StreamPipes Connect: Semantics-Based Edge Adapters for the IIoT

Philipp Zehnder(✉), Patrick Wiener, Tim Straub, and Dominik Riemer

FZI Research Center for Information Technology,
Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
{zehnder,wiener,straub,riemer}@fzi.de

Abstract. Accessing continuous time series data from various machines and sensors is a crucial task to enable data-driven decision making in the Industrial Internet of Things (IIoT). However, connecting data from industrial machines to real-time analytics software is still technically complex and time-consuming due to the heterogeneity of protocols, formats and sensor types. To mitigate these challenges, we present *StreamPipes Connect*, targeted at domain experts to ingest, harmonize, and share time series data as part of our industry-proven open source IIoT analytics toolbox StreamPipes. Our main contributions are (i) a semantic adapter model including automated transformation rules for pre-processing, and (ii) a distributed architecture design to instantiate adapters at edge nodes where the data originates. The evaluation of a conducted user study shows that domain experts are capable of connecting new sources in less than a minute by using our system. The presented solution is publicly available as part of the open source software Apache StreamPipes.

Keywords: Industrial Internet of Things · Edge processing · Self-service analytics

1 Introduction

In order to exploit the full potential of data-driven decision making in the Industrial Internet of Things (IIoT), a massive amount of high quality data is needed. This data must be integrated, harmonized, and properly described, which requires technical as well a domain knowledge. Since these abilities are often spread over several people, we try to enable domain experts with little technical understanding to access data sources themselves. To achieve this, some challenges have to be overcome, such as the early pre-processing (e.g. reducing) of the potentially high frequency IIoT data close to the sensor at the edge, or to cope with high technological complexity of heterogeneous data sources. The goal of this paper is to simplify the process of connecting new sources, harmonize data, as well as to utilize semantic meta-information about its meaning, by providing a system with a graphical user interface (GUI).

A. Harth et al. (Eds.): ESWC 2020, LNCS 12123, pp. 665–680, 2020.
https://doi.org/10.1007/978-3-030-49461-2_39

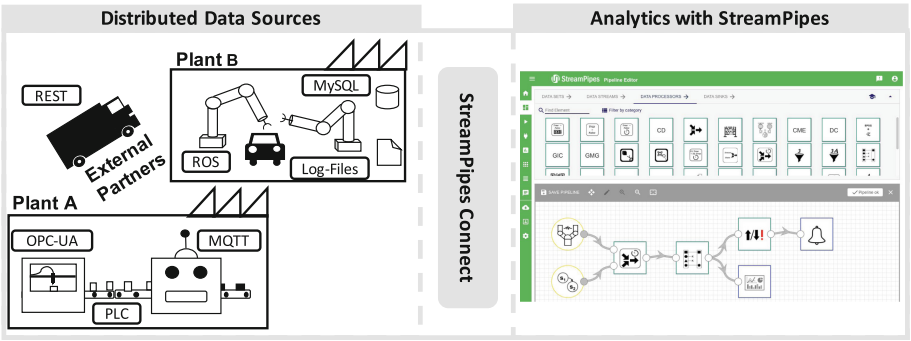

Fig. 1. Motivating scenario of a manufacturing company

Our solution, StreamPipes Connect, is made publicly available as part of the open-source, self-service data analytics platform *Apache StreamPipes (incubating)*[1]. StreamPipes [14] provides a complete toolbox to easily analyze and exploit a variety of IoT-related data without programming skills. Therefore, it leverages different technologies especially from the fields of big data, distributed computing and semantic web (e.g. RDF, JSON-LD). StreamPipes is widely adopted in the industry and is an incubating project at the Apache Software Foundation.

Figure 1 shows a motivating scenario of a production process in a company with several plants. It further illustrates the potentially geo-distributed heterogeneous data sources that are available in such a company. However, the challenge is how to enable domain experts to connect and harmonize these distributed heterogeneous industrial streaming data sources. First we show how our approach differs from existing related work in Sect. 2. To cope with the distributed setting we leverage a master worker paradigm with a distributed architecture (Sect. 3). Adapters are deployed on edge devices located within a close proximity to sources, to early filter and transform events. We use a semantics based model to describe adapters and to employ transformation rules on events (Sect. 4). The model covers standard formats and protocols as well as the possibility to connect proprietary data sources. In Sect. 5, the implementation of our approach and the GUI is explained in detail. We present results of a conducted user study to evaluate the usability of our system, in addition to the performance tests carried out in Sect. 6. Lastly Sect. 7 concludes our work and presents an outline of planned future work.

[1] https://streampipes.apache.org/.

2 Related Work

Data flow tools with a GUI are commonly used to process and harmonize data from various sources. Applications like Talend[2], or StreamSets[3] can be used for Extract, Transform, Load (ETL) tasks, wich is a well elaborated field where the goal is to gather data from many heterogeneous sources and store it in a database. Using such tools still requires a lot of technical knowledge, especially because they are not leveraging semantic technologies to describe the meaning of data. Another tool in this field is Node-RED[4], which describes itself as a low-code solution for event-driven applications. Node-RED is designed to run on a single host. However, our approach targets distributed IIoT data sources like machines or sensors. Therefore, data can be processed directly on edge devices, potentially reducing network traffic. There are also approaches leveraging semantic technologies for the task of data integration and harmonization. WInte.r [9] supports standard data formats, like CSV, XML, or RDF, further it supports several strategies to merge different data sets with a schema detection and unit harmonization. In contrast to our approach, it focuses on data sets instead of IIoT data sources. The goal of Spitfire [12] is to provide a Semantic Web of Things. It focuses on REST-like sensor interfaces, not on the challenge of integrating sensors using industrial protocols and high-frequency streaming data, that require local preprocessing. The Big IoT API [6] enables interoperability between IoT platforms. Thus the paper has a different focus, we focus on domain experts to connect data, especially from IIoT data sources.

Distributed architectures like presented in [8] are required to cope with the distributed nature of IIoT data sources. In the paper, a lightweight solution to ease the adoption of distributed analytics applications is presented. All raw events are stored in a distributed storage and are later used for analytics. The authors try to adopt a very lightweight approach and do not describe the semantics of events or transform them. In our approach, data is transformed and harmonized directly in the adapter at the edge. This eases the analytics tasks downstream usually performed by a (distributed) stream processing engine, such as Kafka Streams. Such engines provide solutions to connect to data sources with Kafka Connect[5]. It is possible to create connectors that publish data directly to Kafka. They provide a toolbox of already integrated technologies, such as several databases or message brokers. Still, a lot of configuration and programming work is required to use them.

Other industry solutions to cope with the problem of accessing machine data are to build custom adapters, e.g. with Apache PLC4X[6]. This requires a lot of development effort and often is targeted at a specific use case. We leverage such tools to enable an easy to configure and re-usable approach. Another way to

[2] https://www.talend.com/.
[3] https://streamsets.com/.
[4] https://nodered.org/.
[5] https://www.confluent.io/connectors/.
[6] https://plc4x.apache.org/.

access machine data is to use a unified description, like the Asset Administration Shell (AAS) [2]. It is introduced by the Platform Industry 4.0 and provides a unified wrapper around assets describing its representation and technical functionality. There are also some realizations of the concept, as described in [16]. In our approach we try to automatically create an adapter by extracting sample data and meta-data from the data source. Thus, this allows us to work with data sources that do not have a specific description like the AAS.

3 Architecture

The main design decisions for our architecture are based on the goal of creating a system for both small, centralized as well as large, distributed environments. Therefore, we decided to implement a master/worker paradigm, where the master is responsible for the management and controlling of the system and the workers actually access and process data. To achieve this, we need a lightweight approach to run and distribute services. Container technologies offer a well suited solution and are particularly suitable for edge and fog processing scenarios [7]. Figure 2 provides an overview of our architecture showing the data sources and the compute units located close to the sources, running the services of the system. The StreamPipes backend communicates with the master, which manages all the worker containers, as well as the adapter instances running in the workers. For the communication between the individual components we use JSON-LD. The master persists the information about the workers and running adatpers in a triple store.

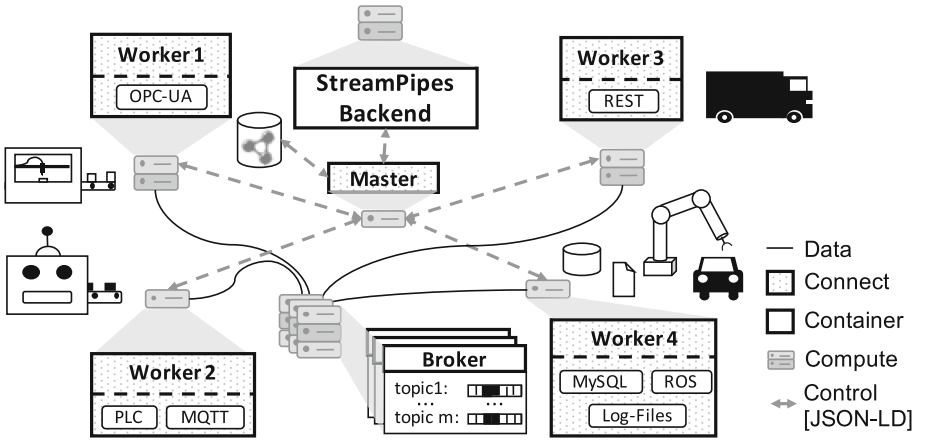

Fig. 2. Architectural overview of our system

Once a new worker is started, it is registered at the master, providing information which adapter types (e.g. PLC, MQTT) are supported. When an adapter

instance is instantiated to connect a new machine, the data is directly forwarded to a distributed message broker, as shown in Fig. 2. New worker instances can be added during runtime to extend the system and the master schedules new adapters accordingly. The master coordinates and manages the system. For the transmission of the harmonized data we rely on already existing broker technologies, e.g. Apache Kafka.

4 Adapters

The adapter model is the core of our approach and provides a way to describe time series data sources. Based on this model, adapters are instantiated, to connect and harmonize data according to pre-processing rules applied to each incoming event. Such adapter descriptions are provided in RDF serialized as JSON-LD.

4.1 Adapter Model

Figure 3 shows our semantic adapter model. The *Adapter* concept is at the core of the model. Each adapter has a *StreamGrounding* describing the protocol and format used to publish the harmonized data. Additionally to sending unified data to a message broker, adapters are capable of applying *Transformation Rules*.

DataSets and *DataStreams* are both supported by the model. For a better overview of the Figure, we present a compact version of the model with the notation {*Stream, Set*}, meaning there is one class for streams and one for sets. From a modeling and conceptual point of view, there is no difference in our approach between the two types. We treat data sets as bounded data streams, which is why we generally refer to data streams from here onwards.

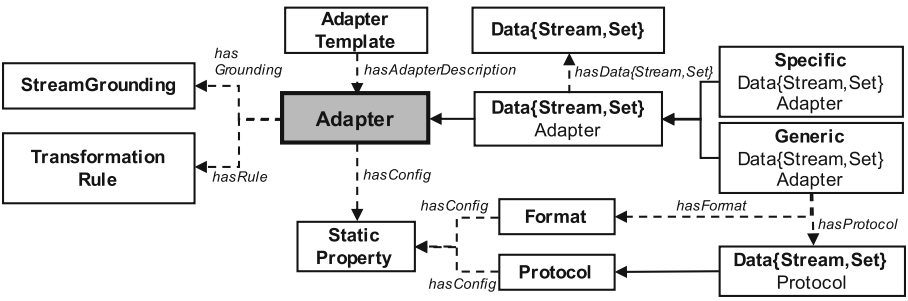

Fig. 3. Core of our adapter model

Further, there are two types of Data Stream Adapters, *GenericDataStreamAdapters* and *SpecificDataStreamAdapters*. A GenericDataStreamAdapter consists of a combination of a *DataStreamProtocol* (e.g. MQTT), responsible for

connecting to data sources and formats (e.g. JSON) that are required to convert the connected data into the internal representation of events. Since not all data sources comply with those standards (e.g. PLC's, ROS, OPC-UA), we added the concept of a *SpecificDataStreamAdapter*. This can also be used to provide custom solutions and implementations of proprietary data sources. User configurations for an adapter can be provided via *StaticProperties*. They are available for *Formats*, *Protocols*, and *Adapters*. There are several types of static properties, that allow to automatically validate user inputs (e.g. strings, URLs, numeric values). Configurations of adapters (e.g., protocol information or required API keys) can be stored in *Adapter Templates*, encapsulating the required information. Listing 1.1 shows an instance of a *GenericDataStreamAdapter*, with MQTT as the protocol and JSON as a format.

```
@prefix sp: <https://streampipes.apache.org/vocabulary/v1/> .


<sp:adapter1>
  a sp:GenericDataStreamAdapter ;
  rdfs:label "Temperature Sensor" ;
  sp:hasProtocol <sp:protocol/stream/mqtt> ;
  sp:hasFormat <sp:format/json> ;
  sp:hasDataStream <sp:dataStream1> ;
  sp:hasRule <sp:transformationrule1> .

<sp:protocol/stream/mqtt>
  a sp:DataStreamProtocol ;
  rdfs:label "MQTT" ;
  sp:config <sp:staticproperty1>, <sp:staticproperty2> .

<sp:format/json>
  a sp:Format ;
  rdfs:label "JSON" .

<sp:staticproperty1>
  a sp:FreeTextStaticProperty ;
  rdfs:label "Broker URL" ;
  sp:hasValue "tcp://mqtt-host.com:1883" .

<sp:staticproperty2>
  a sp:FreeTextStaticProperty ;
  rdfs:label "Topic" ;
  sp:hasValue "sensor/temperature" .
```

Listing 1.1. Example for a MQTT adapter instance

4.2 Transformation Rule Model

Oftentimes, it is not sufficient to only connect data, it must also be transformed, reduced, or anonymized. Therefore we introduce transformation rules, visualized in Fig. 4, to either change the value of properties, schema, or the stream itself.

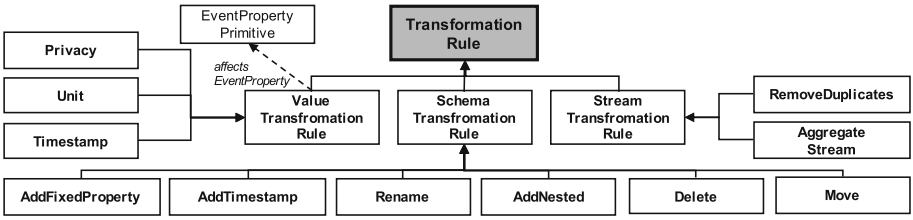

Fig. 4. Model of the transformation rules with all Value-, Schema-, and StreamTransformationRules

Our approach uses transformation rules to describe the actual transformation of events. Based on these rules, pre-processing pipelines are automatically configured in the background, which run within an adapter instance. The following table presents an overview of an extensible set of transformation rules, which are already integrated.

Scope	Rule	Example
Schema	*Add Fix Property*	{} → {"id": "sensor5"}
	Add Nested	{} → {"a": {}}
	Move	{"a": {"b": 1}} → {"a": {}, "b": 1}
	Add Timestamp	{} → {"timestamp": 1575476535373}
	Rename	{"old": 1} → {"new": 1}
	Delete	{"a": 1} → {}
Value	*Privacy* (SHA-256)	{"name": "Pia"} → {"name": "ca9..."}
	Unit ($°C \rightarrow °F$)	{"temp": 41} → {"temp": 5}
	Timestamp	{"time": "2019/12/03 16:29"} →{"time": 1575476535373}
Stream	*Remove Duplicates*	{"a": 1},...,{"a": 1} → {"a": 1}
	Aggregate	{"a": 2},...,{"a": 1} → {"a": 1.5}

Listing 1.2 shows an example instance of the *UnitTransformationRule*. It is part of the adapter model in Listing 1.1 and describes how to transform the temperature value form the unit degree celsius into degree Fahrenheit. All instances of the rules look similar. The configuration parameters of the individual rules differ, for example instead of the fromUnit and toUnit, the rename rule contains the old and the new runtime name.

```
<sp:transformationrule1>
  a sp:UnitTransformRule ;
  sp:runtimeKey "temperature" ;
  sp:fromUnit "http://qudt.org/vocab/unit#DegreeFahrenheit" ;
  sp:toUnit "http://qudt.org/vocab/unit#DegreeCelsius" .
```

Listing 1.2. Unit transformation rule instance example

4.3 (Edge-) Transformation Functions

Events of connected data sources are transformed directly on the edge according to the introduced transformation rules, by applying transformation functions, event by event. A function takes an *event* e and *configurations* c as an input and returns a transformed *event* e'. The model is expandable and new features can be added by a developer. An instance of an adapter contains a set of functions which are concatenated to a pre-processing pipeline. Equation (1) shows how an event is transformed by multiple functions. Each function represents a transformation rule from our model. To ensure that the transformations are performed correctly the rules must be applied in a fixed order. First the schema, then the value, and last the stream transformations.

$$F(e) = f_n(f_{...}(f_1(e, c), ...), c) = e' \tag{1}$$

The unit transformation function for example takes the property name, the original unit and the new unit as a configuration input. Within the function the value of the property is transformed according to the factors in the qudt ontology[7]. Figure 5 shows a complete pre-processing pipeline of our running example. On the left the raw input *event* e is handed to the first function f_1 that changes the schema. The result of each function is handed to the next function in addition to the configurations. In the end, the final *event* e' is sent to the defined *StreamGrounding* of the adapter.

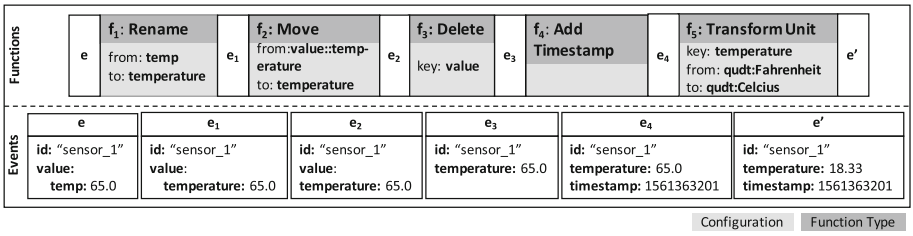

Fig. 5. Example of a pre-processing pipeline

[7] https://www.qudt.org/.

5 Implementation

We integrated our implementation into the open source software Apache StreamPipes (incubating), which is publicly available on GitHub[8].

5.1 Adapter Marketplace

Figure 6 shows the adapter marketplace containing an overview of all protocols, specific adapters, and adapter templates. Currently, we integrated 25 different adapters and we continually integrate new ones. For streaming protocols, PLCs (e.g. Siemens S7), OPC-UA[9], ROS [13], MQTT [3], FTP, REST (iterative polling), MySQL (subscribing to changes), InfluxDB, Kafka, Pulsar are integrated. Further we support several data set protocols like files (can be uploaded), HDFS, FTP, REST, MySQL, InfluxDB. Additionally to those generic adapters, we have integrated several open APIs, like openSenseMap[10] resulting in specific adapters. This number is also constantly growing, since adapters can be stored and shared as adapter templates. Templates are serialized into JSON-LD, that can be exported and imported into other systems. They are also listed in the data marketplace.

5.2 Adapter Modeling Process

Once a user selects the adapter that should be connected, a guided configuration process is started. This process is the same for data sets and data streams and just differs slightly between generic and specific adapters. We illustrate the

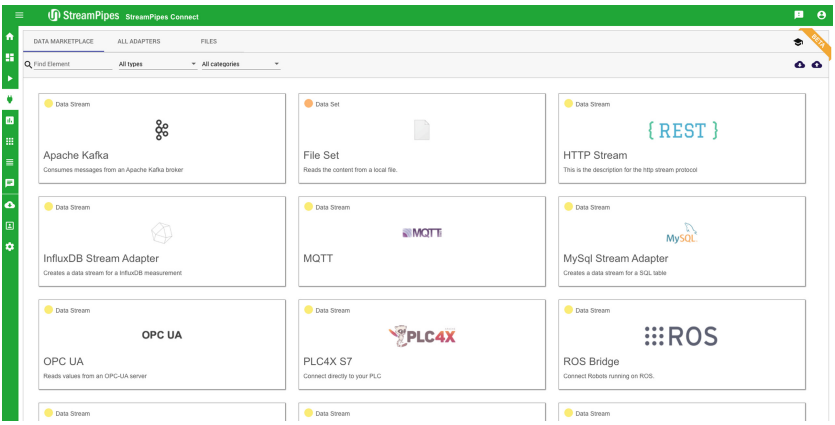

Fig. 6. Overview of the data marketplace

[8] https://github.com/apache/incubator-streampipes.
[9] https://opcfoundation.org/.
[10] https://opensensemap.org/.

modeling process of a generic adapter with the example of a temperature sensor introduced in Fig. 5. The sensor values are provided over MQTT and are serialized in JSON:

1. **Select adapter/protocol:** First a user must select the specific adapter or protocol form the marketplace, shown in Fig. 6.
2. **Configure adapter/protocol:** In the next step a configuration menu is presented to the user. In ❶ of Fig. 7 an example for the MQTT protocol is shown. The broker URL, optional credentials for authentication and the topic must be provided.
3. **Configure format (optional):** For generic adapters additionally the format must be configured. In our example a user must select JSON.
4. **Refine event schema:** So far the technical configurations to connect data sources were described, now the content of the events must be specified. Figure 7 in ❷ shows the event schema. Users can add, or delete properties, as well as change the schema via a drag-and-drop user interface. Further shown in ❸ additional information can be added to individual properties, like a description, the domain property, or the unit. Based on the user interaction the transformation rules are derived in the background.
5. **Start adapter:** In the last step a name for the adapter must be provided. Additionally a description or an icon can be added. In this step it is also possible to define a maximum frequency for the resulting data stream, or to filter duplicate events. Again, rules are derived from the user interaction. Users just interact with the GUI and the system creates the rules, resulting in an intuitive way of interacting with the system without worrying about the explicit modeling of the rules.

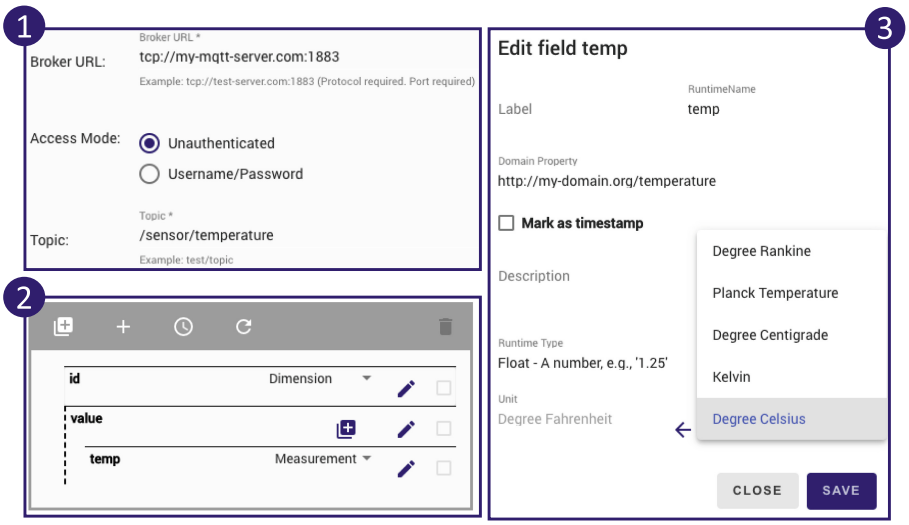

Fig. 7. Screenshots of adapter modeling process

5.3 Semantics Based User Guidance

We try to use the semantic model and meta-data as much as possible to help and guide the user through the system. All user inputs are validated in the GUI according to the information provided in the adapter model (e.g. ensure correct data types or URIs). Additionally, the system uses information of the data sources, when available, during the configuration steps 2./3., described in the previous section. Unfortunately, the usefulness of those interfaces highly depends on the selected adapter/protocol, since not all provide the same amount of high quality information. For example, some message brokers provide a list of available topics. Other technologies, like PLCs often have no interface like that and the users have to enter such information manually. But still, this user input is checked and when an error occurs during the connection to the source a notification with the problem is provided to the user.

Furthermore, the schema of the events is guessed by reading sample data from the source. Once the endpoint of the source is connected, we establish a connection to gather some sample data. Based on this data a guess of the schema is provided and suggested to the user in the GUI. The realization for the individual implementations of this schema guess is again very different. For CSV files for example it depends if they have a header line or not. For message brokers sending JSON a connection has to be established to gather several events to get the structure of the JSON objects. Other adapters like the one for OPC-UA can leverage the rich model stored in the OPC server to already extract as much meta-information as possible. All of this information is harmonized into our semantic adapter model, where we also integrate external vocabularies, and presented in the GUI to the user. Users are able to refine or change the model.

Also on the property level we try to leverage the semantics of our model to easily integrate different representations of timestamps, by providing a simple way to harmonize them to the internal representation of UNIX timestamps. Another example are unit transformations. Based on the qudt ontology only reasonable transformations are suggested to the user.

6 Evaluation

In our evaluation we show that domain experts with little technical knowledge are able to connect new sources. Additionally, we present performance results of adapters and where the system is already deployed in production.

6.1 User Study

Setup: For our user study, we recruited 19 students from a voluntary student pool of the Karlsruhe Institute of Technology (KIT) using hroot [4]. The user study took place at the Karlsruhe Decision & Design Lab (KD^{2}Lab)[11] at the KIT. The overall task was to connect two data sources with measurements of environment sensors as a basis, to create a live air quality index, similar to the one in [1]. Since most of the participants did not have a technical background and never worked with sensor data before, we gave a 10 min introduction about the domain, the data sources, what it contains (e.g. particulate matter PM2.5/PM10, nitrogen dioxide NO_2, ...), and how an air quality index might look like. After that, the participants went into an isolated cabin to solve the tasks on their own, without any further assistance by the instructors. As a first task, they had to connect data from the openSenseMap API [11], an online service for environmental data. The goal of the second task was to connect environmental data from official institutions, therefore data provided by the 'Baden-Wuerttemberg State Institute for the Environment, Survey and Nature Conservation' was used. This data is produced by officially standardized air measuring stations distributed all over the state. After finishing the two tasks, the participants were forwarded to an online questionnaire, where they had to answer several questions to assess how usable the system was in their experience. For the questions, we used three standardized questionnaires as well as additional questions. To ensure that the participants answer the questions carefully, we added control questions to the questionnaire. Three participants answered those control questions wrong, resulting in a total of 16 valid answers.

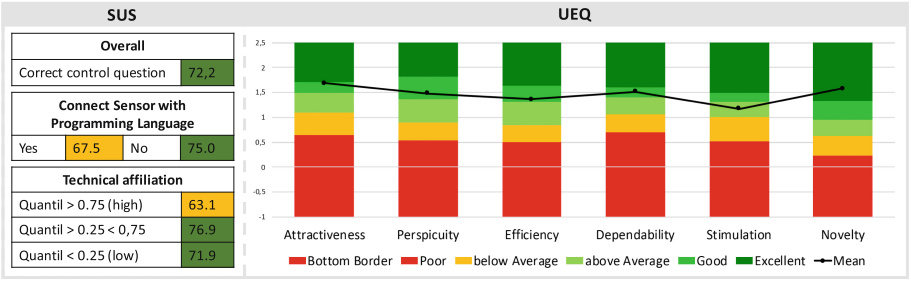

Fig. 8. Results of SUS & UEQ

Results: First, we present the results of the System Usability Scale (SUS) [5], which measures how usable a software system is by comparing results to the average scores of 500 websites[12]. A score above 71.4 is considered as good result. We use the same colors to indicate how well the score is compared to the other

[11] http://www.kd2lab.kit.edu/.
[12] https://www.trymyui.com/sus-system-usability-scale.

systems. On the left of Fig. 8, the overall result of 72.2 can be seen. Since we have a high variance of technical expertise within our participants we grouped the results according to the technical experience. First we grouped them into two groups, whether they stated to be able to connect sensors with any programming language of their choice or not. Participants not able to develop a connector for sensors with a programming language find the system more useful (good system, mean: 75.0) than participants who are able to connect a sensor with a programming language of their own choice (acceptable system, mean: 67.5). Second, we grouped them according to their technological affinity from high to low. For that, we adopted the items of the Technology Readiness Index (TRI) [10] in order to frame the questions on the expertise in using programming IDE's and data tools. We can use this as a control to measure how affine participants are in using technologies (e.g. IDE's). Participants with a high technology affinity (quantile > 0.75) find the system not as useful as less technology affine participants, but still acceptable (mean: 63.1). Participants with an average technology affinity find the system the most useful (good system: mean: 76,9). Participants with a low technology affinity (quantile < 0.25) find the system good as well, however a bit less useful as the average class (mean: 71,9). This is in line with the assumption, that such a tool is especially useful for non-technical users. The SUS gives the tool a rating of a good system. The participants used the system for the first time and only for a duration of 15 to 20 min. In this respect, this is already a very good score and it is likely to assume that the score would be higher when more experienced users would have participated.

For the second questionnaire, the User Experience Questionnaire (UEQ) was chosen [15]. It consists of six categories: Attractiveness, Perspicuity, Efficiency, Dependability, Stimulation, and Novelty. For each of these categories, a Likert scale is provided to indicate how good the system is compared to other systems evaluated with the UEQ. Figure 8 shows the results of the UEQ on the right. All the results of the individual categories are above average. The results of the categories Attractiveness, Perspicuity, Efficiency, and Dependability are considered as good. The result of the Novelty of the system is even rated as excellent. The figure also reveals that the results of all categories are equally good meaning we do not have to focus on a single aspect. It also suggests that there is still room for further improvement, but for a first user study the results are already very promising. Together with the results from the SUS, this means that the system is not only usable (i.e. fulfils its purpose) but also gives a good experience when using it (i.e. fun experience).

Additionally, we added own questions to the questionnaire to get some information which is especially relevant for our work. To see how technical the students were, we asked them whether they are able to connect new sensors in a programming language of their choice or not. Just 5 of the participants answered with yes, while 11 gave a negative answer. This indicates we had a good mix of technical experience of the participants, as our system focuses on less technical users with little to no programming experience. We asked the participants, if they think, once they are more familiar with the system, they are able to

connect new data sources in under one minute. 14 answered with yes and 2 with no. This shows that our approach is simple to use and efficient, as even the less technical participants state they can connect new data sources in under one minute, which is usually a technical and time-consuming task. Regarding the question whether they think they are capable of connecting new real-time sensor data with our system, all of the participants answered with yes. This means all participants are capable of creating new adapters with the system. We also monitored the interaction of the users with the system to find out how long they approximately needed to complete the individual tasks. The result was that users took between 3 to 5 min for each task. Overall, the results of the user study show that StreamPipes Connect is already rated as a good system, which can be used by domain experts to quickly connect new data sources.

6.2 Performance Evaluation

Setup: For the evaluation we connected the events of the joint states of a robot arm via ROS. The frequency of the data stream is 500 Hz and the event size is 800 Bytes. This data was connected and processed with the ROS adapter without any delays. To discover the limits of our system we created an adapter with a configurable data generator. Therefore, we used the temperature event and transformed it with the same rules as in our example in Fig. 5. For the test setup we used a server running the StreamPipes backend and two different commonly used edge devices for the worker instance. We used a Raspberry Pi 4 and an Intel NUC. To test the maximum performance of an adapter within a worker we produced events as fast as the worker could process them. For each device we ran 6 different set-ups, all with a different lengths of the pipeline shown in Fig. 5.

Results: Figure 9 shows the results of the performance test. Each test ran 15 times and the mean of sent Events per second is plotted in the chart. For the NUC we produced 10.000.000 events per test and for the Raspberry Pi 5.000.000 events.

The results of the figure show that if no pre-processing pipeline is used the events are transmitted the fastest and the longer the pre-processing pipeline is, the less events are processed. The only exception is the delete function, which removes a property of the event and thus increases the performance. The NUC performs significantly better then the raspberry Pi, but for many real-world use cases a Pi is still sufficient, since it also processes 54.000 events per second (with no pre-processing function). The add timestamp and transform unit functions have an higher impact on the performance than the other tested functions.

6.3 Usage

Apache StreamPipes (incubating) was developed as an open source project over the last couple of years by the authors of this paper at the FZI Research Center for Information Technology. Since November 2019, we transitioned the tool to the Apache Software Foundation as a new incubating project.

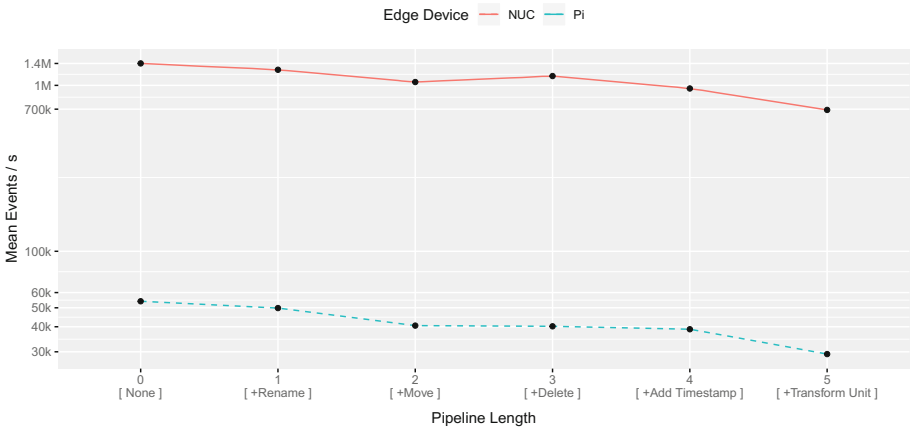

Fig. 9. Performance test results over 15 test runs

We successfully deployed StreamPipes in multiple projects in the manufacturing domain. One example is condition monitoring in a large industrial automation company. We connected several robots (Universal Robots) and PLCs to monitor a production process and calculate business-critical KPIs, improving the transparency on the current health status of a production line.

7 Conclusion and Future Work

In this paper, we presented StreamPipes Connect, a self-service system for ingestion and harmonization of IIoT time series data, developed as part of the open source IoT toolbox Apache StreamPipes (incubating).

We presented a distributed, event-based data ingestion architecture where services can be directly deployed on edge devices in form of worker nodes. Workers send real-time data from a variety of supported industrial communication protocols (e.g., PLCs, MQTT, OPC-UA) to a centralized message broker for further analysis.

Our approach makes use of an underlying semantics-based adapter model, which serves to describe data sources and to instantiate adapters. Generated adapters connect to the configured data sources and pre-process data directly at the edge by applying pipelines consisting of user-defined transformation rules. In addition, we further presented a graphical user interface which leverages semantic information to better guide domain experts in connecting new sources, thus reducing development effort.

To achieve the goal of providing a generic adapter model that covers the great heterogeneity of data sources and data types, the flexibility of semantic technologies was particularly helpful. Especially the reuse of vocabularies (e.g. QUDT) facilitates the implementation significantly. The user study has shown us that modeling must be easy and intuitive for the end user.

For the future, we plan to further support users during the modeling process by recommending additional configuration parameters based on sample data of the source (e.g. to automatically suggest message formats).

References

1. Air quality in Europe (2017). https://doi.org/10.2800/850018
2. Adolphs, P., et al.: Structure of the administration shell. Continuation of the development of the reference model for the Industrie 4.0 component. ZVEI and VDI, Status Report (2016)
3. Banks, A., Gupta, R.: MQTT version 3.1.1. OASIS Stand. **29**, 89 (2014)
4. Bock, O., Baetge, I., Nicklisch, A.: hroot: Hamburg registration andorganization online tool. Eur. Econ. Rev. **71**, 117–120 (2014). https://doi.org/10.1016/j.euroecorev.2014.07.003. http://www.sciencedirect.com/science/article/pii/S0014292114001159
5. Brooke, J., et al.: SUS-A quick and dirty usability scale. In: Brooke, J., Jordan, P.W., Thomas, B., Weerdmeester, B.A., McClelland, I.L. (eds.) Usability Evaluation Industry, pp. 184–194. CRC Press, London (1996)
6. Bröring, A., et al.: The big IoT API - semantically enabling iot interoperability. IEEE Pervasive Comput. **17**(4), 41–51 (2018). https://doi.org/10.1109/MPRV.2018.2873566
7. Ismail, B.I., et al.: Evaluation of docker as edge computing platform, pp. 130–135. IEEE (2015). https://doi.org/10.1109/ICOS.2015.7377291
8. Kirmse, A., Kraus, V., Hoffmann, M., Meisen, T.: An architecture for efficient integration and harmonization of heterogeneous, distributed data sources enabling big data analytics. In: Proceedings of the 20th International Conference on Enterprise Information Systems, Funchal, Madeira, Portugal, pp. 175–182. SCITEPRESS - Science and Technology Publications (2018). https://doi.org/10.5220/0006776701750182
9. Lehmberg, O., Brinkmann, A., Bizer, C.: WInte.r - a web data integration framework, p. 4 (2017)
10. Parasuraman, A.: Technology readiness index (TRI) a multiple-item scale to measure readiness to embrace new technologies. J. Serv. Res. **2**(4), 307–320 (2000)
11. Pfeil, M., Bartoschek, T., Wirwahn, J.A.: Opensensemap-a citizen science platform for publishing and exploring sensor data as open data. In: Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings. vol. 15, p. 39 (2015)
12. Pfisterer, D., et al.: Spitfire: toward a semantic web of things. IEEE Commun. Mag. **49**(11), 40–48 (2011). https://doi.org/10.1109/MCOM.2011.6069708
13. Quigley, M., et al.: ROS: an open-source robot operating system. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe, Japan (2009)
14. Riemer, D., Kaulfersch, F., Hutmacher, R., Stojanovic, L.: Streampipes: solving the challenge with semantic stream processing pipelines. In: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, pp. 330–331. ACM (2015)
15. Schrepp, M., Hinderks, A., Thomaschewski, J.: Applying the user experience questionnaire (UEQ) in different evaluation scenarios. In: Marcus, A. (ed.) DUXU 2014. LNCS, vol. 8517, pp. 383–392. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07668-3_37
16. Tantik, E., Anderl, R.: Integrated data model and structure for the asset administration shell in Industrie 4.0. Proc. CIRP **60**, 86–91 (2017)