., Vaisman, R.: Temporal RDF. In European Conference on The Semantic Web (ECSW 2005) pp. 93—107 (2005)
8. Komazec, S., Cerri, D.: Towards Efficient Schema-Enhanced Pattern Matching over RDF Data Streams. In: 10th ISWC. Springer, Bonn, Germany (2011)
9. 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 (Oct 2011)
10. Li, G., Jacobsen, H.: Composite subscriptions in content-based publish/subscribe systems. Middleware 2005 pp. 249–269 (2005)
11. Lopes, N., Zimmermann, A., Hogan, A., Lukácsy, G., Polleres, A., Straccia, U., Decker, S.: RDF needs annotations. In: W3C Workshop - RDF Next Steps (Jun 2010)
12. Tappolet, J., Bernstein, A.: Applied temporal RDF: efficient temporal querying of RDF data with SPARQL. pp. 308–322. ESWC 2009 Heraklion, Springer-Verlag (2009)