

Semantic Stream Processing in Dynamic Environments Using Dynamic Stream Selection

Michael Jacoby¹ and Till Riedel²

¹ Fraunhofer IOSB, Karlsruhe, Germany
`michael.jacoby@iosb.fraunhofer.de`,

² TECO, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
`riedel@teco.edu`

Abstract. Cyber-physical systems (CPS) require a new level of dynamics in information processing. Databases and query approaches need to be extended towards dynamic stream aggregation and analysis systems. In this paper, we designed ECQELS, a semantic stream processing engine, to support CPS applications by adding essential features like dynamic sensor selection. We present a feature complete first implementation and show competitive performance results.

Keywords: stream processing, semantic streaming, dynamic stream selection, ECQELS

1 Motivation

When control systems, appliances, enterprise software and large scale information processing architectures were still considered different computing domains the system design was a comparably straightforward tasks. A deeper integration of information processing with the real world brought an unknown level of dynamics to all those domains. Examples range from intelligent user-adaptive building automation systems to intelligent products that drive their own production process. Such cyber-physical systems have different common properties:

Dynamic data: systems operate on a continuum of streaming, dynamically updated data from different sources

Linked data: various data from different systems is loosely linked in different and often evolving structures. Common models and semantics are needed to work on data along and across value chains.

Advanced querying, filtering and interlinking of relevant data sources is supported by so-called semantic stream processing engines. As we show in this paper, however, more language features are needed to implement cyber-physical systems, like the ones mentioned before. Namely, this includes

Dynamic sensor selection: In both future building automation and industrial production scenarios, the number and location of sensors and information sources changes over time. It is difficult to support dynamic discovery and interlinking not only of data but also of distributed data sources.

Real-time-based and -triggered windows: The time semantics of processing events often highly differs in embedded and semantic processing. Near real-time processing with clear time semantics is a minimum requirement for CPS.

Modularity: Nested query constructs are needed for modularity, re-usability and auto-generation of queries. The expressiveness of a query language is a precondition for more advanced use cases.

To tackle those aspects, we have designed a system called Extended Continuous Query Evaluation over Linked Streams (ECQELS). While the system conceptually builds upon existing work it has been designed from scratch to support our application scenarios.

Section 2 gives an introduction to semantic stream processing and an overview of related work with focus on semantic stream processing engines. Section 3 covers the design of ECQELS and introduces the concept of *Dynamic Stream Selection*. In Section 4 details on ECQELS, the proof-of-concept implementation of a semantic stream processing engine supporting the dynamic stream selection pattern, is presented. Section 5 shows the evaluation results of the ECQELS engine and Section 6 concludes the paper with an outlook on future work.

2 Background and Related Work

Semantic stream processing is highly related to the concepts of continuous querying and complex event processing (CEP) and they altogether can be summarized with the concept of information flow processing introduced in [5]. In contrast to static systems, queries are not only executed once against an already populated database but rather are evaluated over time as new data enters. A stream of results is then passed to the querying user or another system, allowing complex systems of systems. To be able to express continuous queries special concepts and language constructs of a query language are needed [5]. Many different query languages and engines for stream processing are available [6,4,7,8,9,10,11,12]. Probably the most acknowledged query language is CQL (Continuous Query Language) [6] as it introduces multiple basic concepts that can be found in most other query languages such as Stream-to-Relation operators (windows), Relation-to-Relation operators (derived from traditional query languages such as SQL) and Relation-to-Stream operators (I(nsert)stream, D(elete)stream, R(elational)stream).

Semantic stream processing extends this constructs to interlinked data working on dynamic temporary graphs constructed by streams. In the domain of semantic streaming the most popular query languages/engines are CQELS [4], C-SPARQL [7] and SPARQL_{Stream} [8]. Instead of extending SQL like CQL they are based on SPARQL and extend it by some of the concepts introduced in CQL.

While most languages only have a quite limited support for CEP constructs, CQELS announced a new version, called CQELS-CEP, with special support for Complex Event Processing[13]. The W3C RDF Stream Processing Community Group¹ is also working on a unified query language for RDF stream processing[14]. As outlined above, however, no current streaming processing engine fulfills all the properties we see as minimally needed to support common CPS use cases. Our work explores the implication of introducing those essential features while retaining the current expressiveness.

¹ <https://www.w3.org/community/rsp/>

3 Design of ECQELS

The driving idea begin the design of ECQELS was to develop a semantic streaming engine that is suited for the use in dynamically changing environments like real-world scenarios massively using mobile sensors. To address this issue, ECQELS introduces the *REFRESH* keyword, which enables dynamic adaption of queries at run-time. It can be applied either to parts of the query fetching background knowledge, i.e. *GRAPH* or *SERVICE* patterns, or to *STREAM* patterns using implicit addressing. If applied to patterns fetching background knowledge the background knowledge will be updated periodically and (potentially) new query results will be computed and return. If applied to a stream pattern using implicit addressing it causes the streams taken into account to answer this query to dynamically change during run-time and therefore it is called *Dynamic Stream Selection*.

Listing 1 shows three different ways of stream addressing side-by-side. Listing 1a shows an example query defined in the CQELS language which uses explicit addressing of a stream with the URI <http://example.com/sensors/1> whereas Listing 1b shows an example of implicit stream addressing using CQELS. The downside of implicit stream addressing in CQELS is, that the binding of the variable *?stream* is only evaluated once when the query is registered and can't change during query run-time. Therefore, if over the lifetime of a query a new stream becomes available it will not be considered in the query as the binding of the variable *?stream* will not be updated. ECQELS solves this issue with the introduction of the Dynamic Stream Selection concept. In Listing 1c an example is given where the binding of the variable *?stream* is updated every ten seconds. This means that every ten seconds the streams this query consumes are chosen depending on the current value of the binding of the variable *?stream*.

To a limited extend, this behavior can be emulated using CQELS by periodically executing a first query containing only the pattern to find the streams of interest and then generate a query explicitly addressing all streams, stop and unregister the old version and register and start the new version of the query. The drawback of this approach is that every time the streams addressed change, the query has to be stopped and therefore all intermediate results are completely lost. In a scenario where we want to aggregate information over a longer time window e.g. 1 hour, but have a high frequency of changes in available streams, e.g. every 1 minute, the query won't be able to produce any result as it will never run longer than one minute before being stopped, modified and started again. ECQELS can execute these kind of queries as it does not stop the whole query but only dismisses streams that are no longer present when the binding is updated and thus is able to maintain intermediate results.

4 Implementation of ECQELS

The development of ECQELS was motivated by trying to enable domain-specific rule-based automation with semantic stream technology [3] and started out as an extension to CQELS. Unfortunately, we realized that CQELS has a number of drawbacks that we were not able to overcome such as missing support for nested queries, which was essential, as we wanted to auto-generate queries from models defined in a visual domain-specific programming language. Furthermore, CQELS uses a purely event-based evaluation strategy for time-based windows. This means

```

SELECT ?sensor ?val
WHERE {
  STREAM <http://example.com/sensors/1> [TRIPLES 1]
  { ?sensor :hasValue ?val . }
}

```

(a) CQELS query using explicit stream addressing.

```

SELECT ?sensor ?val
WHERE {
  STREAM ?stream [TRIPLES 1]
  { ?sensor :hasValue ?val . }
  ?stream a :SensorStream .
}

```

(b) CQELS query using implicit stream addressing.

```

SELECT ?sensor ?val
WHERE {
  STREAM ?stream [TRIPLES 1] [REFRESH 10s]
  { ?sensor :hasValue ?val . }
  ?stream a :SensorStream .
}

```

(c) ECQELS query using the Dynamic Stream Selection pattern.

Listing 1: Example queries showing different types of stream addressing.

if you write a query asking for the average value of a stream of the last five minutes every minute you do not get a result every minute, as you would expect but rather a new result every time a new triple arrives on the stream. This property renders it useless for executing most automation tasks. Therefore, we decided to implement ECQELS from scratch but to keep the CQELS language as a query language and add new features to it.

The key features of ECQELS are

- Proof-of-Concept implementation of the Dynamic Stream Selection pattern
- REFRESH keyword available on *STREAM*, *GRAPH* and *SERVICE* pattern
- Full support of SPARQL 1.1 features
- RStream [6] output strategy
- Combination of event & time-based evaluation strategy
- Full life-cycle management for queries
- REST-based server implementation with support for Server-Sent Event (SSE)-based RDF streams

ECQELS is implemented using Java and based on the Apache Jena ARQ framework. It is published as open source on GitHub².

5 Evaluation

5.1 Benchmark Design and Runtime Environment

Evaluating the performance of ECQELS w.r.t. Dynamic Stream Selection is hard as to our knowledge there is no other tool that supports this functionality. Therefore,

² <https://github.com/ecqels>

we decided to evaluate the performance of ECQELS without using the Dynamic Stream Selection feature. We introduce a new benchmark based on the one published together with CQELS [4] taking into account not only performance but also correctness. This is achieved by measuring the feature *average response time*, defined as the time between insertion of a triple on a stream and the reception of the corresponding expected result, instead of the *average query execution time* used in [4], which is from our point of view a feature much more relevant for real-life applications.

The benchmark comprises five queries which are executed against a varying size of background knowledge of 10k, 100k and 1M triples with a new triple being inserted on the stream every 10ms. It was executed on a desktop machine with an Intel i7-2600k, a quad core processor running at 3.40GHz, and 8GB RAM running Windows 8.1. The JVM was started with the arguments `-Xms1024m -Xmx4g`. The settings ensure a minimum heap size of 1GB and a maximum heap size of 4GB respectively.

5.2 Results and Analysis

Figure 1 shows the benchmark results for three representative queries (based on the RFID example introduced by CQELS) comparing the average response time for CQELS and ECQELS with a varying size of background knowledge. For query 1 and 5 the performance of both systems can be considered the same but for query 3 CQELS has a constant run-time whereas the runtime of ECQELS grows linear with the size of the background knowledge. This is due to multiple reasons. First, query 3 contains two triple patterns over the graph <http://deri.org/floorplan/> with a shared variable, which can be seen as a self-join from database perspective and causes a quadratic run-time w.r.t. the size of the background knowledge. CQELS however has a constant run-time because it uses an elaborated evaluation strategy that only computes incremental changes when a new triple arrives on a stream. For query 5 we see that CQELS now also has a linear run-time which is due to the aggregation function used in query 5 preventing CQELS forcing CQELS to not only compute incremental changes. In summary, we can say that ECQELS can compete with the performance of CQELS at least for small size of background knowledge or simple queries.

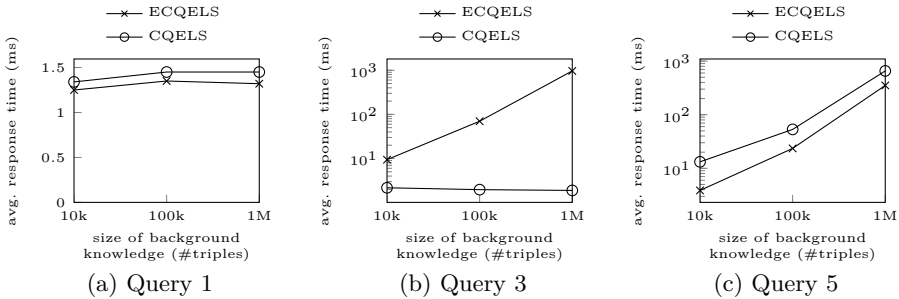


Fig. 1: Results of performance evaluation for a single query.

6 Conclusion & Future Work

In this paper we introduced a new semantic stream processing engine that incorporates concepts like Dynamic Stream Selection and makes the use of semantic stream processing technologies much more feasible in dynamic environments like cyber-physical systems. The evaluation that for most of the scenarios the performance of our ECQELS implementation is comparable to CQELS while providing many new essential features.

More work on the prototype is needed on some features like time-based windows. While providing a more intuitive behaviour than comparable implementations ECQELS has some drawbacks. For example, triple-based windows can still lead to non-deterministic results that might be difficult for a developer to deal with. We, however, have already used our prototype runtime successfully as backend for a visual model-driven development environment with end-users. In future work we hope to proof that together with good development support our language extensions make the system a good candidate for practical CPS development.

In this paper, we focused on presenting the underlying language extensions and their implementation. We were able to show promising performance results already based on a naive implementation. With the public release of our software, we hope to make it valuable system component for semantic data-driven cyber-physical systems.

References

1. Arasu, A., Babu, S., Widom, J. (2006). The CQL continuous query language: Semantic foundations and query execution. *VLDB Journal*, 15(2), 121-142. doi:10.1007/s00778-004-0147-z
2. Prud'Hommeaux, Eric, and Andy Seaborne. SPARQL query language for RDF. W3C recommendation 15 (2008).
3. Jacoby, Michael. (2011). Enabling Domain-Specific Rule-Based Automation With Semantic Stream Technology (Master's Thesis). doi:10.5445/IR/1000056564
4. Danh, L.-P., Minh, D.-T., Parreira, J. X., Hauswirth, M. (2011). A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. *Semantic Web - Iswc 2011, Pt I*, 7031(24761), 370-388.
5. Cugola, G., & Margara, A. (2011). Processing Flows of Information : From Data Stream to Complex Event Processing, V(i), 359-360.
6. Arasu, A., Babu, S., & Widom, J. (2006). The CQL continuous query language: Semantic foundations and query execution. *VLDB Journal*, 15(2), 121-142. doi:10.1007/s00778-004-0147-z
7. Barbieri, D. F. (2009). C-SPARQL : SPARQL for Continuous Querying. *Language*, 427(c), 1061-1062. doi:10.1145/1526709.1526856
8. Calbimonte, J.-P., Oscar, C., & Gray, A. (2010). Ontology-based Access to Streaming Data Sources. 7th Extended Semantic Web Conference ESWC2010, 6496 LNCS(PART 1), 2-3.
9. Mileo, A., Abdelrahman, A., Policarpio, S., & Hauswirth, M. (2013). StreamRule: A nonmonotonic stream reasoning system for the semantic web. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7994 LNCS, 247-252.
10. Rinne, M., Nuutila, E., & Törmä, S. (2012). INSTANS: High-performance event processing with standard RDF and SPARQL. *CEUR Workshop Proceedings*, 914, 101-104. doi:10.1109/ICDE.2013.6544856

11. Anicic, D., & Fodor, P. (2011). EP-SPARQL: a unified language for event processing and stream reasoning. *Proceedings of the 20th international conference on World wide web*, 635-644. doi:10.1147/sj.433.0598
12. Zhou, Q., Simmhan, Y., & Prasanna, V. (2012). SCEPter : Semantic Complex Event Processing over End-to-End Data Flows, (April), 1-20.
13. Dao-Tran, M., & Le-Phuoc, D. (2015). Towards enriching CQELS with Complex Event Processing and path navigation. *CEUR Workshop Proceedings*, 1447, 2-14.
14. Aglio, D. D., Calbimonte, J., Valle, E. Della, & Corcho, O. (n.d.). Towards A Unified Language for RDF Stream Query Processing.

Machine Learning for Cyber Physical Systems
Selected papers from the International Conference
ML4CPS 2016

Beyerer, J.; Niggemann, O.; Kühnert, C. (Eds.)
2017, VII, 72 p. 24 illus., 19 illus. in color., Softcover
ISBN: 978-3-662-53805-0