StreamConnect: Ingesting Historic and Real-Time Data into Unified Streaming Architectures

Philipp Zehnder and Dominik Riemer

FZI Research Center for Information Technology
Haid und Neu Str. 10 14
76131 Karlsruhe, Germany 76131 Karlsruhe, Germany,
zehnder@fzi.de,
riemer@fzi.de,
https://www.fzi.de/en/

Abstract. The web of things provides a steadily increasing amount of both real-time and historic data sources. Yet widespread standards are missing and the heterogeneity of data formats and communication protocols makes the integration of such sources a challenging task often requiring for manual programming effort.
This paper presents a novel, lightweight semantics-based approach to quickly connect heterogeneous data sources to stream processing systems. Our main contributions are i) a new model to represent characteristics of data streams and data sets such as schema and quality independent from the actual run-time format, ii) generic data adapters and methods to automatically discover these characteristics at runtime and iii) a distributed architecture to pre-process (e.g. clean and filter) raw data coming from these adapters directly on the edge before data is processed by a stream processing engine.
Our contribution eases the ingestion of batch and real-time data into unified streaming architectures.

Keywords: Stream Processing, Data Ingestion, Semantic Web

1 Introduction

In recent years, emerging trends such as the web of things have led to an enormous data growth. For instance, manufacturing companies more and more gather, besides existing data sources such as master and customer data, also massive amounts of real-time data sources coming directly from shop floors. At the same time, the web benefits from many data sources that are publicly made available by means of open APIs.

One major benefit of this trend is the ability to integrate and process such sources in real-time as a basis for advanced analytic operations, enabling companies to find correlations such as incident patterns early or even ahead of time.

From an information management perspective, architectural patterns such as publish/subscribe systems have gained popularity, enabling enterprises to establish so-called data backbones that collect data in a single, yet distributed messaging system such as Apache Kafka [1]. At the same time, modern distributed

streaming engines such as Apache Flink [2] are able to process both real-time and historical data in a unified streaming architecture. Such architectures, also known as Kappa architecture [3], reduce the effort to deploy and maintain two different code bases for batch processing of historical data and stream processing for quickly computing real-time views required by other Big Data architectures such as the Lambda architecture [3].

However, a still remaining open problem is the accessibility and ingestion of data sources into such architectures for further processing. The development of adapters for individual data sources is still a highly manual task [4] due to the heterogeneity of protocols and diversity of data formats. This usually requires for both technological expertise as well as domain knowledge to understand the meaning of gathered data. The main objective of this paper is to reduce the technical effort for the integration of new data sources into a big data streaming-only infrastructure by introducing a semantics-based adapter concept. This objective poses a number of technical challenges and requirements that need to be considered:

- **Temporal Aspect:** Adapters need to be able to handle both real-time and historical data.
- **Adapter Configuration:** A solution must provide a higher level of abstraction to enable domain experts to configure adapters, which still requires a lot of technical understanding.
- **Data Cleaning:** Since adapters are often long running processes, they should ensure that the quality of the data does not change over time.
- **(Edge) Pre-Processing:** Simple pre-processing steps such as filtering, transforming or aggregating data should be executed locally close to the sensor to avoid sending noisy data to the messaging system.

This paper is structured as follows: In section 2, we present a motivating scenario that illustrates the need and general approach of our contribution. Section 3 introduces an event model we developed for both raw-data and semantically described virtual sensors. After defining the model, section 4 describes the adapter architecture and illustrates how a new adapter can be modeled. Finally, section 5 presents the related work followed by section 6, conclusion and outlook.

2 Motivating Scenario

This section provides an illustrative scenario that shows the challenges that must be solved when integrating multiple heterogeneous data sources with a single adapter concept. As an example we will present how an adapter for a new temperature sensor on an oil rig is created. The events are stored in a message broker, like Apache Kafka, and can then be used by other systems like StreamPipes[1] [5] to build and execute processing pipelines, shown in figure 1.

[1] http://streampipes.fzi.de

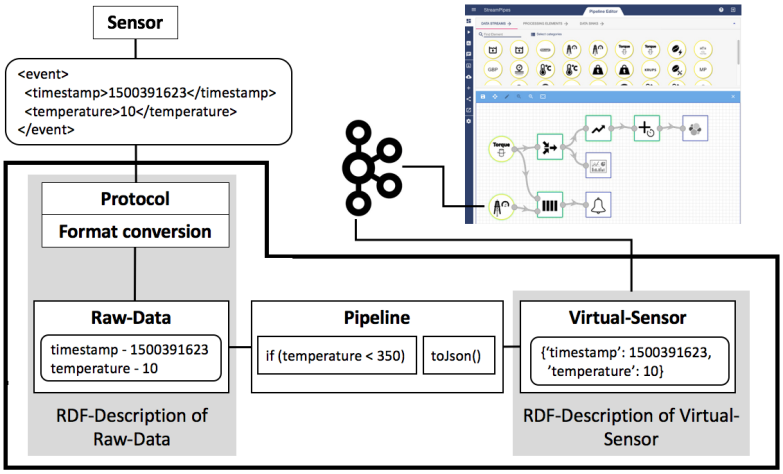

Fig. 1: Example: Functionality of the adapter

To create a new adapter, the user makes use of a model editor, which realizes a guided process where the user has to provide mandatory information about the data in a graphical interface. As a first step in the modeling process, it must be defined what kind of data should be processed: Real-time data or historic data (e.g. a CSV file). In our example, we assume that the temperature sensor originates from a real-time data source. Afterwards, the communication protocol for accessing the data source needs to be selected based on several available options. In our example, the temperature sensor provides a REST interface that must be polled every second. The modeling process is semi-automatic and the system tries to guess as much as possible, like the format of data and semantic content. This is done based on the available meta data or by extracting sample data from the source. Once the adapter has been initialized, raw data is processed in the pipeline according to rules that are inferred from the model and is transformed into the virtual sensor representation. In our example, the pipeline filters out all events with a temperature higher then 350. This rule is automatically created by the framework due to the RDF description of the virtual sensor which describes the measurement range to be lower then 350. Furthermore, the raw data event is transformed into JSON. Finally, events are emitted by the adapter and put onto a message broker. They can then be consumed by processing engines, like for example StreamPipes or Apache Spark [6].

3 Event Model

This section introduces an RDFS-based event model, which is based on the Semantic Event Producer model of [7, section 7.3] and further re-uses parts of the Semantic Sensor Network Ontology [8] and the QUDT Ontology [9] for representing measurement units. We present two variations of the model, the

raw data model and the virtual sensor. The raw data model is a subset of the virtual sensor model, which contains only basic information about the data. Both models can be seen in figure 2, with the raw data model highlighted in bold. The event model is instantiated in a design phase, once a new adapter is being created by a domain expert.

The **raw data** model has two types of *data sequences*, a *Data Set* and a *Data Stream*, each of them has a grounding, the protocol and a format. Further, it has a simple event schema that consist of a *runtimeName*, for example the column name of a CSV table, and the according *runtimeType*, the type of data that is stored in the table column (e.g. String or Integer). Besides primitive types, an event schema can also describe nested structures and lists. Additionally, a measurement unit for properties can be provided (e.g. temperature measured in degrees celsius or fahrenheit).

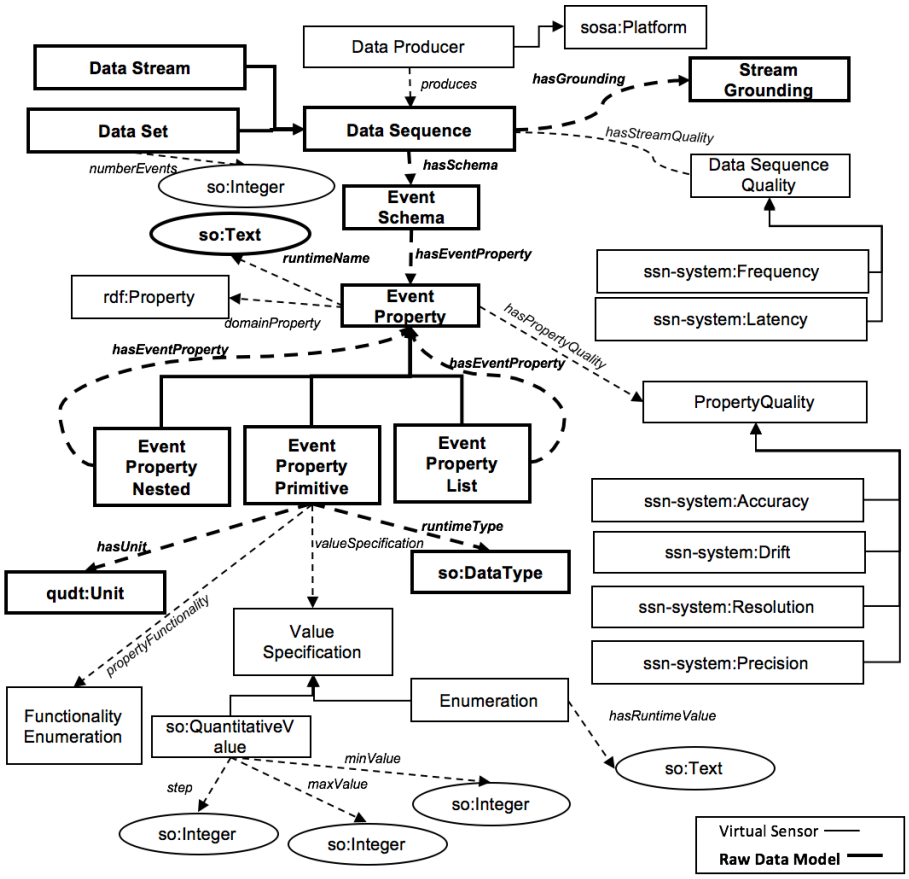

Fig. 2: Model of virtual sensor

The **virtual sensor** has a more detailed semantic description as shown in figure 2. The *Data Sequence* is produced by a *Data Producer*. Further *qualities*, like the frequency of a *Data Sequences* can be described and it has one event schema, consisting of multiple *EventProperties*. They can have different *PropertyQualities* (e.g. accuracy or precision of a sensor measurement), a *runtime name* (often the same as the runtime name of the raw data model), and a *domain property* for example 'http://schema.org/location'. Properties might also be modeled as lists or nested structures. The *EventPropertyPrimitives* have a *runtime type* and a *unit*. The *Functionality Enumeration* can be used to mark a property for example as a timestamp. Furthermore, the *ValueSpecification* can be used in order to restrict possible values of the property.

4 Adapter Architecture and Modeling

This section describes the runtime-architecture of the adapters. Adapters are modeled via a graphical user interface by a domain expert and are automatically instantiated.

Figure 3 shows the different components of the adapter architecture. In the beginning, a data converter has the main task to establish a connection to the data source, collect data and transform it into the internal raw data format. After data is available in the internal format, it is transformed into the virtual sensor data representation via the raw data pipeline. This pipeline is explained in more detail later in this section. The last component of the adapter is a broker, where virtual sensor events are sent to in order to be consumed by other applications and tools. All adapters have two interfaces, one for accessing real-time data and one for providing the schema description. Data set adapters have an additional interface to start a *data replay* and to serve data from different time-slots to multiple consumers. This is not needed for real-time data since all events are immediately emitted as they are produced.

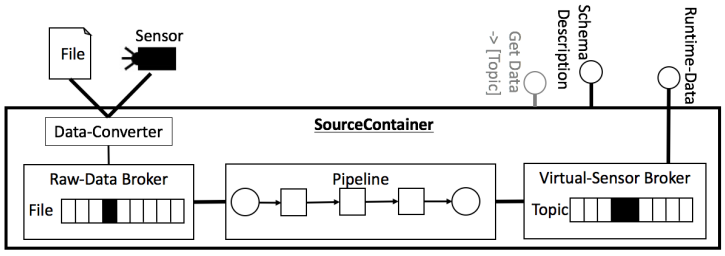

Fig. 3: Architecture of the adapter

4.1 Raw Data Pipeline

Each adapter has its own individual raw data pipeline. Those pipelines are created automatically according to the model defined by the user and consists of multiple processing agents, which are based on previous work we have implemented in StreamPipes. A user can use and configure the agents via a graphical user interface. In this work, agents are configured automatically according to the model description.

Within a pipeline, multiple data transformations can take place. In general, our idea is to use the defined semantics to automatically transform data during runtime. Such transformations could be enrichments with context information, filter out unreasonable values and transformations to ensure that the resulting events always have the same schema and quality. The first component in a pipeline is always the source providing raw-data while the last one is always a sink that writes the virtual sensor values to the resulting broker. Currently there is support for five kinds of agents in the pipeline, a structural transformer, a unit transformer, a filter, a frequency reducer and a schema agent. Agents are developed in a way that it is possible to add more at a later point in time. First, we will describe the structural transformer agent:

Structural Transformer Agent The task of the *Structural Transformer Agent* is to transform the internal raw-data representation into the structure required by the virtual sensor. This is done via mappings between the two model schema's. One example would be to a flat data structure into a nested structure or vice versa. Another example could be to flatten a *property list* into *primitive properties*. All operations are performed on the internal format and are inferred from the models based on the provided runtime name. At runtime, each data point is transformed individually in a stateless manner.

Unit Transform Agent The *Unit Transformer Agent* uses the unit information of the *EventPropertyPrimitive* to automatically provide the correct measurement unit. Users only need to model the structure of the required format for the event property and the system automatically transforms the data. This is accomplished using the QUDT Ontology [9], that provides information about different units and also contains a conversion formula for individual conversions. First, the measurement values are transformed into a standard metric, afterwards this standard metric is further transformed into the goal unit.

Filter Agent The Filter Agent filters data values out that are not compatible to the virtual sensor description. Filter rules are extracted from the semantic model, for example the *PropertyQualities* and the *ValueSpecification* as described in section 3. For instance, if a quantitative value has a modeled range from 0 to 10 and the measured value is 11, the system infers that this is a false value and can automatically remove it from the output stream. This agent ensures that data consumers can expect only semantically correct data, which reduces the probability of run-time errors.

Frequency Reducer Agent This agent changes the frequency of the data, according to the actual values of the properties. When the user activates this agent during the design phase, all values of the events are monitored during runtime. If the agent detects that values of the events do not change over a period of time, the frequency for emitting new events is reduced. With this agent, it is possible to reduce the amount of data sent over the network without loosing information.

Schema Agent Some information about the stream must not be modeled by the user, but is inferred automatically at run-time. In this case, the schema description is adapted accordingly. This agent does not transform data, it monitors runtime data and changes the schema description. Two such examples are frequency and latency of the produced events. These values are measured at runtime and are constantly updated in the schema description.

5 Related Work

Integrating semantic web technologies and big data streaming architectures is becoming more and more relevant. One example is Strider [10], which consumes data from Apache Kafka and uses Apache Spark to optimize query planning. Our approach is complementary and could be used to easily integrate new data sources into Kafka and process it with this framework.

There are also other solutions leveraging from semantic data models for streaming data from sensors like for example [11], the sensor middleware for OpenIoT [12]. The authors use the SSN ontology and also have the concept of virtual sensors. One difference is that we mainly use semantics during the design process to automatically transform data later during runtime, but we do not focus on processing RDF data at runtime.

A programming model that is related to our overall architecture are foglets [13]. This architecture consists of a central cloud computing instance and several edge nodes located closer to the sensors in the networking stack. With foglets, it is possible to distribute programs across all the computing instances and perform some processing steps on edge nodes an some in the cloud. Our approach is similar, but our programming model is more lightweight as we clean data directly on the edge nodes, where no further programming is required.

6 Conclusion and Outlook

In this paper, we presented a framework for data adapters that are capable of ingesting real-time and historic batch data into unified stream processing engines. We introduced a lightweight, RDFS-based model for raw data sources and an extended model to represent virtual sensors. These models can be instantiated by domain experts with little technical knowledge using a graphical user interface.

Based on these models, our contribution consists of a generic adapter architecture to automatically consume, pre-process and harmonize data. Our approach bridges the gap between a large variety of data sources and the processing engine that performs the actual algorithms.

In our future work, we plan to further extend our framework by supporting more protocols and formats and also extending the (semi-) automatic transformation capabilities of our raw-data pipeline.

References

1. J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.
2. P. Carbone, A. Katsifodimos *et al.*, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.
3. R. C. Fernandez, P. R. Pietzuch *et al.*, "Liquid: Unifying nearline and offline big data integration," in *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA,*, 2015.
4. S. Bischof, C. Martin, A. Polleres, and P. Schneider, *Collecting, Integrating, Enriching and Republishing Open City Data as Linked Data.* Cham: Springer International Publishing, 2015, pp. 57–75. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-25010-6_4
5. D. Riemer, F. Kaulfersch, R. Hutmacher, and L. Stojanovic, "Streampipes: Solving the challenge with semantic stream processing pipelines," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '15. New York, NY, USA: ACM, 2015, pp. 330–331. [Online]. Available: http://doi.acm.org/10.1145/2675743.2776765
6. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.
7. D. Riemer, "Methods and tools for management of distributed event processing applications," Ph.D. dissertation, Dissertation, Karlsruhe, Karlsruher Institut für Technologie (KIT),, 2016.
8. K. Taylor, S. Cox, K. Janowicz, D. L. Phuoc, A. Haller, and M. Lefrançois, "Semantic sensor network ontology," W3C, Candidate Recommendation, Jul. 2017, https://www.w3.org/TR/2017/CR-vocab-ssn-20170711/.
9. R. Hodgson, P. J. Keller, J. Hodges, and J. Spivak, "Qudt - quantities, units, dimensions and data types ontologies," Tech. Rep., 2017, http://www.qudt.org/.
10. X. Ren and O. Curé, "Strider: A Hybrid Adaptive Distributed RDF Stream Processing Engine," pp. 1–17, 2017. [Online]. Available: http://arxiv.org/abs/1705.05688
11. J.-P. Calbimonte, S. Sarni, J. Eberle, and K. Aberer, "Xgsn: An open-source semantic sensing middleware for the web of things." in *TC/SSN@ ISWC*, 2014, pp. 51–66.
12. J. Soldatos, N. Kefalakis *et al.*, "Openiot: Open source internet-of-things in the cloud," in *Interoperability and open-source solutions for the internet of things.* Springer, 2015, pp. 13–25.
13. E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems.* ACM, 2016, pp. 258–269.