

# Integrating Semantic Knowledge in Data Stream Processing

Simon Beckstein, Ralf Bruns, Jürgen Dunkel, Leonard Renners

University of Applied Sciences and Arts Hannover, Germany  
Email: forname.surname@hs-hannover.de

**Abstract.** Complex Event Processing (CEP) has been established as a well-suited software technology for processing high-frequent data streams. However, intelligent stream based systems must integrate stream data with semantical background knowledge. In this work, we investigate different approaches on integrating stream data and semantic domain knowledge. In particular, we discuss from a software engineering perspective two different architectures: an approach adding an ontology access mechanism to a common Continuous Query Language (CQL) is compared with C-SPARQL, a streaming extension of the RDF query language SPARQL.

## 1 Introduction

Nowadays, much information is provided in form of data streams: sensors, software components and other sources are continuously producing fine-grained data that can be considered as streams of data. Examples of application fields exploiting data streams are traffic management, smart buildings, health monitoring, or financial trading. Intelligent decision support systems analyze stream data in real-time to diagnose the actual state of a system allowing adequate reactions on critical situations.

In recent years, Complex Event Processing (CEP) [10] has been established as a well-suited software technology for dealing with high frequent data streams. In CEP each data item in a stream is considered as an *event*. CEP uses Continuous Query Languages (CQL) to describe patterns in event streams, which define meaningful situations in the application domain.

However, for *understanding* the business meaning of stream data, the data items must be enriched with semantical background knowledge. For instance in traffic management, velocity measures must be related to specific knowledge about the road network (e.g. road topology and speed limits). In contrast to data streams, this background or domain knowledge is usually rather static and stable, i.e. without frequent changes.

Ontologies defined by Description Logic (DL) [8] provide a well-known formalism for knowledge representation, that can also be used for describing background knowledge. DL distinguishes two different aspects: (1) the TBox contains terminological or domain concepts, and (2) the ABox defines assertional

knowledge or individuals of the concepts that are defined in the TBox. Common languages for describing semantic knowledge are the Resource Description Framework (RDF) for the TBox and the Ontology Language OWL<sup>1</sup> for the ABox. SPARQL [11] provides a standard query language for retrieving knowledge represented in form of RDF data.

Note that SPARQL was originally developed to process static data and is therefore not suitable for the processing of data streams. Otherwise, conventional CEP languages provide no inherent concepts for accessing ontological knowledge.

In this work, we will investigate different approaches on how to integrate data stream processing and background knowledge bases. In particular, we will discuss two different aspects from a software engineering perspective:

- How can CQL languages provided by standard CEP systems make use of ontology models?
- How useful are recently proposed streaming extensions of SPARQL such as C-SPARQL?

The remainder of the paper is organized as follows. The next section discusses related work and other research approaches. Subsequently, section 3 introduces briefly CEP. Then, section 4 discusses the different types of information that can be exploited in stream based systems. The following sections 5 and 6 describe and compare two different approaches of integrating background knowledge into stream processing: The first approach adds an ontology access mechanism to a common CQL-based architecture. The second one uses C-SPARQL, a streaming extension of SPARQL. The final section 7 provides some concluding remarks and proposes an outlook for further lines on research.

## 2 Related Work

In practice, nearly all stream processing systems are using a proprietary Continuous Query Language (CQL). At present, many mature implementations of event processing engines already exist. Some well-known representatives are ES-PER<sup>2</sup>, JBoss Drools Fusion<sup>3</sup> or Oracle CEP<sup>4</sup>. As already discussed, none of these engines neither target nor support a built-in way to integrate semantic background knowledge.

Another class of approaches target the integration of RDF ontologies with stream processing. Different SPARQL enhancements have been developed in order to query continuous RDF streams. Basically, they all extend SPARQL by sliding windows for RDF stream processing:

- *C-SPARQL* provides an execution framework using existing data management systems and triple stores. Rules distinguish a dynamic and a static part, which are evaluated by a CQL and a SPARQL engine, respectively [5, 4].

<sup>1</sup> <http://www.w3.org/TR/2012/REC-owl2-primer-20121211/>

<sup>2</sup> <http://esper.codehaus.org/>

<sup>3</sup> <http://jboss.org/drools/drools-fusion.html>

<sup>4</sup> <http://oracle.com/technetwork/middleware/complex-event-processing>

- *Streaming-SPARQL* simply extends a SPARQL engine to support window operators [6].
- *EP-SPARQL* is used with ETALIS, a Prolog based rule engine. The knowledge (in form of RDF) is transformed into logic facts and the rules are translated into Prolog rules [1, 2].
- *CQELS* introduces a so called white-box approach, providing native processing of static data and streams by using window operators and a triple-based data model [9].

Beside SPARQL extensions, various proprietary CEP languages have been proposed for integrating stream processing and ontological knowledge: For instance, Teymourian et. al. present ideas on integrating background knowledge for their existing rule language Prova<sup>5</sup> (with a corresponding event processing engine) [13, 14].

In summary, many proposals for SPARQL dialects or even new languages have been published, but so far not many results of practical experiments have been proposed.

This paper examines two different approaches for integrating RDF and stream data from a software engineering perspective. First, we extend the well-known CQL of ESPER with mechanisms for accessing RDF ontologies. Then, this approach is compared with C-SPARQL, one of the SPARQL extensions that integrates SPARQL queries and stream processing.

### 3 Complex Event Processing - Introduction

Complex Event Processing (CEP) is a software architectural approach for processing continuous streams of high volumes of events in real-time [10]. Everything that happens can be considered as an *event*. A corresponding *event object* carries general metadata (event ID, timestamp) and event-specific information, e.g. a sensor ID and some measured data. Note that single events have no special meaning, but must be correlated with other events to derive some understanding of what is happening in a system. CEP analyses continuous streams of incoming events in order to identify the presence of complex sequences of events, so called *event patterns*.

A *pattern match* signifies a meaningful state of the environment and causes either creating a new *complex event* or triggering an appropriate action.

Fundamental concepts of CEP are an *event processing language* (EPL), to express *event processing rules* consisting of *event patterns* and *actions*, as well as an *event processing engine* that continuously analyses event streams and executes the matching rules. Complex event processing and event-driven systems generally have the following basic characteristics:

---

<sup>5</sup> <https://prova.ws/>

- *Continuous in-memory processing*: CEP is designed to handle a consecutive input stream of events and in-memory processing enables real-time operations.
- *Correlating Data*: It enables the combination of different event types from heterogenous sources. Event processing rules transform fine-grained simple events into complex (business) events that represent a significant meaning for the application domain.
- *Temporal Operators*: Within event stream processing, timer functionalities as well as sliding time windows can be used to define event patterns representing temporal relationships.

## 4 Knowledge Base

In most application domains, different kinds of knowledge and information can be distinguished. In the following, the different types of knowledge are introduced by means of a smart building scenario:<sup>6</sup> An energy management system that uses simple sensors and exploits the background knowledge about the building, environment and sensor placement.

The main concepts used in the knowledge base are *rooms* and *equipment*, such as *doors* and *windows* of the rooms. Rooms and equipment can be attached with certain *sensors* measuring the *temperature*, *motion* in a room or the *state* of a door or a window, respectively. By making use of this background information, the raw sensor data can be enriched and interpreted in a meaningful manner. For instance, room occupancies due to rescheduled *lectures* or ad-hoc meetings can be identified for achieving a situation-aware energy management. In this sample scenario, we can identify three types of knowledge classified according to their different change frequencies:

1. ***Static knowledge***: We define static knowledge as the knowledge about the static characteristics of a domain, that almost never or very infrequently changes. A typical example in our scenario is the structure of a building and the sensor installation.

Static knowledge can be modeled by common knowledge representation formalisms such as ontologies. Because this information does usually not change, appropriate reasoners can derive implicit knowledge before the start of the stream processing. OWL can serve as a suitable knowledge representation language that is supported by various reasoners, for example KAON2<sup>7</sup> or FaCT++<sup>8</sup>.

2. ***Semi-dynamic knowledge***: We consider semi-dynamic knowledge as the knowledge about the expected dynamic behavior of a system. It can be represented by static knowledge models, e.g. ontologies, as well. In our scenario, a class schedule predicts the dynamic behavior of the building: though the

<sup>6</sup> More details about the smart building scenario can be found in [12].

<sup>7</sup> <http://kaon2.semanticweb.org/>

<sup>8</sup> <http://owl.man.ac.uk/factplusplus/>

class schedule can be defined by static data (e.g facts in an ontology), it causes dynamic events, e.g. each monday at 8:00 a 'lecture start' event. Of course, real-time data produced by sensor could outperform the predicted behavior, e.g. if a reserved class room is not used.

3. **High-dynamic knowledge:** The third type of knowledge is caused by unforeseeable incidents in the real world. It expresses the current state of the real world and cannot be represented by a static ontology. Instead the current state has to be derived from continuous stream of incoming data. This type of knowledge can be described by an event model specifying the types of valid events.<sup>9</sup> Examples in our scenario are sensor events representing observations in the physical world, e.g. motion, temperature, or the state of a window or door, respectively.

The three knowledge types introduced above provide only a basic classification scheme. As already discussed in the introduction (section 1), various types of information must be integrated and correlated in order to derive complex events that provide insight to the current state of a system.

## 5 Using Semantic Knowledge in Event Processing

In this section, we will investigate how the different types of knowledge introduced above can be integrated in stream processing – in particular, how ontological knowledge can be exploited in stream processing.

We start our discussion with a small part of an ontology for our energy management scenario (see Figure 1). This sample ontology is used in the following paragraphs for discussing the different knowledge integration approaches. The model defines the three concepts 'room', 'sensor' and 'equipment' and their relationships. It shows that each room can contain sensors and equipment. Furthermore, it specifies that a certain sensor is either directly located in a certain room or attached to an equipment located in a room.

Note that the location of a sensor can be inferred from the location of the equipment it is attached to. The dashed line describes this implicit property, which can be expressed as role composition in Description Logic:

$isAttacedTo \circ IsEquippedIn \sqsubseteq hasLocation$ . A DL role composition can be considered as a rule: If a sensor is attached to an equipment and the equipment is equipped in a certain room, then the sensor is assumed to be located in the same room.

Listing 1.1 defines two individuals (*Window362* and an attached contact sensor *C362W*) using the RDF turtle notation<sup>10</sup>. Using the above presented DL rule, it can be inferred that the contact sensor is located in room 362 and the triple (`:C362W :hasLocation :Room362`) can be added to the knowledge base.

In the same way, further role and concept characteristics of the ontology can be used for reasoning purposes.

<sup>9</sup> Note that such an event model can also be formally defined by an OWL ontology.

<sup>10</sup> <http://www.w3.org/TR/turtle/>

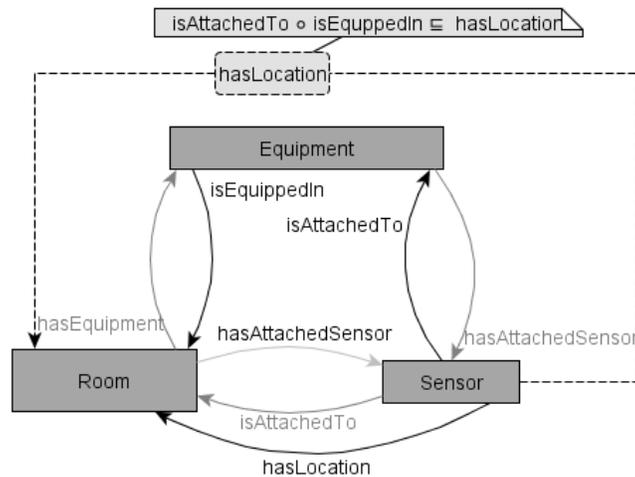


Fig. 1. OWL ontology relationship

```

:Window362
  rdf:type :Window ,
  :isEquippedIn :Room362 .

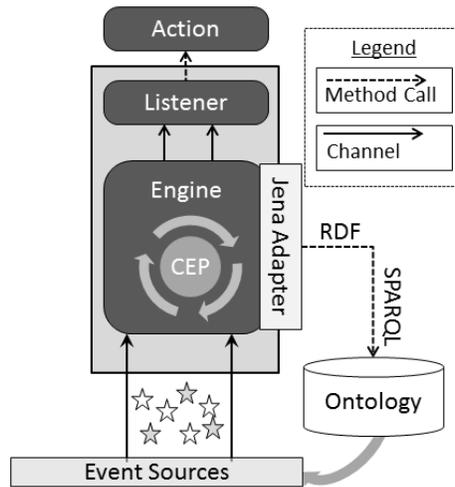
:C362W
  rdf:type :ContactSensor ,
  :isAttachedTo :Window362 .
  
```

Listing 1.1. Some sample entries of the domain knowledge

### 5.1 ESPER

As a first approach of integrating stream data and background knowledge we have chosen the established event processing engine ESPER. Since it is a regular CQL based engine it does not natively support the access of additional knowledge bases. Figure 2 depicts the conceptual architecture of the approach. Different event sources send streams of events via message channels to the ESPER CEP engine. The event sources provide all events in a format that is processable by ESPER, for instance simple Java objects (POJOS). The cycle within the engine should denote that the events are processed in several stages. Each stage transforms relatively simple incoming events into more complex and meaningful events.

**Knowledge Access:** As already mentioned, ESPER does not inherently support a specific access to a knowledge base such as an OWL ontology, but it provides a very general extension mechanism that allows invoking static Java methods within an ESPER rule. Such methods can be used for querying a Java domain model, a database or any other data source. To make our OWL domain



**Fig. 2.** Architecture using ESPER as CEP component

model accessible from ESPER rules, we implemented an adapter class that uses the Jena Framework<sup>11</sup> to query the ontology via SPARQL.

**Events:** Because ESPER can only process Java objects, the adapter has to map RDF triples to Java objects. For instance, the mapping transforms an RDF-URI identifying a sensor to an ID in the Java object. Each Java class corresponds with a certain concept of the ontology TBox.

**Queries:** ESPER provides its own event processing language that is called *ESPER Event Query Language (EQL)*. EQL extends SQL with temporal operators and sliding windows. A simple example is given in Listing 1.2 that shows how motion in a certain room is detected by an ESPER query.

```
SELECT room
FROM pattern[every mse=MotionSensorEvent],
      method:Adapter.getObject(mse.sensorID)
AS room
```

**Listing 1.2.** A sample ESPER query

Actions triggered by a pattern match are implemented in a listener class that must be registered for an ESPER rule. A listener can call any event handling Java method or create a new complex event. The example rule looks rather simple, because the access to the knowledge base is hidden behind the method call (here: `Adapter.getObject(mse.sensorID)`). In our case, the adapter executes a SPARQL query using the Jena framework as shown in Listing 1.3.

<sup>11</sup> <http://jena.apache.org>

```

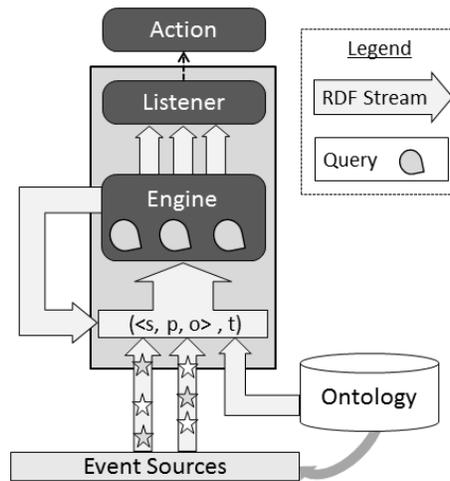
PREFIX    : <http://eda.inform.fh-hannover.de/sesame.owl>
PREFIX    rdf: <http://www.w3.org/1999/02/
           ↵/22-rdf-syntax-ns#>
SELECT ?room ?object
WHERE { :"+sensorID+" :isAttachedTo ?object ;
           :hasLocation ?room .
}

```

**Listing 1.3.** SPARQL query in the Jena-Adapter method *Adapter.getObject(sensorID)*

## 5.2 C-SPARQL

As an alternative approach, we investigate a software architecture using C-SPARQL<sup>12</sup>, a streaming extension of SPARQL. Figure 3 illustrates the main building blocks of the architecture. The main difference to the previous approach is, that all event sources produce a continuous stream of RDF data. This means that the entire knowledge base of the system uses RDF as uniform description formalism.



**Fig. 3.** Architecture using C-SPARQL as CEP component

**Knowledge access:** In this approach, C-SPARQL queries are used for accessing the homogeneous RDF knowledge base. A single C-SPARQL query can combine incoming RDF streams with static background knowledge (also represented in RDF).

<sup>12</sup> We used the 'ReadyToGoPack', an experimental implementation of the concept in [4, 5], available on <http://streamreasoning.org>

**Events:** The events themselves arrive as continuous streams of RDF triples. To allow stream processing with RDF triples, they must be extended with a timestamp. Thus, each event can be described by a quadruple of the following form:

$$((subj_i, pred_i, obj_i), t_i)$$

The *subject* is a unique event identifier, the *predicate* and *object* describe event properties. The *timestamp* is added by the engine and describes the point of time the event arrived. Listing 1.4 shows a set of RDF triples describing a simplified temperature sensor event.

```

: event123 rdf:type                : 2
      ↪ TemperatureSensorEvent
: event123   : hasSensorId         : 3432
: event123   : hasValue            24.7^^xsd:double

```

**Listing 1.4.** A sample temperature event

**Queries:** C-SPARQL queries are syntactically similar to SPARQL. Listing 1.5 shows a C-SPARQL query expressing the same pattern as the ESPER query in Listing 1.2. In contrast to SPARQL, it provides language extensions for temporal language constructs like (sliding) time and batch windows as shown at the end of the FROM STREAM-clause. The FROM-clause selects the various data streams

```

SELECT ?room
FROM STREAM <http://eda.inform.fh-hannover.de/
      ↪ MotionSensorEvent.trdf>
      ↪ [RANGE 10s STEP 1s]
FROM <http://eda.inform.fh-hannover.de
      ↪ /sesame.owl>
WHERE {
  ?mEvent rdf:type                :MotionSensorEvent ;
           :hasSensorID ?sid .
  ?sid    :hasLocation ?room .
}

```

**Listing 1.5.** A sample C-SPARQL query

that are processed in the query. Each C-SPARQL query can either generate new triples that can be processed by another query or call a (Java) listener class to trigger an action.

An interesting point to mention is that the C-SPARQL engine internally transforms the query into a dynamic part dealing with the event stream processing and a static part accessing the background knowledge. These parts are each individually executed by a suitable engine or query processor. This behav-

ior is transparent for the user as the entire rule is written in C-SPARQL and the rule result contains the combined execution outcome.

## 6 Comparison

In this section, we will investigate the capabilities of two introduced approaches of integrating stream processing and background knowledge. Based on our practical experiences, we discuss the two architectures from a software engineering perspective. Table 1 summarizes the results of the comparison. The criteria will be discussed in more details in the following paragraphs.

**Table 1.** Comparison of CQL (ESPER) and C-SPARQL

	ESPER	C-SPARQL
Maturity	+	-
Event Pattern Expressiveness	+	o
Conceptual Coherence	-	+
Dynamic Rules	o	+
Heterogeneous knowledge sources	o	-
Stream Reasoning Support	-	o

**Maturity:** ESPER is a widely used event processing engine, which is under development by an active open source community for many years and, consequently, has reached a stable and market-ready state. It provides a comprehensive documentation and several guides, as well as tutorials. In contrast, C-SPARQL, and the ready-to-go-pack in particular, is a conceptual prototype. This means that the implementation is not as mature and, furthermore, it is not as good documented as ESPER. So far, there are no published experiences about real-world projects using C-SPARQL.

**Event Pattern Expressiveness:** According to its maturity, ESPER provides a rich set of operators for specifying event patterns, e.g. for defining different types of sliding windows or various even aggregations operators. The event algebra of C-SPARQL is less expressive compared to ESPER, but, nevertheless, it supports all important features for general event processing tasks.

**Conceptual Coherence:** C-SPARQL allows the processing of stream data and the integration of static background knowledge by using only one paradigm (or language). Listing 1.5 shows a C-SPARQL query that combines event stream processing and SPARQL queries. In this sense, a C-SPARQL query is self-contained and coherent: only C-SPARQL skills are necessary for understanding it.

In contrast, ESPER does not support inherent access to knowledge bases. Consequently, specialized Java/Jena code must be written to integrate background data. The ESPER-based architecture combines the ESPER query language (EQL) for stream processing and Jena/SPARQL code implemented in a Java adapter class to query knowledge bases. The ESPER rules are not self-contained and delegate program logic to the adapter classes. Note that this can

also be viewed as an advantage: hiding a (perhaps) big part of the logic in method calls results in simpler and easier understandable rules.

**Dynamic Rules:** Changing a rule at runtime is difficult in ESPER, because modifying an ESPER rule can cause a change of the EQL pattern *and* of the SPARQL query in the listener class of the corresponding rule. In this case, the code must be recompiled. C-SPARQL makes changes much easier, because only the C-SPARQL query must be adjusted. Such queries are usually stored as strings in a separate file, which can be reloaded at runtime - even for rules including completely new queries of the knowledge base.

**Heterogeneous knowledge sources:** C-SPARQL is limited to ontological background knowledge stored in RDF format. In contrast, ESPER can be extended by arbitrary adapters allowing the usage of different knowledge sources. For instance, beside RDF triple stores also relational databases or NoSQL data sources can be used. However, the access methods have to be implemented and maintained by hand, as mentioned in the previous paragraph.

**Stream Reasoning Support:** Both approaches do not support stream reasoning, i.e. implicit knowledge is not automatically deduced when new events arrive. Conventional reasoners can only deal with static data, but not with high-frequent RDF streams. But, because (static) background knowledge changes infrequently, a conventional reasoning step can be processed, if a new fact in the static knowledge base appears.

Considering the two approaches from a conceptional point of view, C-SPARQL is better suited for inherent reasoning. For instance, SPARQL with RDFS entailment can be achieved by using materialization or query rewriting [7]. These approaches must be extended to stream processing. First discussions about this issue can be found in [15] and [3].

## 7 Conclusion

In this paper, we have discussed two different architectural approaches of integrating event stream processing and background knowledge.

The first architecture uses a CQL processing engine such as ESPER with an adapter class that performs SPARQL queries on a knowledge base. In this approach stream processing and knowledge engineering is conceptually and physically separated.

The second architecture is based on an extension of SPARQL to process RDF data streams. C-SPARQL allows integrated rules that process stream data and query RDF triple stores containing static background knowledge. Thus, C-SPARQL provides a more homogeneous approach, where query logic, event patterns and knowledge base access are combined in one rule and is, therefore, superior from a conceptional point of view.

Otherwise, CQL engines are well-established in real-world projects and at this time, they offer higher maturity and better performance. Therefore, CQL-based systems are (still) superior from a practical point of view.

Generally, the integration of semantic reasoning into stream processing is still an open issue that is not fully supported by any approach yet. Stream reasoning is therefore an important and promising research field to put effort in and has several work in progress, for example the approaches in [3].

## Acknowledgment

This work was supported in part by the European Community (Europäischer Fonds für regionale Entwicklung) under Research Grant EFRE Nr.W2-80115112.

## 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. ACM (2011)
- [2] Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in etalis. *Semantic Web* pp. 397–407 (2012)
- [3] Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. *ESWC* pp. 1–15 (2010)
- [4] 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. pp. 441–452. *EDBT* (2010)
- [5] Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying rdf streams with c-sparql. *SIGMOD Rec.* pp. 20–26 (2010)
- [6] Bolles, A., Grawunder, M., Jacobi, J.: Streaming sparql - extending sparql to process data streams. In: *The Semantic Web: Research and Applications*, pp. 448–462 (2008)
- [7] Glimm, B.: Using sparql with rdfs and owl entailment. In: *Reasoning Web*, pp. 137–201. *Lecture Notes in Computer Science*, Springer Berlin Heidelberg (2011)
- [8] Krötzsch, M., Simancik, F., Horrocks, I.: *A description logic primer*. *CoRR* (2012)
- [9] Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: *The Semantic Web – ISWC 2011*, pp. 370–388 (2011)
- [10] Luckham, D.C.: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley (2002)
- [11] Prud'hommeaux, E., Seaborne, A.: Sparql query language for rdf, <http://www.w3.org/TR/rdf-sparql-query/>
- [12] Renners, L., Bruns, R., Dunkel, J.: Situation-aware energy control by combining simple sensors and complex event processing. In: *Workshop on AI Problems and Approaches for Intelligent Environments*. pp. 29–34 (2012)
- [13] Teymourian, K., Paschke, A.: Enabling knowledge-based complex event processing. In: *Proceedings of the 2010 EDBT/ICDT Workshops*. pp. 37:1–37:7. *ACM* (2010)
- [14] Teymourian, K., Rohde, M., Paschke, A.: Fusion of background knowledge and streams of events. In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. pp. 302–313. *ACM* (2012)
- [15] Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. In: *Journal on Data Semantics II*, pp. 1–34. *Lecture Notes in Computer Science* (2005)