RDF Stream Processing Prototyping with Streaming MASSIF

Pieter Bonte[0000−0002−8931−8343] Femke Ongenae[0000−0003−2529−5477]

IDLab, Ghent University – imec
Pieter.Bonte@UGent.be

Abstract. Stream Reasoning (SR) is the research field that aims to solve the heterogeneity, noisiness, and incompleteness problems that come with the processing of continuously produced data. The use of Semantic Web technologies allows to mitigate many of these problems. RDF Stream Processing (RSP), a subfield of SR, specifically deals with the processing of RDF data streams. Many RSP engines and solutions exist, however, none provide the flexibility to rapidly create a prototype for new RSP applications or aid in the design of new components. In this demonstration paper, we showcase Streaming MASSIF Prototypes, a RSP prototyping platform that provides the necessary tooling to effortlessly create new RSP applications. Furthermore, the available tooling and the inherent flexibility of the platform allow researchers and developers to fully focus on the design of new RSP components, hiding superfluous complexities and configurations.

Keywords: Stream Reasoning · Prototyping · RDF Stream Processing

1 Introduction

Data is being produced at a volume and velocity that out limits the ability to consume the data as a whole [8]. Therefore, data intensive applications consume data continuously as it is being produced. However, data (streams) typically cope with challenges of heterogeneity, noisiness, and incompleteness, which the Stream Reasoning (SR) research field aims to solve [4]. The use of Semantic Web technologies allows to mitigate many of these problems. RDF Stream Processing (RSP), a subfield of SR, focusses on processing RDF data streams in particular. Many RSP engines and solutions exist [1, 6, 2], however, lacking the tooling for easy prototyping of RSP applications and limiting the development and evaluation of new RSP components. RSP applications are typically composed of multiple parts, e.g. a windowing function to chop the continuous stream in processable chunks, a SPARQL engine for selecting elements from the streams, (expressive) reasoning for inferring implicit and missing facts, and optionally temporal reasoning for detecting time-related dependencies [3]. In this paper, we demonstrate Streaming MASSIF Prototypes, an easy-to-use RSP prototyping platform, built upon the Streaming MASSIF platform [3]. Our prototyping platform offers the tooling to easily compose new RSP applications and provides the flexibility to focus on building new RSP components. We have decomposed various RSP applications and engines in various abstract building blocks. These building blocks can be composed in any arbitrary order, allowing the creation of a flexible RSP processing graph. With Streaming MASSIF Prototypes, RSP applications can easily be created because we offer the necessary tooling to get started and focus on the definition of the business logic. New components can easily be integrated as we provide abstractions, flexible interface, and monitoring functionality out of the box.

2 State of the art

Looking at most RSP engines and applications we can see that they can be decomposed into some abstract building blocks:

C-SPARQL [1] first windows the data stream into processable chunks, optionally allows performing RDFS reasoning to infer missing facts while selecting data from the window using a SPARQL engine.

CQELS [6] integrates the windowing directly into the evaluation of the queries.

CityPulse [7] uses CQELS to select data from the data streams, ASP to reasoning upon the selection, and CEP to enable temporal reasoning.

Streaming MASSIF [3] uses C-SPARQL to select data from the stream, a Description Logic (DL) reasoner to perform expressive reasoning and abstract events and CEP on top of the abstractions.

Many more engines exist, most can be abstracted to the following building blocks:

- Windowing: a function to divide the continuous stream into processable chunks.
- Selection: a selection function a select and filter data according to a registered query.
- Abstraction: a reasoning component to infer implicit and missing facts.
- Temporal: a temporal reasoning component to infer temporal dependencies.

However, none of the existing engines or platforms provide any tools to easily load different data sources, handle the results, monitor the different building blocks, add some flexibility to reorder the building blocks, or add new components. We provide the needed tooling for easy prototyping new RSP applications and components.

3 Stream Reasoning Prototyping

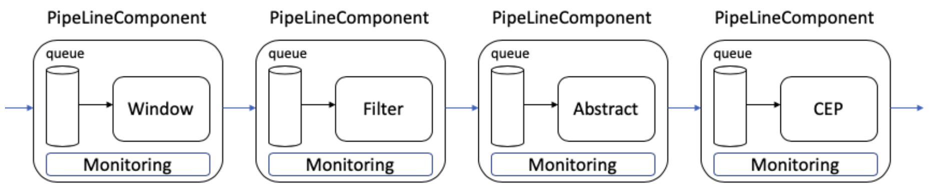


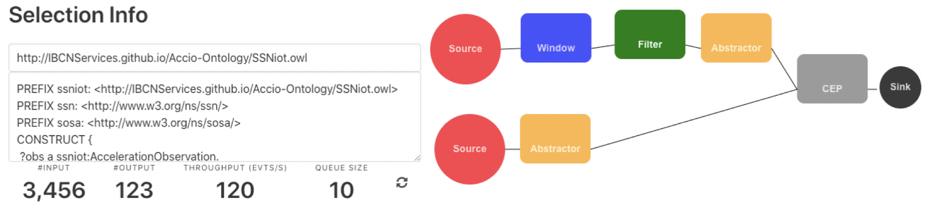
Fig. 1. Visualization of the pipeline architecture.

To enable easy prototyping and integration of new RSP components, Streaming MASSIF Prototypes has been built on top of the following principles:

1. **Flexible interfaces**: in order to integrate new components, or compose building blocks in any order, interfacing between components should be flexible and transparent. Inspired on the flexibility of the Unix pipelines, we chose the use text as interface. This allows to easily plugin new components. Throughout the platform, we support semantic data using the Turtle serialization (or column separated tabular data if data has not been mapped to a semantic model yet).
2. **Abstract components:** all components are being abstracted on the highest level to *PipeLineComponent*s, this allows that upstream or downstream components can be decoupled and are not depending on any specific implementation. To make prototyping easier, we support the following components:
   (a) **Sources**: the data ingestion points that load the data streams into the platform. Support for Kafka, HTTP Post and get (with configurable timeouts), websockets and reading from file (with configurable timeouts).
   (b) **Windows**: function to window the data stream in processable chunks. Support for sliding and tumbling windows. Current implementation: Esper[1].
   (c) **Filters**: allow to select and filter certain parts from the data stream through SPARQL-queries. Current implementation: Apache Jena[2].
   (d) **Abstractors**: reasoning step that allows to infer implicit facts and abstract events to high-level concepts. Current implementations: HermiT [5] for expressive ontology reasoning and C-Sprite [2] for RDFS-reasoning.
   (e) **CEP**: Complex Event Processing (CEP) module that allows detecting temporal patterns. Currently through Esper.
   (f) **Mappers**: Mapping raw data to semantic data.
   (g) **Enrichment**: allows to enrich the data stream with static data.
   (h) **Sinks**: functions as the end-point of the data stream. Support for Kafka, HTTP, WebSockets, writing to file, and print sinks (to terminal or web interface).

   The benefit of these abstractions is that multiple implementations are possible. Note that additional components can easily be integrated. Figure 1 depicts an example pipeline configuration using the abstract components.
3. **Monitoring:** As depicted in Figure 1, each component is wrapped in a *PipeLineComponent* that has its own queue for asynchronous processing of the components, and a monitoring layer to allow easy evaluation of the component's metrics. The metrics can be accessed through an HTTP endpoint and contain the processing times, the number of events in the queue, and the throughput.
4. **Processing Graphs:** Each component has the flexibility to have multiple inputs and outputs, allowing the creation of a processing graph for RSP applications.
5. **Graphical Interface:** To simplify the creation of the processing graph, we provide a drag and drop web interface that allows to compose RSP workflows. Figure 2 visualizes an example composition on the right pane. The left pane shows the registered query of the filter component and its monitoring details. Alternatively, the configuration can be loaded in JSON-LD format.

[1] http://www.espertech.com/
[2] https://jena.apache.org/



http://IBCNServices.github.io/Accio-Ontology/SSNiot.owl

```
PREFIX ssniot: <http://IBCNServices.github.io/Accio-Ontology/SSNiot.owl>
PREFIX ssn: <http://www.w3.org/ns/ssn/>
PREFIX sosa: <http://www.w3.org/ns/sosa/>
CONSTRUCT {
 ?obs a ssniot:AccelerationObservation.
```

#INPUT	#OUTPUT	THROUGHPUT (EVTS/S)	QUEUE SIZE
3,456	123	120	10

Fig. 2. Example of the RSP workflow composer (right) and the information pane (left) detailing the configuration and monitoring details of the filter component.

4 Demonstrator

In this demonstrator, we show how RSP applications, consisting of various components, can be built using the workflow composer. The demonstrator shows how complex applications can be built through drag and drop interaction and declarative definitions of the components. It shows how we can be debug RSP applications by investigating intermediate results of each of the components in the user interface and how each component can be monitored in order to detect bottlenecks. A video of the demonstrator can be found here[3].

Acknowledgments: Pieter Bonte is funded by a postdoctoral fellowship of Fonds Wetenschappelijk Onderzoek Vlaanderen (FWO) (1266521N).

References

1. 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 **4**(01), 3–25 (2010)
2. Bonte, P., Tommasini, R., De Turck, F., Ongenae, F., Valle, E.D.: C-sprite: Efficient hierarchical reasoning for rapid rdf stream processing. In: Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems. pp. 103–114 (2019)
3. Bonte, P., Tommasini, R., Della Valle, E., De Turck, F., Ongenae, F.: Streaming massif: cascading reasoning for efficient processing of iot data streams. Sensors **18**(11), 3832 (2018)
4. Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook. Data Science **1**(1-2), 59–83 (2017)
5. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: an owl 2 reasoner. Journal of Automated Reasoning **53**(3), 245–269 (2014)
6. 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: International Semantic Web Conference. pp. 370–388. Springer (2011)
7. Puiu, D., Barnaghi, P., Tönjes, R., Kümper, D., Ali, M.I., Mileo, A., Parreira, J.X., Fischer, M., Kolozali, S., Farajidavar, N., et al.: Citypulse: Large scale data analytics framework for smart cities. IEEE Access **4**, 1086–1108 (2016)
8. Tommasini, R., Calvaresi, D., Calbimonte, J.P.: Stream reasoning agents: Blue sky ideas track. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1664–1680 (2019)

[3] https://github.com/IBCNServices/StreamingMASSIF/wiki/Streaming-MASSIF-Prototypes